

577.1

Linux Incident Response and Analysis

Licensed To: Kaleb Sego <kaleb_sego@outlook.com> June 12, 2025



PLEASE READ THE TERMS AND CONDITIONS OF THIS COURSEWARE LICENSE AGREEMENT ("CLA") CAREFULLY BEFORE USING ANY OF THE COURSEWARE (DEFINED BELOW) ASSOCIATED WITH THE SANS INSTITUTE COURSE. THIS IS A LEGAL AND ENFORCEABLE CONTRACT BETWEEN YOU (THE "USER") AND THE ESCAL INSTITUTE OF ADVANCED TECHNOLOGIES, INC. /DBA SANS INSTITUTE ("SANS INSTITUTE") FOR THE COURSEWARE. BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA.

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

BY ACCESSING THE COURSEWARE, USER AGREES TO BE BOUND BY THE TERMS OF THIS CLA. USER FURTHER AGREES 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 User does not agree to the terms of this CLA, User should not access the Courseware. User may return the Courseware to SANS Institute for a 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. User may not sell, rent, lease, trade, share, or otherwise transfer the Courseware in any way, shape, or form to any person or entity without the express written consent of SANS Institute. Additionally, User may not upload, submit, or otherwise transmit Courseware to any artificial intelligence system, platform, or service for any purpose, regardless of whether the intended use is commercial, educational, or personal, without the express written consent of SANS Institute. User agrees that the failure to abide by this provision would cause irreparable harm to SANS Institute that is impossible to quantify. User therefore agrees to a base liquidated damages amount of \$5000.00 USD per item of Courseware infringed upon or fraction thereof. In addition, the base liquidated damages amount shall be doubled for any Courseware less than a year old as a reasonable estimation of the anticipated or actual harm caused by User's breach of the CLA. Both parties acknowledge and agree that the stipulated amount of liquidated damages is not intended as a penalty, but as a reasonable estimate of damages suffered by SANS Institute due to User's breach of the CLA.

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. A written amendment or addendum to this CLA that is executed by SANS Institute and User may accompany this Courseware.

SANS Institute may suspend and/or terminate User's access to and require immediate return of any Courseware in connection with any (i) material breaches or material violation of this CLA or general terms and conditions of use agreed to by User, (ii) technical or security issues or problems caused by User that materially impact the business operations of SANS Institute or other SANS Institute customers, or (iii) requests by law enforcement or government agencies.

SANS Institute 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:

The Apple® logo and any names of Apple products displayed or discussed in this book are registered trademarks of Apple, Inc.

PMP® and PMBOK® are registered trademarks 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.

VMware Workstation Pro®, VMWare Workstation Player®, VMWare Fusion®, and VMware Fusion Pro® are registered trademarks of VMware, Inc. Used with permission.

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

Courseware licensed to User under this CLA may be subject to export laws and regulations of the United States of America and other jurisdictions. User warrants he or she is not listed (i) on any sanction programs list maintained by the U.S. Office of Foreign Assets Control within the U.S. Treasury Department ("OFAC"), or (ii) denied party list maintained by the U.S. Bureau of Industry and Security within the U.S. Department of Commerce ("BIS"). User agrees to not allow access to any Courseware to any person or entity in a U.S. embargoed country or in violation of a U.S. export control law or regulation. User agrees to cooperate with SANS Institute as necessary for SANS Institute to comply with export requirements and recordkeeping required by OFAC, BIS or other governmental agency.

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

FOR577.I

Linux Incident Response and Threat Hunting



Linux Incident Response and Analysis

© 2024 SANS Institute | All Rights Reserved | Version J02_01

Author:

Taz Wake

twake@sans.org

<https://twitter.com/tazwake>

DIGITAL FORENSICS

 FOR498 Digital Acquisition and Rapid Triage GBFA	 FOR500 Windows Forensic Analysis GCFE
 FOR518 Mac and iOS Forensic Analysis & Incident Response GIME	 FOR585 Smartphone Forensic Analysis In-Depth GASF

INCIDENT RESPONSE & THREAT HUNTING

 FOR508 Advanced Incident Response, Threat Hunting & Digital Forensics GCFA	 FOR509 Enterprise Cloud Forensics and Incident Response GCFR	 FOR528 Ransomware and Cyber Extortion
 FOR572 Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response GNFA	 FOR577 LINUX Incident Response and Threat Hunting	 FOR578 Cyber Threat Intelligence GCTI
 FOR608 Enterprise-Class Incident Response & Threat Hunting GEIR	 FOR610 REM: Malware Analysis Tools & Techniques GREM	 FOR710 Reverse-Engineering Malware: Advanced Code Analysis
 SEC504 Hacker Tools, Techniques & Incident Handling GCIH		

This page intentionally left blank.

Course Material

- You should receive:
 - Books 1–5
 - Workbook
 - ISO-A (course VM, including initial evidence)
 - ISO-B (additional evidence for capstone)
- Course Dropbox link: <https://for577.com/577-dropbox>
 - Updates and reference material



This page intentionally left blank.

FOR577 Course Agenda



Section 1: Linux Incident Response and Threat Hunting

Section 2: Disk Analysis and Evidence Collection

Section 3: Linux Logging and Log Analysis

Section 4: Live Response and Volatile Data

Section 5: Advanced IR Techniques

Section 6: Capstone

This page intentionally left blank.

FOR577.I

Linux Incident Response and Threat Hunting



Incident Response and Threat Hunting

SANS | DFIR

FOR577 | Linux Incident Response and Threat Hunting

5

This page intentionally left blank.

Section I: Incident Response and Threat Hunting

Agenda

Why Incident Response Is Needed

The Incident Response Process

SRL Skunkworks

Linux Introduction

Package Management

Threat Intelligence and Host-based Threat Hunting



FOR577 | Linux Incident Response and Threat Hunting 6

This page intentionally left blank.

Why We Need Incident Response

Detection is improving, but attacks still get through.

About 13% of malware targets Linux.

Good News

Bad News

Median Dwell Time

416 > 16

Days in 2011

Days in 2022

Source: Mandiant M-Trends 2023

This is mostly because of ransomware.

Ransomware

Median
9
Days

Other Investigations

Median
17
Days

SANS | DFIR

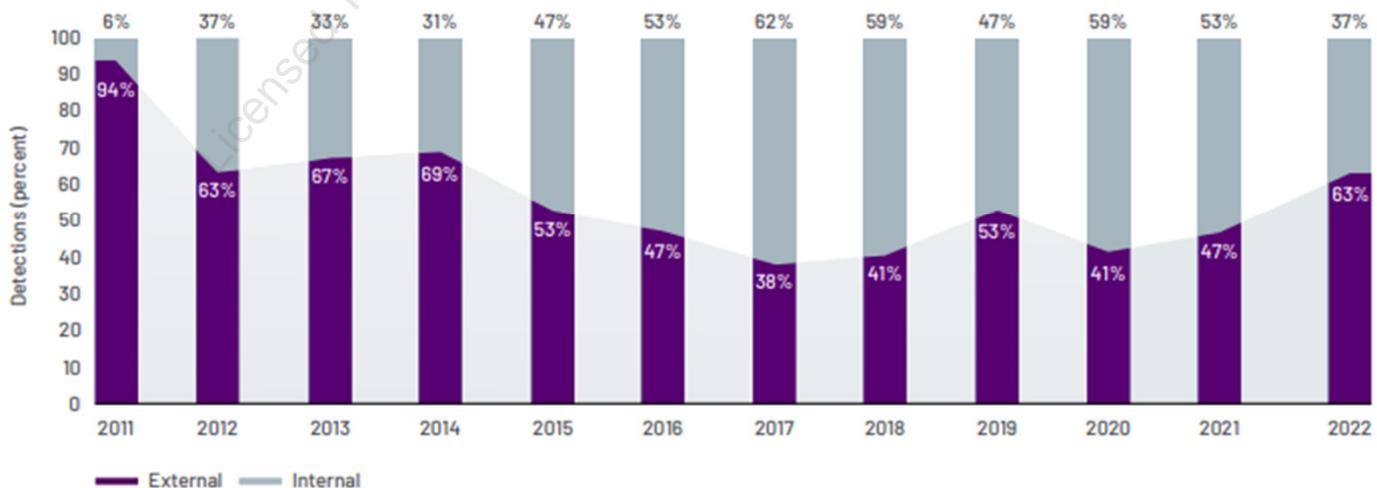
FOR577 | Linux Incident Response and Threat Hunting

7

At a very basic level, we need incident response because a sufficiently determined adversary will **always** get through our defenses. Incident response is how we build and execute a plan about what to do when this happens.

This doesn't mean we are making it easy for them, and it is a myth to think the attackers have an advantage. It does, however, mean that we need to be realistic about the threats we face and the measures we need to take. Security, detection, and response are constantly evolving and improving. A decade ago, things weren't great. Most attacks weren't even detected by the victims themselves. Although this has improved over time, in recent years, unfortunately, the trend is heading back toward external detection (see chart below). This creates a situation where a lot of our incident response work is actually trying to "catch up" with the attackers.

Detection by Source, 2011-2022



As recently as 2011, it was taking a long time to detect attacks, even in organizations with (at the time) cutting-edge security tools. This is reflected in the "dwell time" metric, showing the number of days an attacker was present before detection. This has significantly reduced over the decade:

Global Median Dwell Time, 2011-2022

	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022
All	416	243	229	205	146	99	101	78	56	24	21	16
External	—	—	—	—	320	107	186	184	141	73	28	19
Internal	—	—	—	—	56	80	57.5	50.5	30	12	18	13

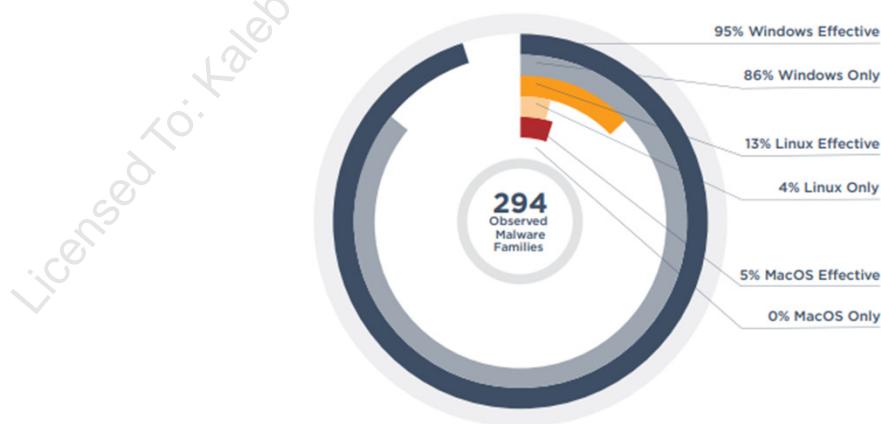
(Source M-Trends 2023 Report)

The good news is that most of this reduction is down to improved security controls, better monitoring, threat hunting, and more effective incident response. Where organizations implement improvements, there are significant improvements.

The bad news is that some of this reduction is simply down to how prevalent ransomware is. M-Trends reported that it accounted for 25% of all incidents it investigated in 2020.² Ransomware is a noisy attack and hard to miss, even in organizations with almost no detective controls. This has resulted in a mixed dwell time being identified—in cases where the attack is driven by ransomware, the dwell time is around five days. In other incidents, it's still over a month with a median dwell time of 45 days.

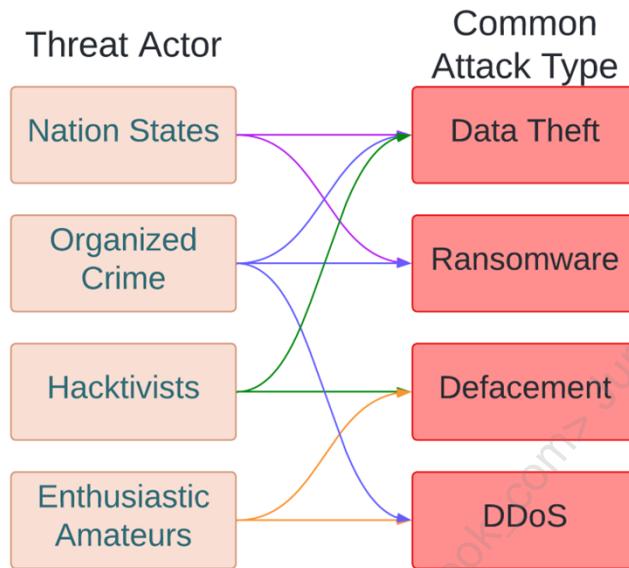
Another significant change is the increase in malware that runs on Linux platforms, with M-Trends (2021) identifying around 13% of observed malware families effective on Linux, with around 4% specifically targeting Linux. This is definitely less than Windows, but it is growing.

EFFECTIVENESS OF OBSERVED MALWARE FAMILIES BY OPERATING SYSTEM, 2020



1. Mandiant M-Trends annual report 2023: <https://for577.com/m-trends-2023>
2. Mandiant M-Trends annual report 2021: <https://for577.com/m-trends-2021>

Attacks and Attackers



Who is driving these numbers?

Today, the range of threats targeting an enterprise can feel limitless. The news is often filled with stories about complex attacks and sophisticated exploits.

If you are new to this, the scope and scale of attacks can feel overwhelming. However, this course is designed to help you become an integral part of the solution. We will look at the tools and techniques you can use to improve your investigations on Linux platforms and the concepts you can use to make your overall incident response more effective.

To try and keep things simple and make our discussions easier, we will group attackers into one of four types.

Nation-States: This is the traditional advanced persistent threat (APT) we hear so much about. Although dating back to the mid-2000s, the term APT became most well-known from Mandiant's M-Trends and its in-depth APT1 report. The term normally means a well-resourced, determined attacker who is not easily dissuaded from their mission.

Organized Crime: Cybersecurity attacks can be very, very profitable. As a result, criminal groups have become heavily involved in this area. Crime gangs have a range of capabilities; some design code for others to use, some simply get access to a network and sell, and others run a full end-to-end attack. Most criminal attacks today involve ransomware at some stage, although this is normally after exfiltrating sensitive data, which is then released if the ransom isn't paid.

Hacktivists: The next category is more varied. The term *hacktivists* covers any individuals or groups who believe they have a moral imperative to carry out their activities. The range of motivations is limitless and unpredictable, but generally, these groups target based on political/government, animal rights, or environmental issues. Hacktivist capabilities vary wildly, but the majority of attacks tend to be defacements.

Enthusiastic Amateurs: The final category is more of a catch-all. This group is hardest to define fully but often includes people trying things they've seen in the news or simply doing things out of curiosity. Attacks are often unsophisticated ("script kiddie") and frequently result in denial of service rather than effective compromise.

On the previous page, we have also included four common attack types. Again, this is a simplification, but it highlights the common objectives we see from different attack groups.

Data Theft: This is common for nation-states and organized crime groups. It is also seen (but less common) with hacktivists, most notably the data leaks regarding multiple off-shore financial houses in the mid-2010s. Any sensitive data or intellectual property can be targeted, and attackers will often collect data with no immediate value.

Ransomware: Possibly the most common form of attack in 2020–2021. This is an attack where the victim's data is encrypted, and they need to pay to get the decryption key. This is most often used by criminal gangs, although some nation-state attackers have been seen deploying ransomware. Nation-state ransomware use is often more destructive, without a mechanism to decrypt or deploy as an anti-forensics measure. Most ransomware attacks also include a data theft element.

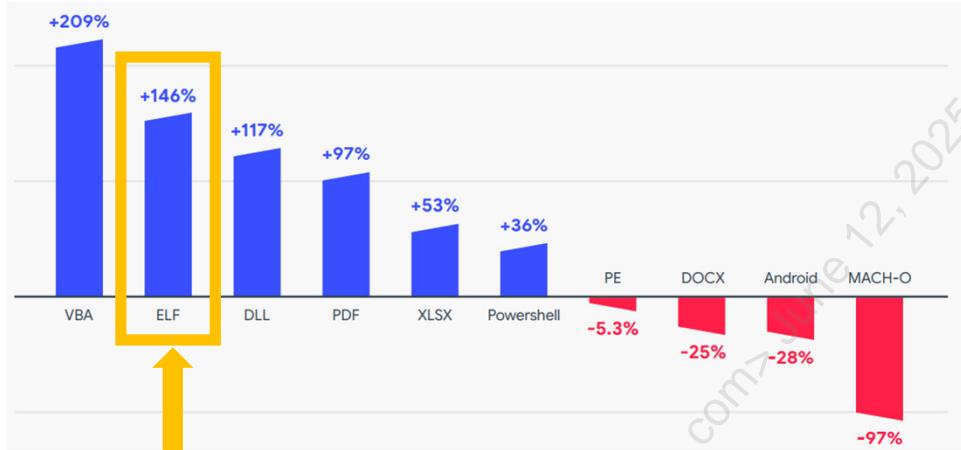
Defacement: Often used by hacktivists or propaganda attacks, defacements are when the attacker modifies the victim's websites to display a message. This is often a boast or claim about whatever the hacktivist group is upset about.

Distributed Denial of Service (DDoS): Very popular with attackers in the early 2000s to 2010s, this is still popular with some criminal groups, where it is used as part of an extortion threat, and very common with enthusiastic amateur attackers. There are reports of nation-state groups using this to support political or military activity. Generally, DDoS attacks have become less effective as more businesses move behind content delivery networks (CDNs) or other filtering/availability services. A related attack (denial of service) often happens when "script kiddies" use badly designed proof-of-concept code and crash systems.

Important Note: The categories above and on the previous page are only to make discussion easier. Frequently, adversaries and adversary groups defy effective categorization. In incident response, it is important that we deal with the incident we have rather than our expectations of a specific attacker.

Malware Trends

Malware distribution trends, 2021 (versus 2020)



Source: VirusTotal 2021 Trends Report

Data provided by VirusTotal¹ shows some interesting changes in malware trends between 2020 and 2021.

Although the largest increase (in VBA malware) shows that MS Windows platforms remain heavily targeted, the increase in ELF-based malware is significant. This shows that adversaries are recognizing the need to specifically target the Linux platform and designing malware appropriately.

At the same time, attacks on the Windows platform are moving away from traditional "malware" (PE files with malicious content) and moving back to the type of attacks that were very common in the early 2000s. These are generally negative documents with macro payloads.

A lot of this is likely driven by the prevalence of EDR tools and other security products in Windows environments. This makes it challenging for attackers to drop malware and have confidence it will run, and in recent years, it has led to an increase in "living off the land" attacks. However, most current Linux environments have less security tooling deployed, with admins feeling it is unnecessary.

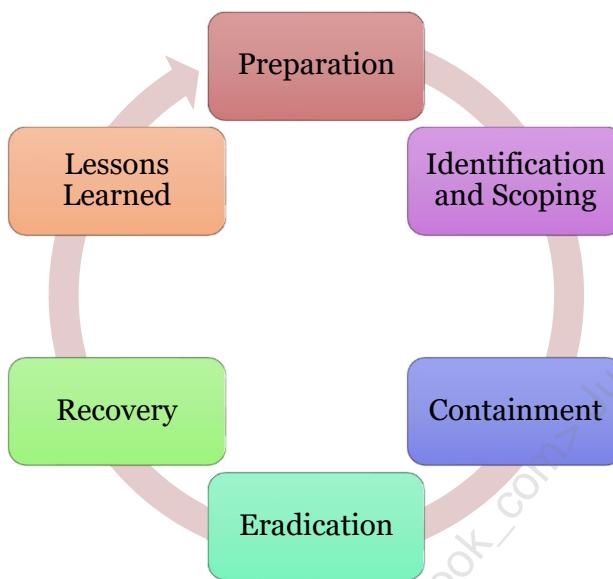
1. VirusTotal report <https://for577.com/vt-2021>



The Incident Response Process

This page intentionally left blank.

The SANS Six-Step Incident Response Process



Computer intrusion incident response can be broken down into a six-step incident process. This model originated from United States government agencies and has been extensively documented by the US National Institute for Standards and Technology (NIST).¹ It has a long and proven history of success and is a good starting place to evaluate the steps required to respond and recover from an incident.

Overview of the Six-Step Incident Response Process

The six steps of this model are preparation, identification, containment, eradication, recovery, and lessons learned.

Preparation

Incident response methodologies emphasize preparation—not only establishing a response capability. Hence, the organization is ready to respond to incidents but also prevent incidents by ensuring that systems, networks, and applications are sufficiently secure.

It is essential that incident responders take good notes through the incident response (IR) process, and guidance (or tooling) for this should be established in the preparation phase.

Identification

Identification is triggered by a suspicious event. This could be an alert from a security appliance, a call to the help desk, or the result of something discovered via threat hunting. Event validation should occur, and a decision should be made as to the severity of the finding. Once an incident response has begun, this phase is used to better understand the findings and begin scoping the network for additional compromise.

Containment and Intelligence Development

In this phase, the goal is to rapidly understand the adversary and begin crafting a containment strategy. Responders must identify the initial vulnerability or exploit, how the attackers are maintaining persistence and laterally moving in the network, and how command and control are being accomplished. In conjunction with the previous scoping phase, responders will work to have a complete picture of the attack and often implement changes to the environment to increase host and network visibility. Threat intelligence is one of the key products of the IR team during this phase.

Eradication and Remediation

Arguably, the most important phase of the process, eradication, aims to remove the threat and restore business operations to a normal state. However, successful eradication cannot occur until the full scope of the intrusion is understood. A rush to this phase usually results in failure. Remediation plans are developed, and recommendations are implemented in a planned and controlled manner. Example changes to the environment include:

- Blocking malicious IP addresses
- Blackholing malicious domain names
- Rebuilding compromised systems
- Coordinating with cloud and service providers
- Enterprise-wide password changes
- Implementation validation

Recovery

Recovery leads the enterprise back to day-to-day business. The organization will have learned a lot during the incident investigation and will invariably have many changes to implement to make the enterprise more defensible. Recovery plans are typically divided into near-, mid-, and long-term goals, and near-term changes should start immediately. The goal during this phase is to improve the overall security of the network and to detect and prevent immediate reinfection. Some recovery changes could include:

- Improving an enterprise authentication model
- Enhanced network visibility
- Establishing a comprehensive patch management program
- Enforcing a change management program
- Centralized logging (SIM/SIEM)
- Enhancing the password portal
- Establishing a security awareness training program
- Network redesign

Lessons Learned

In this phase, the incident response team can learn from what worked and what didn't work. The goal is to make future incidents run better and improve organizational efficiency. Additionally, any threat intelligence created during the response can be consumed (rules created in the SIEM, hashes blocked, etc.). During the lessons learned phase, it is important to avoid turning it into an organizational fight between departments. The goal should be an objective assessment of what processes and tools worked well and what failed. It can also be used to identify any training or deployment gaps.

This phase relies entirely on the notes taken by the IR team. It is essential that good notes are kept by all involved.

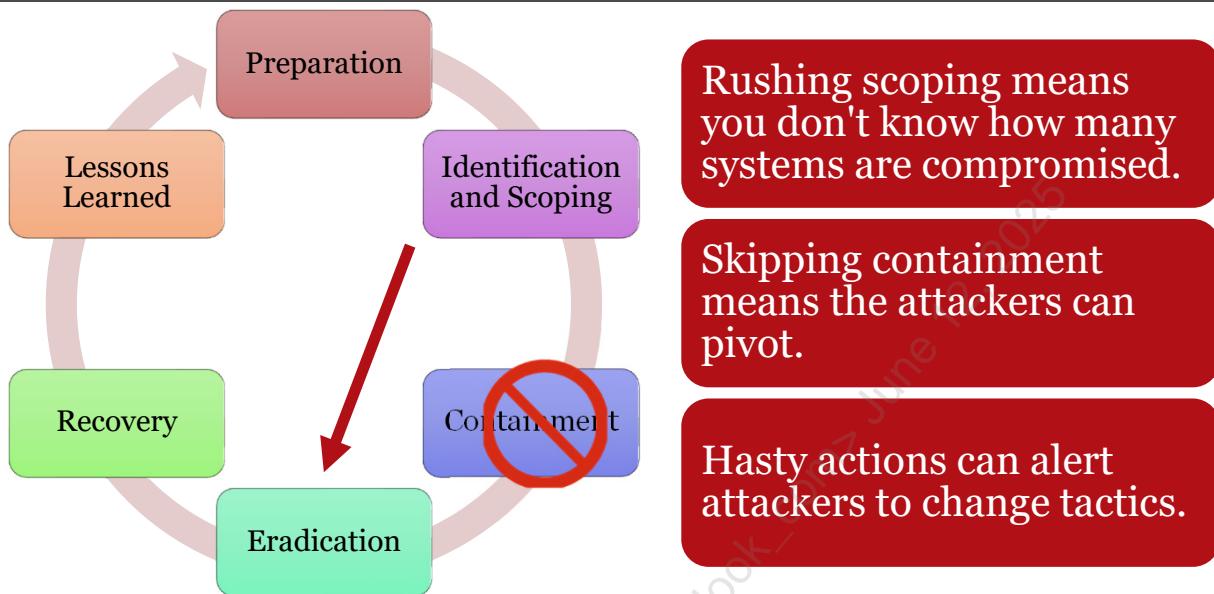
Optional Step

Follow-Up

Follow-up is used to verify that the incident has been mitigated, the adversary has been removed, and additional countermeasures have been implemented correctly. This step combines additional monitoring, network sweeps looking for new breaches, and auditing the network (penetration tests and compliance) to ensure new security mechanisms are in place and functioning normally.

1. NIST Computer Security Incident Handling Guide <https://for577.com/nist-sp-800>

Common Mistakes? Skipping the Investigation



Whatever process people choose, PICERL, NIST, and so on, a common problem is that, in practice, few organizations allow their IR teams to follow the process correctly. Often, this is the result of immense pressure from senior executives, which leads to responders skipping immediately to what they believe is eradication. While it does mean you can get to recovery quickly, how sure can you be that the attack is actually eradicated? Incomplete identification, scoping, and containment means you don't really know how the attack started, how many systems were compromised, how the attackers got persistence, and so forth. Rushing to eradication can leave responders with a false sense of what has happened and eliminate the advantages of an intelligence-led response workflow.

The understandable panic that results from detecting malicious activity can lead to "pulling the plug," shutting down systems, blocking IP addresses, and resetting accounts—all as quickly as possible. This will stop any further attack from the specific vector; it almost never leads to full eradication. Remember, it is unlikely that you've detected the initial foothold. Also, the median dwell time is still around 45 days. It is much more likely that the first alert is related to secondary attacker activity, such as lateral movement to a sensitive system or accessing a privileged account. Rushing to eradication before understanding the attack in detail simply lets the attackers know that response is underway and gives them an opportunity to switch to other systems or alternative footholds. This leads to a game of whack-a-mole where the incident responders are blindly chasing after the attackers, with little actual progress.

This type of response leaves the initiative with the attacker. They are then able to redeploy assets and will quickly learn what is and isn't detectable by the defenders. Unfortunately, for the defenders, this gives a significant advantage to the attacker and can often lead to higher-impact breaches. Removing a small portion of the attacker-owned infrastructure is unlikely to slow them down, even if it makes the incident response team feel like they've succeeded.

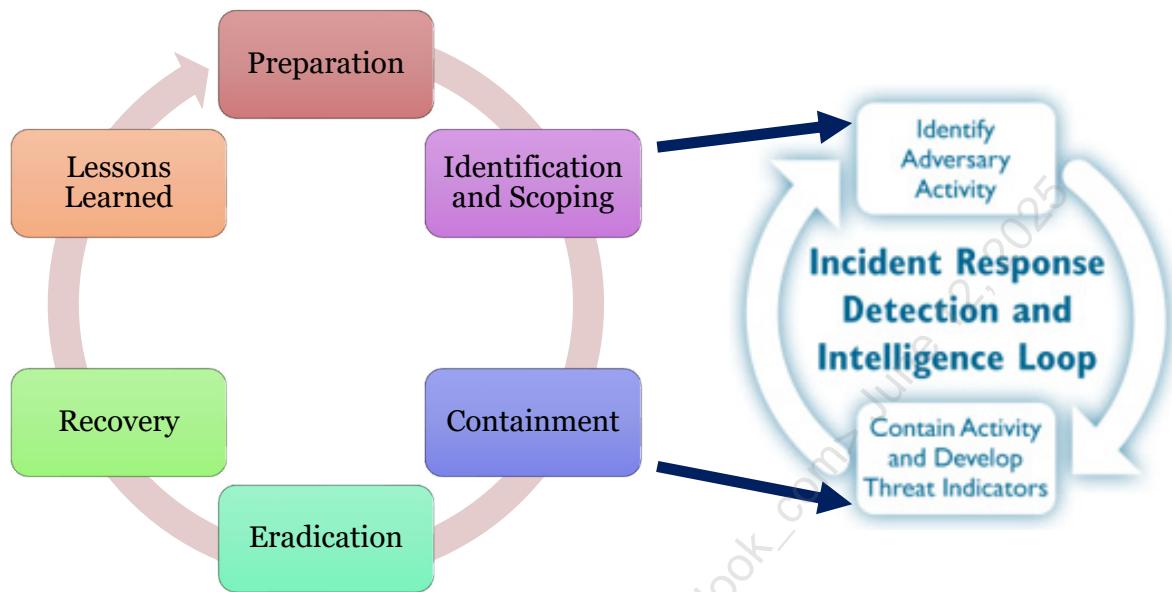
Given how much of a problem this is, what causes the rush to eradication/remediation?

Often, it is a direct result of the organization's fear that they will lose data stored on the system. The data may be viewed as so valuable that the risk of exposing it for one more second is too great to tolerate. As a result, many IR teams know it is a bad idea to remediate too early but are compelled to do so by management's fear of the horse leaving the barn.

But keep in mind, if the attackers have been on the network for days or weeks already (remember, 45-day median dwell time), saving hours or even days might be a mistake. Continuing the analogy, closing the barn door after the horses have been let out is pointless. The rush can also lead the attacker to react to your remediation actions before you are prepared to counteract. They might assume your actions are the beginning of a full-scale remediation and begin a major exfiltration or destructive process on other systems they control. This act/react/counteract model is the norm for intrusion response. As a result, the better you can scope your incident and learn about your adversary, the more eventual control you will have over the results.

Bottom line: Do not react too quickly to an incident by pulling the power. Teams need to move toward intelligence-driven incident response.

The Detection and Intelligence Loop



The importance of intelligence-driven incident response means that during incident response, two of the most important phases are *identification and scoping* and *containment*. Together, these form a mini-cycle called the **Detection and Intelligence Loop**. During an incident, the most valuable activities will be in this mini-cycle.

Here, the responders will analyze the systems they've identified to find indicators. The indicators are used to scope the incident and identify other potentially compromised devices. As new devices are found, they are also analyzed to find more indicators, and the cycle continues. This cycle continues until the IR team believes they have fully scoped the incident.

How do you know when to stop?

This is a challenging question to answer, and every incident will be different. A good guideline is that when your scoping/hunts no longer identify new systems, you have likely finished the detection and intelligence loop.

Intelligence Development and Active Defense



Containment/Active Defense

- Prevent or slow additional access during monitoring and collection phases
 - Full-scale host/network monitoring
 - Data decoy
 - Bit mangling
 - Traffic shaping
 - Adversary network segmentation

Avoid playing your hand!

Intelligence Development

- Tools, techniques, and procedures (TTPs)
- Understanding adversary intent
- Malware gathering
- IOC development
- Campaign identification

SANS

DFIR

FOR577 | Linux Incident Response and Threat Hunting

19

A lot of the time spent in the detection and intelligence loop is spent in the containment phase. This is a combination of developing knowledge about the attack (intelligence development) and taking measures to seize the initiative or restrict the attacker's freedom to operate (containment).

There are two key parts to this activity.

Containment

There is often a misunderstanding about containment. This doesn't always mean locking the attackers out of a system (that is eradication); rather, we should view it as taking measures to degrade the attacker's capabilities, monitor their activities, and deny their objectives. Containment shouldn't be seen as an end state, but it can provide enough breathing room for the incident responders to analyze the attack well enough to build an effective eradication and recovery plan.

One way to think of containment is to control the environment well enough that the attacker's movements are monitored with increasing visibility into their behaviors.

Any techniques you deploy to restrict, limit, or otherwise degrade the attack's capabilities would be considered part of containment. Additionally, being able to deploy enhanced monitoring on suspect devices or intercepting attacker traffic would be part of the containment strategy.

Intelligence Development

During containment, it is essential that the incident response team work to develop intelligence about the adversary. This can't be overstated. If you fail to collect threat intelligence during incidents, then you are effectively starting from the beginning with every investigation. You will also find it nearly impossible to deploy an effective eradication strategy; bluntly, any success is as likely to be down to chance as to good incident response.

Collecting intelligence supports the scoping. For example, you analyze some malware and find it uses an environment variable. You can then use this intelligence to sweep your enterprise to discover if any other systems have the environment variable—and if so, you can expand the scope of the attack. This is referred to as an "indicator of compromise" (IOC), although some organizations prefer to use the term *indicator of attack* (IOA). During this course, we will use both terms interchangeably.

Developing IOCs is really important for incident response teams. They become force multipliers in that they allow a small number of responders to scan a large environment quickly. It is often the only way incident response teams can hope to deal with an advanced adversary.

Although this is not a cyber threat intelligence (CTI) course, CTI is vital and will be covered in more detail shortly. For now, keep in mind that IOCs/IOAs do not need to be malware in the sense of an evil executable file. Attackers take advantage of legitimate system tools and commands and often strive to look like normal users. This makes it vital to consider behavioral indicators, such as the choice of commands, as well as more traditional ones, like IP addresses and file hashes.

During IR, you will eventually reach the point where your intelligence is no longer providing new devices, and you have enough of an understanding of the attacker's intent to assess future actions. This is a good time to move out of containment and begin eradication.

Why Do We Go Slow? Remediation Is Hard!



Threat actors are good at persistence and surviving our response.



Threat actors learn, react, and adapt to our countermeasures.



Persistent attackers *will* return.

The main IR goal for most organizations is getting the attacker out of the network and returning to normal business operations. The issue is that this is harder than lots of people realize, and the business pressure to return to normal creates a rush to get to this step, even if it means the other stages are incomplete.

Unfortunately, without proper scoping and intelligence collection, successful remediation is not possible. All that we end up doing is alerting the attacker to our response and causing them to change and become undetectable. This creates a resource-draining cycle of near-constant incidents.

A better approach is to work through each stage fully and take time to remediate. This maximizes the potential to effectively close gaps, improve monitoring, and harden the environment. It is best to view remediation as a relay race, with each team/step doing its bit and then handing it off to the next, rather than a simple sprint to return to normal operations.

One final point: Remember, the attacker will return. Persistent attackers who want something out of your network will keep going. Any weakness in your remediation efforts will be exploited.

The Future of DFIR: Will AI/LLM Change Everything?



The future is always uncertain.



AI is only a tool; what matters is how it is used.



Advances in technology are always changing DFIR.



It doesn't have a significant impact today, but you should prepare for tomorrow.

We can't discuss digital forensics and incident response processes without considering "artificial intelligence," also referred to using a range of names such as AGI, LLM, and so on. We will use the term AI here to cover both technologies.

First, an important caveat: the future is always uncertain, and technological change is often unexpected. However, today, the impact from AI is limited with regard to both the intrusions we respond to and the way we respond. This leads to the next point: AI is a tool used by attackers and defenders, so we should avoid any impulsive responses, such as banning access or, at the other extreme, throwing all our data into an AI system.

There are genuine concerns that our adversaries can use AI to speed up the discovery of vulnerabilities as well as the entire intrusion process. But even in these situations, our incident response will likely remain the same: we gather evidence, we analyze it, we determine what the adversary has done, and then we remediate and hopefully prevent it from happening again.

From a defensive perspective, AI offers some advantages—it can help us speed up how we process data, it can speed up our understanding of attack paths, and it can speed up how we interpret an intrusion. However, with today's AI models, none of this fundamentally changes the way we perform digital forensics or IR. We still need skilled incident responders to look at evidence and form an idea of how an attack took place.

AI for DFIR—Some Considerations



Validation

Do you trust the answers?



Data Privacy

Where is the data being processed?



Probably "Bad" Ideas

Uploading private evidence to a public AI system



Possibly "Good" Ideas

Code snippets
Scripting help



When it comes to AI use in DFIR, you have some things to consider as part of your planning process:

- How will you validate the output? Most AI tools today have a very high inaccuracy rate when asked for factual output. This is occasionally described as "hallucinations," but whatever terminology you use, it does mean you need to put extra effort into validating any output.
- Where is the data being processed? You need to be very careful about where you are sharing internal data, especially if it is legally privileged or sensitive. This might be less of an issue if you have an internally developed AI platform, but you still need to validate this with your legal teams.
- In general terms, there are bad ideas for using AI—for example, uploading all your internal case data to a public ChatGPT instance and asking it to analyze the material is not something we would suggest.
- Alternatively, where AI can be useful is for building search queries, guiding junior responders in writing useable (if not perfect) code, and so on. AI can also be useful for threat intelligence, such as using it to generate an alternative hypothesis to test your evidence against.

Linux Incident Response and Threat Hunting



SRL Skunkworks



FOR577 | Linux Incident Response and Threat Hunting 24

This page intentionally left blank.

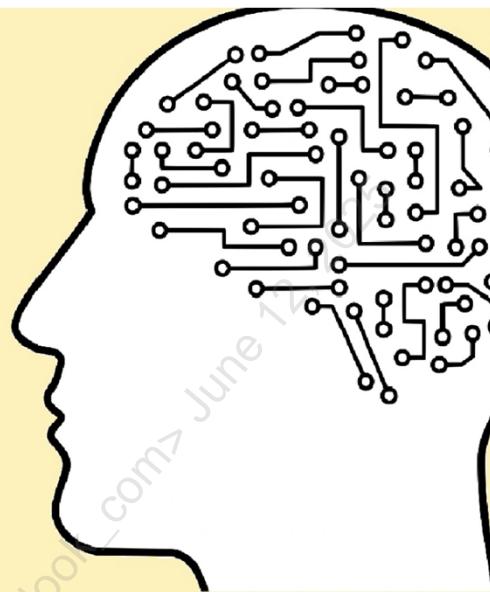
SRL Skunkworks Overview

 SRL Skunkworks [Who we are](#) [What we do](#) [Numbers](#) [Work with us](#) [Testimonials](#)

Cutting edge research making humanity better and safer!

We push the boundaries of Artificial Intelligence, Machine Learning and human-bio-engineering to create a future where we can hold our place amongst the stars.

[Get in touch](#)



SANS | **DFIR** FOR577 | Linux Incident Response and Threat Hunting 25

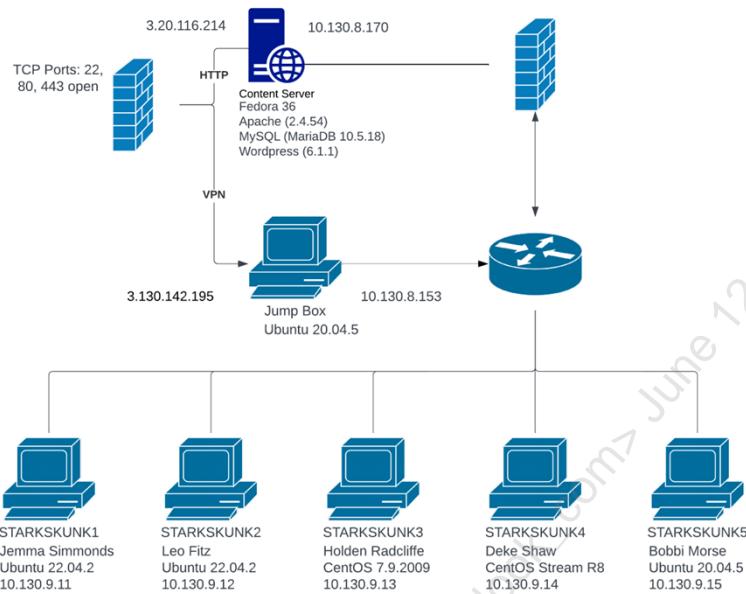
"Stark Research Labs (SRL) specializes in metals, alloys, and biotech research and development for the application of defensive and offensive technologies. SRL specializes in development for North American and European counter-terrorism sectors and for the Department of Advanced Projects (DoAP) under contract from the American Advanced Research Projects – Omni Labs Division (AARP-OLD) and Harbinger Corporation.

SRL is a world leader in the fight to protect our children against head injuries. Our new extremely light bicycle helmets with patented lightweight metal alloys have saved countless lives in bicycle accidents worldwide, but we don't only specialize in metal research. SRL aims to be at the cutting edge of all technology sectors.

Formed in 2018, the Skunkworks is a highly specialized research unit focused on developing Artificial Intelligence products to support defense initiatives. Initially part of the Stark Enterprises environment, following a significant breach of the alloys research network in 2018, Stark Enterprises decided that it was best to isolate the Skunkworks to minimize the risk that a breach of one entity would be able to spread. In 2021, the research team was fully separated and stood up its own environment."

Unlike the main Stark Enterprise environments, the Skunkworks is an entirely Linux network and is managed internally. There is occasional supervision and support from SRL, and recently, the Stark Helpdesk team, Spader Tech, has been providing some support, but there is no regular access for any accounts outside the Skunkworks team. Day-to-day admin and configuration are carried out by the researchers themselves, all of whom are experienced Linux users.

Skunkworks Network Diagram



Skunkworks is a reasonably simple environment. This is good for incident response, as it means that documentation actually exists around how it should look. The downside of this is that the only segmentation is really the DMZ, which hosts the SSH jumpbox and the researchers' content server.

Within the research team, there are five users, all of whom use a Linux distro for their daily work. The users are:

- Jemma Simmonds. Skunkworks director and lead researcher, uses Ubuntu Linux. Username: jsimmonds
- Leo Fitz. Skunkworks technical lead for creating human-machine interfaces, uses Ubuntu Linux. Username: lfitz
- Holden Radcliffe. Researcher, uses CentOS Linux. Username: hradcliffe
- Deke Shaw. Researcher, uses CentOS Linux. Username: dshaw
- Bobbi Morse. Researcher and office manager, uses Ubuntu Linux. Username: bmorse

A PDF copy of this network diagram is provided in the virtual machine provided with this course. It can be located on the Resources tab on the electronic workbook.

Skunkworks Security Information

- Isolated network within Stark Enterprises
- Remote access uses a "jumpbox" in DMZ
- Environment set up by experienced Linux users
- No EDR deployed
- CMS hosted in the DMZ used by researchers
- Workstations use a mix of Ubuntu and CentOS
- CMS server runs on Fedora 36
- Patches/updates applied by users
- Auditd installed on each endpoint
- SELinux installed on Fedora/CentOS devices
- AppArmor installed on Ubuntu devices
- Individual accounts for users to log in
- Sudo requires passwords
- Key-based authentication used for SSH



The Skunkworks is representative of a Linux-centric enterprise created by people who are comfortable with using Linux and administering their own environment.

- The network is fully isolated from other parts of Stark Enterprises. This was enforced by the SRL security administrators (rsydow and cbarton) following previous security incidents.
- Remote access to the workstations is provisioned by a jumpbox in the demilitarized zone (DMZ). Again, this was enforced by SRL security. Individual users within Skunkworks can either access their workstations directly or, when working remotely, they can tunnel in. SRL security prohibits users from taking the workstations away from Stark Enterprises' property.
- The users are experienced Linux users and have built their environment to fit their own needs.
- EDR has been requested by the SRL administrators, but this has not yet been deployed in the environment.
- The researchers use a content management system (CMS) in the DMZ to share team management information. No sensitive data should be stored here.
- Workstations use a mix of Ubuntu and CentOS as their daily operating systems.
- The server hosting the CMS runs on Fedora 36.
- Users are responsible for patching/updating their own systems. There is no central patch management tool running.
- Auditd has been installed on each endpoint to default settings.
- All devices have SELinux (CentOS/Fedora) or AppArmor installed and set to the default configuration.
- Users log in with individual accounts and use sudo for privilege escalation.
- SSH uses key-based authentication throughout the environment.

There are some issues, however, which may need to be identified by the investigation team.

Something Odd?

○ Clint Barton

17/3/2023 19:54



To Bobbi Morse **Copy** Jemma Simmonds

CB

[Reply](#) [Reply all](#) [Forward](#) [Delete](#) [Add to Safe Senders](#)

[Add to Blocked Senders](#)

Hi Bobbi,

This isn't good. I am concerned that there is some strange activity on your network. Your device is being a bit odd and I am going to do some checks.

I hope this isn't anything significant but I am uncomfortable with the network as it is.

Jemma - I've copied you in as we need to have an urgent call about this.

Clint.



FOR577 | Linux Incident Response and Threat Hunting

28

In early March, the IT team at Stark Enterprises decided that having the Skunkworks as a totally isolated environment wasn't the best idea. While it did give separation in the event of an intrusion, it meant that there was no way to centrally monitor or provide security coverage from the Stark Enterprises/Stark Research Labs (SRL) teams. Because SRL has endured several cyber attacks, leadership decided to bring the Skunkworks back into the main network, at least with regard to monitoring.

This led to a project for Clint Barton, the IT Security administrator at SRL, to set up a centralized log collection point. In addition, Clint started checking into some of the activity in the Skunkworks network.

The project began in mid-March, and very quickly (2023-03-17), Clint discovered some strange network activity from STARKSKUNK5 (owned by Bobbi Morse, the office manager/administrator). The traffic appeared to be SSH attempts from STARKSKUNK5 to other devices in the network, which is unusual for that account. When Clint discovered this, he notified Jemma Simmonds (the lead researcher) and Bobbi Morse and then began a basic investigation.

Something Bad?

○ Clint Barton

To Jemma Simmonds

18/3/2023 14:58



CB

[Reply](#) [Reply all](#) [Forward](#) [Delete](#) [Add to Safe Senders](#)
[Add to Blocked Senders](#) [≡](#)

Hi Jemma

The more I look into this, the more concerning it is becoming. I can see some unusual activity from STARKSKUNK5.

I don't think this is Bobbi, so we aren't treating it as an insider issue, but it is serious enough that I am going to engage some IR support.

Thanks,

Clint



FOR577 | Linux Incident Response and Threat Hunting 29

Following his initial investigation, Clint decided it was unlikely this was deliberate behavior by Bobbi and currently does not believe this is an insider threat issue. However, Clint is concerned that Bobbi's machine might be compromised, and this might allow a threat actor access to her mailbox. As a result, Clint has removed Bobbi from his email when it came to notifying Jemma of his belief.

At this time, because Clint is the sole IT Security person within Stark Enterprises, he has decided to reach out to the external Incident Response team and utilize their services.



Lab 1.1: Stark Skunkworks Introduction

Estimated time: 25 minutes

This page intentionally left blank.



Linux Overview

This page intentionally left blank.

Quick History of Linux

A personal project takes over the world.

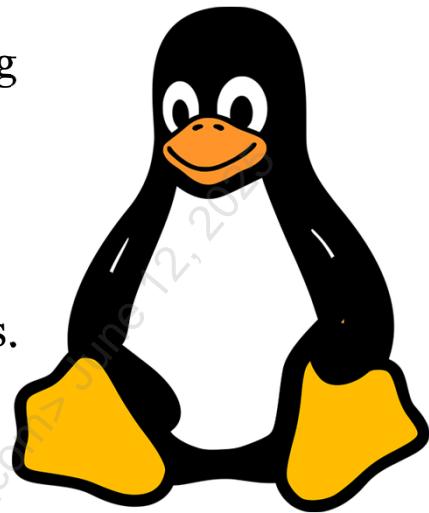


This page intentionally left blank.

What Is Linux?

Quick History Lesson:

- Linux is the kernel—**not** the operating system!
- First announced on August 25, 1991, on Usenet.
- Initially designed as a free alternative to BSD/System V on home computers.
- Today, about 70% of web servers and virtually all supercomputers run Linux.



Linux is really the name for the kernel around which an operating system is built. When we talk about the operating system itself, the "distribution" is really what we should say. However, for ease of discussion during this course, we will use *Linux* to refer to any distribution that uses a Linux kernel and only mention specific distributions when we need to highlight the difference.

The Linux kernel began as a personal project by Linus Torvalds,¹ who was looking to create a free operating system. It started as a hobby project to provide a 32-bit, Unix-like system that would run on computers with Intel 80386 (i386) or 80486 (i486) processors and be affordable to most users. The idea was for it to deliver the capabilities people had on expensive BSD/System V terminals (often mainframes/minicomputers) on hardware they could buy off the shelf or even build themselves.

Linux quickly moved on from being a hobbyist tool, and big production environments like NASA realized it was possible to use the kernel to replace supercomputers with much more affordable components. The best-known example of this was the Beowulf² cluster developed in the mid-1990s. Although it never quite got to the "Linux on the Desktop" dream, the kernel itself arguably powers the modern world. Estimates state that around 70% of public-facing web servers run a version of Linux,³ and by 2005, the majority of supercomputers had moved from traditional Unix to a Linux distribution.⁴ Now, pretty much every enterprise environment has Linux systems somewhere in the estate.

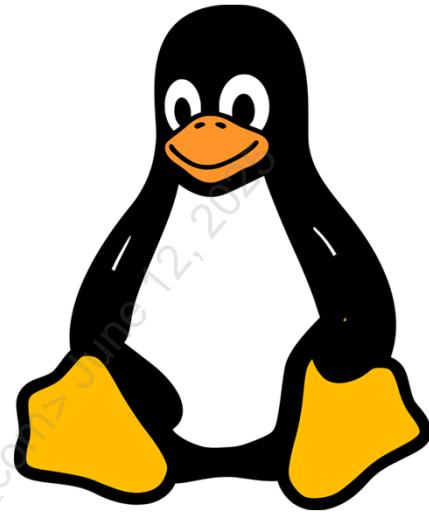
1. <https://for577.com/linuxborn>
2. <https://for577.com/beowulf>
3. <https://for577.com/netcraft>
4. <https://for577.com/supercomputers>

Where Do You Find Linux?

Short answer: anywhere

Linux kernel can be found in:

- Web servers/database servers
- Firewalls
- Routers
- Appliances/IOT
- Cloud systems/containers
- Phones
- Windows 10/11 and later
- And many more places



The Linux kernel powers a range of modern systems. It would be impossible to list every location where you might find a system running the kernel because the flexibility and adaptability the system has means it can literally be configured to run anything. When we look at smart devices in the home, they almost always run Linux.

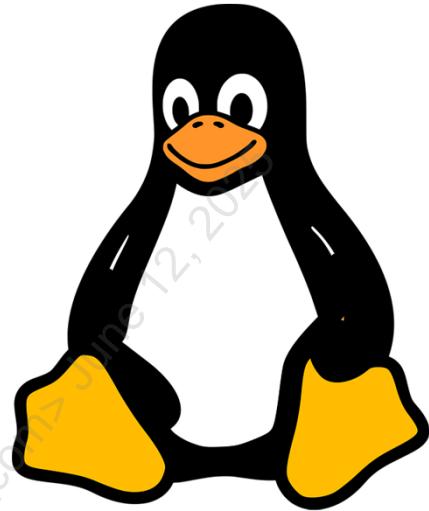
With Windows 10 (and later) and the Windows Subsystem for Linux 2.0 (WSL 2), a fully functional Linux kernel runs within a hypervisor on the Windows host. This creates additional complexities for any host-based investigation and additional attack surface for adversaries.

In the enterprise, even a Windows-dominant one, you will find the Linux kernel powering the infrastructure—firewalls, routers, and so on.

All of this leads to an increasing need for incident responders to understand the Linux environment and be comfortable gathering evidence, analyzing data, and hunting across multiple platforms.

The Distro Matters

- Linux is the kernel.
- The "distro" is the *operating system*.
- Differences matter:
 - GUI differences
 - Package management
 - Log locations
 - *More on this later*



As mentioned before, the name Linux should only refer to the kernel. What matters to most users is the distribution (distro), which is really the operating system. Although we will continue to use *Linux* to mean the generic operating system during this course, it is important to understand the differences.

The list of possible Linux distros is constantly changing, and the user community creates new ones all the time. This has the potential to be an IR nightmare; however, most distros have a tiny user base and never show up in an enterprise environment. Additionally, the complexity of building a fully featured operating system means that almost all distros are based on one of the main branches. The website DistroWatch¹ provides a good place to see what is new, what is trending, and what has changed within the distro community.

In general, distros tend to fall into three categories:

- Based on Debian: Debian, Ubuntu, Mint, Knoppix, Kali, Parrot, Tails
- Based on Red Hat: CentOS, Red Hat Enterprise Linux, Fedora, Oracle Linux
- Everything else. This includes some popular distros like Gentoo, Arch, OpenSUSE, and Slackware but also tends to include the lesser-known platforms rarely seen in an enterprise.

For most users, the choice of distro influences the default graphical user interface (GUI) and the way software is installed. This can lead to confusing combinations. For example, the default GUI for Ubuntu is Gnome, with APT as the package manager. Knoppix uses LXDE with APT as a package manager. Fedora (Red Hat) uses Gnome by default but the RPM package manager. To add to the complexity, these are just defaults. Users can install and configure almost any GUI or package management tool.

When it comes to enterprise IR, it is important that you understand what distros are in use in your enterprise and how they are configured.

In practical terms, RHEL and Ubuntu dominate the commercial world. Approximately 32% of servers run RHEL² or a derivative. Other reporting indicates that about 30% of websites run on Ubuntu.³ Cloud platforms have drastically increased the use of Linux and generally make it trivially easy to spin up a CentOS or Ubuntu image. As a result, it is very rare to see anything else in corporate use.

In the course author's experience throughout 2023, which has involved over 60 incidents on Linux running in a corporate environment, every incident was on RHEL/CentOS or Ubuntu.

While it is important to have a good appreciation of the various distros, most IR work will be on one of the two main distributions.

1. DistroWatch <https://for577.com/dwatch>
2. Red Hat market share <https://for577.com/redhatshare>
3. Server use statistics <https://for577.com/servershare>

Linux DFIR Challenges

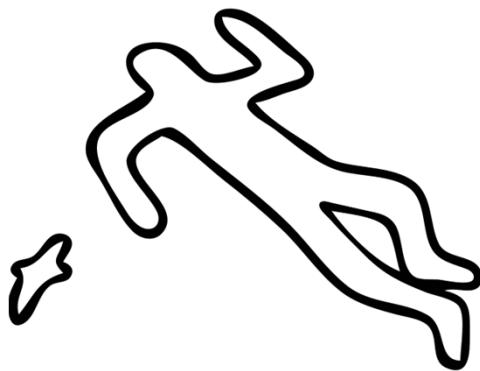


FOR577 | Linux Incident Response and Threat Hunting

37

This page intentionally left blank.

Challenge I: The Victim System



Individual Configurations

Default Settings

Inexperienced Administrators

Sporadic Security Tool Deployment

Linux creates some challenges for defensive security in general, and particularly incident response. Although they are not insurmountable, you need to be aware of them and take them into account during your enterprise incident response.

Individual Configurations

Linux is very configurable, and this is one of the reasons it has become so popular. This flexibility means you can never be sure how a specific machine will be set up. Trivial changes include things like moving log file locations or running a different GUI, but it can be complex and render standard IR approaches impossible.

Default Settings

Related to the previous point, a lot of production Linux servers are set up with out-of-the-box settings. This is very common on cloud platforms where admins simply launch a standard image with minimal hardening. This can lead to issues such as insufficient logging and excessive privileges for user accounts.

Inexperienced Administrators

Often, organizations struggle to hire/retain experienced Linux administrators. This leads to insecure practices, often making the other common issues more likely and increasing their impact.

Sporadic Security Tool Deployment

There is a tendency for people to assume Linux is secure by default, and, as a result, it is often a low priority for security tooling. Even when tooling is available, the performance on Linux is frequently much worse than what's available on Windows. This forces incident responders to have a range of techniques for getting access to the system, gathering data, and validating activity.

You won't encounter these challenges on every system, but you need to be aware that they exist. They are common enough that it takes planning to deal with them, and that's what we will look at for the rest of this module.

Challenge II: The Responder

Inexperienced IR Staff

Often Stand-Alone Credentials

Limited IR Tools

Command-Line Dominance



The victim systems can present issues, but there are also challenges faced by IR teams based on the team.

Inexperienced IR Staff

Although most IR professionals are comfortable using Linux systems to conduct IR where necessary, they often lack experience in how to respond on the platform. This is best addressed with training, developing processes/playbooks, and practicing.

Credentials

Related to system issues, Linux systems are often set up to use stand-alone credentials, even in well-structured enterprises. This creates challenges for responders who need access as privileged users without either exposing the root user credentials or sharing accounts. The best solution, if domain credentials aren't an option, is to set up privileged IR accounts in advance. Depending on your organizational policy, these should be set to key-based or multi-factor authentication, and any use should generate an alert.

Limited IR Tools

A related issue is the lack of security tooling. This can slow down incident responders and force them to move out of their comfort zone. This is especially significant for responders who are used to well-instrumented Windows environments. The main solution is down to practice and rehearsals. One of the biggest advantages of Velociraptor as an EDR tool is its cross-platform functionality, which minimizes this challenge.

Command-Line Dominance

Although GUIs and GUI tools exist for Linux, responders need to be comfortable gathering data from a command line. Even where a good EDR is available, there will still be times when a responder needs to drop into a shell and be comfortable with commands. This is another area where practice pays off, especially for teams that rarely respond on Linux and might not develop the "muscle memory" needed.



Linux Distros

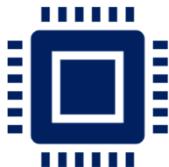
What goes on top of the kernel?

This page intentionally left blank.

Licensed To: Kaleb Sego <kaleb_sego@outlook.com> June 12, 2025

Linux Distributions (Distros)

Distros vary wildly ...



Multi-Platform

- Embedded devices
- End user devices
- Servers
- Supercomputers



Typical Components

- Kernel
- Package management system
- Basic software
- GUI/window environment

The distribution (distro) is really what takes the Linux kernel and turns it into an operating system that end users can use and interact with.

Distros can be found on pretty much any platform, from tiny distros running on embedded devices to fully functioning supercomputers/high-performance computing (HPC) systems. Within this, distros can be commercially maintained, community-maintained, or run as personal projects.

What makes up a distro?

There are no hard and fast rules over what a distro must include beyond the practical requirement of a Linux kernel. In practical terms, however, most functioning distros include the following components:

- This is a package management system or some way to install, update, and remove software.
- Basic software components: This is often a compiler (gcc, etc.) and other system administration tools.
- This is some form of windowing environment (rarely found on supercomputers or embedded distros, though).

In practical terms, the distro maintainer's choices for these components are one of the driving forces behind distros being considered different from others. For example, Alpine Linux uses apk for package management, while Ubuntu uses apt.

When performing incident response on a Linux system, it is important to understand the distro in use as quickly as possible!

What Distro?

Nearly 1k exist; fewer are actively maintained.

Organizations favor:

- Red Hat Enterprise Linux/CentOS
- Ubuntu
- Debian/SUSE (SLES): regionally popular

Cloud platforms:

- Ubuntu
- Amazon Linux
- Alpine Linux in containers

Personal use varies wildly.

A massive challenge we face is the sheer volume of *possible* distros. As of April 2022, there were nearly 1,000 different versions of Linux operating systems registered on distrowatch.com,¹ and the potential for countless others being created by home users as personal projects or by organizations for a specific purpose.

Fortunately, it isn't as complex as this may appear. Only about 270 distros are being actively maintained,² and significantly smaller numbers are used in commercial enterprises.

Most enterprise environments will settle on a single Linux distro, as this makes maintenance and administration easier. Within commercial environments, the overwhelming majority of organizations use Red Hat Enterprise Linux (RHEL)/CentOS or Ubuntu. Debian and SUSE Linux Enterprise Server (SLES) are also seen in enterprises, with Debian being more popular in the Americas and SLES more common in Europe.

Within cloud environments, Ubuntu is the most common. This is closely followed by Amazon Linux in AWS. Amazon Linux itself is a version of RHEL.

The limited resources available in containers mean the environments tend to use a more compact distro—here, Alpine Linux is definitely the most common and often seen as the default for Docker.

1. DistroWatch—search for all distributions: <https://for577.com/alldistros>
2. DistroWatch—search for actively maintained distributions: <https://for577.com/activedistros>

Distros Change



New versions change key features.



Active distros can change frequently.



Enterprise documentation is essential.

Active Linux distros can undergo frequent updates. Ubuntu, for example, produces a new version every six months with an LTS (long-term support) version every two years. RHEL, as a commercial product, issues regular patches, but new versions aren't routinely scheduled. The Linux Kernel itself also has its own update schedules—largely driven by community input.

While most patches result in minor changes, you may find significant differences in the operating system (OS) whenever a version number changes. A good example came when RHEL moved from version 6 to 7 and changed the default filesystem from ext3 to XFS. Fortunately, major differences are rare, but incident responders should make sure they fully understand the environment to which they are responding.

Additionally, patching and updating are often down to individual devices and users. This means that, as an incident responder, you may have to deal with platforms in varying configurations. It is essential that good documentation about configuration and patch status is available, and this should be emphasized during the preparation phase.

What Distros Will We Cover in This Course?

Red Hat Family

- RHEL
- CentOS
- Amazon Linux

Debian Family

- Ubuntu
- Debian



Unfortunately, it isn't realistic to try and cover every distro. Instead, we will focus on the most common distros found in a commercial enterprise, as this is where most incident response will be needed. Fortunately, most distros come from some common families, which means that our understanding of one allows us to understand the others.

The distros we will focus on are:

- **Red Hat Enterprise Linux(RHEL)/CentOS/Amazon Linux:** RHEL is a very popular commercial Linux OS found in many enterprises. CentOS is an open-source version of RHEL provided without commercial support but frequently used by developers and organizations. Amazon Linux is a derivative of RHEL/CentOS, sharing common features.
- **Ubuntu/Debian:** Ubuntu is arguably the most common Linux OS and comes in a free version as well as a commercially supported one. It is a fork of Debian, sharing its common features. This means a lot of the features we will look at will be found on other Debian-based distros.

Although not a focus of this course, we will also look at some features from:

- **SUSE/OpenSUSE:** Most Linux distros are from either RHEL or Debian families, but not SUSE. One of the oldest distros still available, SUSE now comes in an enterprise (commercial version)—SUSE Enterprise Linux Server/SUSE Enterprise Linux Desktop—and a free version called OpenSUSE.
- **Alpine Linux:** This is a very lightweight distro that features very heavily in containers or low-resource environments. It is another independent distro with some unique features around package management and boot daemons.

Linux Terminal Basics

Rocking the command line



FOR577 | Linux Incident Response and Threat Hunting

45

This page intentionally left blank.

Linux Command-Line Basics

Traditional way to interact

Provides flexibility

Supports scripting

```
taz@asgard:~$ fortune | cowsay
/ You work very hard. Don't try to think \
\ as well.
-----  
 \ ^__^
  (oo)\_____
    ||----w |
    ||     ||
```

```
taz@asgard:~/attack$ for i in 10.16.{7..18}.{0..255}; do echo $i >>
target.list; done && for i in $(cat target.list); do nmap -vvvvv -
T4 -p 22,80 $t; done
Starting Nmap 7.80 ( https://nmap.org ) at 2022-07-03 22:23 BST
Initiating Ping Scan at 22:23
Scanning 10.16.7.0 [2 ports]
```

This module will introduce some basic Linux command lines that are extensively used in incident response. Although most modern Linux deployments include a graphical user interface, and this may feel more comfortable to some responders, it is important that we understand the basics of the command line.

Using a terminal (command line) provides some significant advantages to an incident responder, allowing us to be flexible in our activity, and most importantly, it supports scripting. Once you are comfortable with the command line, you can quickly move toward creating shell scripts and automating a significant percentage of your IR workload.

ls: List Directory Contents

Example

- `ls /usr/bin` (List the contents of the /usr/bin directory.)

Useful switches

- `-a` (Show hidden files.)
- `-l` (Use long format and show more details.)
- `-h` (Use human-readable numbers; 1024 = 1k, etc.)
- `--si` (Use human-readable numbers where 1,000 = 1k.)
- `-i` (Show inode numbers.)

The ls command is used to list information about files or directories. If no arguments are supplied, it lists the filenames of all objects in the folder it is run in, sorted alphabetically.

Commonly used switches include:

- `-a` Show hidden files (all files).
- `-l` Use long format. This shows more details like ownership, permissions, and file size.
- `-h` Use human-readable numbers (1024 = 1k).
- `--si` Use SI standard human-readable numbers (1000 = 1k).
- `-i` Show the inode number for a file.
- `-R` This is a recursive search.
- `-A` Do not list implied . (current folder) and .. (parent folder) entries.
- `-t` Sort by time, with the newest entry first.
- `--time-style` Specify how the time will be displayed (e.g. "--time-style=long-iso").
- `--full-time` Display the last modified time in YYYY-MM-DD HH:MM:SS.SSSSSSSSS format.

Note on the timestamps. With the `-l` argument, the ls command displays the last modified timestamp. If this is more than six months old, it will only show the day/month/year. To get access to the full timestamp, you can either use the stat command or one of the following arguments:

- | | |
|--|--|
| <code>--full-time</code> | (as above) |
| <code>--time-style=full-iso</code> | Shows the date in YYYY-MM-DD HH:MM format |
| <code>--time-style=-"%b %e %Y %R"</code> | Shows the date in MMM DD YYYY HH:MM format |

The "Long" Format of ls

```
(taz㉿dagger)-[~/Downloads]
$ ls -al
total 44
drwxr-xr-x  3 taz taz 4096 Dec 14 20:35 .
drwxr-xr-x 24 taz taz 4096 Dec 14 20:37 ..
-rw-r--r--  1 taz taz  735 Dec 14 20:30 coolgames
-rw-r--r--  1 taz taz   55 Dec 14 20:31 .dmrc
-rwxrwxrwx  1 taz taz  14 Dec 14 20:33 evil_chmod → /usr/bin/chmod
-rwxr-xr-x  1 taz taz  350 Dec 14 20:30 evil.sh
-rw-r--r--  1 taz taz 8744 Aug 26 2021 fastFlux.pcap
lrwxrwxrwx  1 taz taz   20 Dec 14 20:32 login.py → /opt/ATTPwn/login.py
-rw-r--r--  1 taz taz  872 Dec 14 20:31 modcheck.sh
-rw-r--r--  1 taz taz  297 Dec 14 20:30 s.cmd
drwxr-xr-x  2 taz taz 4096 Dec 14 20:31 tooling
```

File Type

Owner Permissions

Group Permissions

World Permissions

Owner - Group

File Size

Last Modification Time

Filename

This is the data shown when the `-l` switch is used.

Overview

The output is broken down into "columns" of information as follows (from left to right):

1. File type
2. Permissions
3. Link count
4. Owner
5. Group name
6. File size
7. Modification date (and time if less than six months ago)
8. Filename

The length of this output can vary. The filetype and permissions columns are fixed length (taking up ten characters in total). The other fields will depend on the content. In most display formats, the month will be represented as three characters, but this can also be adjusted by the user.

File Type

The first character is always the file type, where the single character is one of the following:

Character	Meaning
-	(Regular file)
b	(Block special file)
c	(Character special file)
d	(Directory)
l	(Symbolic link)
n	(Network file)
p	(FIFO - named pipe)
s	(Socket)

Permissions

The next nine characters display the file permissions. The first three are the file owner's permissions. The second three are the permissions for other members of the group the file is assigned to, and the last three are the permissions for all other users ("world" permissions).

The permissions are listed in three-character blocks with a dash for any permissions that are missing. For example:

- rwx This shows read, write, and execute permissions.
r-x This shows read and execute permissions (no write permission).

Two other fields can appear here:

- s This shows the setuid or setgid bit is set. This means that when the file executes, it runs with its own UID or GID, not the one of the person executing the file. This allows a file to run with elevated permissions without the need for the user to elevate.

This is an example of the setuid bit being set on the passwd binary to allow ordinary users to change passwords.

```
└─(taz㉿dagger)-[~/Downloads]
$ ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 72424 Mar  3 2022 /usr/bin/passwd
```

- t This shows that the "sticky" bit has been set. Folders with the sticky bit set and files inside a folder with the sticky bit set can only be deleted or renamed by the owner or root user.

This is an example of a folder with the sticky bit set:

```
└─(taz㉿dagger)-[~/Documents]
$ ls -l
total 4
drwxr-xr-t 2 taz taz 4096 Dec 14 22:35 secret_stuff
```

Note: *s* can be found in place of *x* for user or group entries, and the sticky bit is only set on folders.

Link Count

This shows the number of hard links pointing to the file. A hard link is effectively a mirror copy of a given file and references the same inode address. This is different from a soft link (or symbolic link), which is similar to the type you find in a Windows device. With a hard link, if the original file is deleted, the hard-linked file remains and is fully functional.

Owner

This is the user account that owns the file.

Group

This is the group account that owns the file. This defaults to the group the owner is in but can be modified.

File Size

This is the size of the file on disk. By default, this is shown in bytes. This can be rendered in human-readable format with -h, which uses 1000 bytes = 1k. The --si switch renders it with 1024 bytes = 1k.

Modification Timestamp (Default)

This field shows the last time the contents of the file changed. It can render in two ways. If the file modification time is less than 12 months ago, it will show Month, Day, and Time of modification. If it is greater than 12 months, it will show Month, Day, and Year. If the -c switch is added to -l, this will show the metadata change timestamp instead of the file modification timestamp.

Filename

The final field is the filename itself. If this is a symbolic link file, the filename shows the destination of the link file.

This is shown in the following example from a Kali Linux distro.

```
(taz@dagger):[~]
$ ls -l /usr/bin | head
total 827560
-rwxr-xr-x 1 root root          60224 Oct 23  2021 [
-rwxr-xr-x 1 root root          2376 Sep 25 2020 8process.sh
lrwxrwxrwx 1 root root          31 May 11  2022 1password2john → .. /share/john/1password2john.py
-rwxr-xr-x 1 root root          38 Mar 28  2022 2to3 2.7
-rwxr-xr-x 1 root root         14280 Mar 29  2022 411toppm
-rwxr-xr-x 1 root root          39 Aug 15  2020 7z
lrwxrwxrwx 1 root root          24 May 11  2022 7z2john → .. /share/john/7z2john.pl
-rwxr-xr-x 1 root root          40 Aug 15  2020 7za
-rwxr-xr-x 1 root root          40 Aug 15  2020 7zr
```

cat: Concatenate Files or Display Text

Examples

- `cat /etc/passwd` (Display the contents of /etc/passwd.)
- `cat file1 file2 > file3` (Combine file1 and file2 into file3.)

Use notes

- Originally intended to "concatenate" files; takes the content of multiple files and sends it into a single output
- Often used to display text on the screen
- Works well with readable ASCII but can have unreliable output with binary files

Designed as a command to read files sequentially and concatenate them to standard output, cat has become a common way to quickly read text files.

The standard syntax to read a single text file is:

```
cat {filename}
```

Multiple files can be read and will be rendered in the order (left to right) listed on the command line:

```
cat {filename1} {filename2} {...}
```

Wildcard searches can be used, and matching files will be displayed in the sequence the files are matched, normally alphabetically by filename. For example, this would read all the files in a folder:

```
cat /var/log/apache2/*
```

Output redirection can be used to send the cat command data to files or other applications. In this example, the contents of multiple files are being sent to a new file:

```
cat system.conf application.conf > configs.txt
```

Note: stdin/stdout/stderr and Redirectors

stdin

- Standard input
- Normally data typed at the keyboard

stdout

- Standard output
- Normally displays data to the screen

stderr

- Standard error
- Handles error messages, often displaying to the user's screen even when stdout is redirected

Redirectors

- Redirectors can change where stdin/stdout/stderr send data.
- > is an overwrite redirector.
- >> is an append redirector.
- | is used to send the stdout of one command to the stdin of the next command.

An important concept to understand is where the Linux kernel will send output and where it expects to receive input from. This is dictated by file descriptors,¹ also referred to as stdin, stdout, and stderr.

Standard Input

This is file descriptor 0 (or fd 0). In the terminal (command window), stdin defaults to accepting user-entered data. Typically, this is typing on the keyboard, although with redirectors (below), it is possible to pipe the output from one command to the input of the next.

Standard Output

This is file descriptor 1 (or fd 1). In the terminal, stdout defaults to displaying data on the user's screen as shown here:

```
taz@asgard:~$ fortune
You will become rich and famous unless you don't.
```

With redirectors, stdout can be sent to files or the input of a subsequent command.

Standard Error

This is file descriptor 2 (or fd 2). This is slightly more complex. Stderr handles error messages and diagnostics, and although it is a separate stream from stdout, it often sends messages to the same location. There are exceptions; for example, if stdout is redirected away from the terminal (such as writing to a file), stderr will still send messages to the terminal unless the user dictates otherwise.

This is an example of the find command generating stderr messages to the screen because there are files the user doesn't have permission to access.

```
taz@asgard:~$ find /var/log/* -iname "dpkg.*"
/var/log/dpkg.log
find: '/var/log/gdm3': Permission denied
find: '/var/log/private': Permission denied
find: '/var/log/speech-dispatcher': Permission denied
taz@asgard:~$
```

Often, this can be useful for diagnostics or error correction, but it can also be noisy and lead to confusing data. As a result, it is common to see stderr redirected.

Redirectors

Redirection is common in terminal activities. This can range from sending the output of a command to a text file to looping the output from one command to the input of another. The most common are:

- > - Typically used to send output to a file, this redirector will overwrite the destination if it exists (called "clobbering"). If the destination doesn't exist, it will be created. This redirector is also often used to suppress error messages with syntax similar to 2>/dev/null. In this example, the find command has been rerun, this time with stderr sent to /dev/null.²

```
taz@asgard:~$ find /var/log/* -iname "dpkg.*" 2>/dev/null
/var/log/dpkg.log
taz@asgard:~$
```

The output is much cleaner and now only shows pertinent information.

- >> - Also, typically used to send output to a file, this redirector will append to the destination if it exists. If the destination doesn't exist, it will be created.
- < - Used to redirect stdin from a file. This allows you to send the contents of an existing file into a command as direct input.
- | - Normally used to send output to the stdin of the next command. This is referred to as a "pipe" and forms the basis of chained commands in Linux.

This is an example of chaining commands with a | (pipe) redirector. Here, the output of the ls command is passed to a tr command to change the characters from lowercase to uppercase.

```
taz@asgard:~$ ls -lh
total 40K
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Desktop
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Documents
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Downloads
drwxrwxr-x 4 taz taz 4.0K Jun 29 23:32 git
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Music
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Pictures
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Public
drwx----- 5 taz taz 4.0K Jun 29 23:30 snap
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Templates
drwxr-xr-x 2 taz taz 4.0K Jun 29 21:34 Videos
taz@asgard:~$
```

```
taz@asgard:~$ ls -lh | tr a-z A-Z
TOTAL 40K
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 DESKTOP
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 DOCUMENTS
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 DOWNLOADS
DRWXRWXR-X 4 TAZ TAZ 4.0K JUN 29 23:32 GIT
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 MUSIC
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 PICTURES
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 PUBLIC
DRWX----- 5 TAZ TAZ 4.0K JUN 29 23:30 SNAP
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 TEMPLATES
DRWXR-XR-X 2 TAZ TAZ 4.0K JUN 29 21:34 VIDEOS
taz@asgard:~$
```

Although this is a purely decorative example, it shows the impact of being able to send the stdout from one command to the stdin of the next command.

1. File descriptors are unique identifiers for resources. You can read more about them at <https://for577.com/fdesc>.
2. The null device (/dev/null) is a device file that discards any data sent to it but reports to the operating system that the write operation was successful.

Is There a Right or Wrong Way to Use Cat?

Most commands will work with files directly.

Good practice is to avoid pipes where possible.



Example 1

```
$ cat file | sort
```

Example 2

```
$ sort file
```

Pro tip

Do what works for you.

Linux (and Unix) purists will point out that using cat to read a single file and then piping the output to a second command is inefficient because it forks a second process that consumes additional resources.

For example, an analyst has been tasked with identifying how many times the phrase *magic* appears in the xato-net-10-million-passwords-100000.txt wordlist. There are a couple of ways this can be achieved:

The first is an example of what is called "cat abuse":

```
taz@Scimitar:~$ cat xato-net-10-million-passwords-100000.txt | grep magic | wc -l
26
```

The second is simpler and saves a process.

```
taz@Scimitar:~$ grep magic xato-net-10-million-passwords-100000.txt | wc -l
26
```

As you can see, the outcome is the same.

Even though most Linux systems today have sufficient resources that this is unlikely to be a problem, it is still good practice to avoid any unnecessary pipes. However, there will be situations where you have used cat (or less, or head, or tail, etc.) to examine the file before you know what the next steps will be. This can lead to it being easier and faster to leave cat in place rather than rewriting the command.

For this course, we say you should use whatever approach works best for you.

Formatting Output with column

Default output format

```
sansforensics@siftworkstation: ~
$ cat /etc/passwd | grep bash
root:x:0:0:root:/root/bash
sansforensics:x:1000:1000:sansforensics:/home/sansforensics:/bin/bash
```

```
sansforensics@siftworkstation: ~
$ cat /etc/passwd | grep bash | column -t -s :
root          x  0    0   root        /root
sansforensics x  1000 1000 sansforensics /home/sansforensics /bin/bash
```

Formatted as a table using the colon (:) as a delimiter

When viewing text-based output in Linux, there are several options to make it more readable. One of the more useful ones, which we will look at here, is the `column` command. This takes input and formats it into multiple columns. The number and size of the columns are determined by the input source, but this can be overridden by command-line arguments. The program will analyze the input to try and determine the best layout, with evenly distributed data wherever possible. Formatting data in this manner can make it easier to read some evidence items.

The default option for the `column` command is not that useful—it takes the data and formats it by filling each column before it fills rows. Often, this results in all the output being on a single line. A better approach is to use the `-t` argument, which creates a tabular (table-like) layout, as in the example shown on the slide.

If the input data doesn't have natural separators (generally whitespace such as tab or newline characters), then you can pass a separator character with the `-s` argument.

Common arguments include:

- `-t` Table format output.
- `-s` Specify an input separator (the default is whitespace).
- `-o` Specify an output separator (the default is two spaces).
- `-J` JSON output (requires column names to be specified).
- `-N` Specify the column names with a comma-separated list. This will be used for the table header.
- `-n` Provide a name for the table (the default is "table").
- `-h` Show the help file.

Some examples of how this can be used to format output include:

- Showing users with a bash shell in an easier-to-read format:

```
cat /etc/passwd | grep bash | column -t -s :  
root          x  0    0    root        /root      /bin/bash  
sansforensics x 1000 1000 sansforensics /home/sansforensics /bin/bash
```

- Showing users with a bash shell in CSV format output:

```
cat /etc/passwd | grep bash | column -t -s : -o ,  
root      ,x,0  ,0  ,root      ,/root      ,/bin/bash  
sansforensics,x,1000,1000,sansforensics,/home/sansforensics,/bin/bash
```

- Showing users with a bash shell in JSON format output, providing header names for each column and a table name:

```
cat /etc/passwd | grep bash | column -t -s : -J -N  
"username, pwd, uid, gid, user info field, home folder, login shell" -n "bash  
shell users"  
{  
  "bash shell users": [  
    {  
      "username": "root",  
      "pwd": "x",  
      "uid": "0",  
      "gid": "0",  
      "user info field": "root",  
      "home folder": "/root",  
      "login shell": "/bin/bash"  
    }, {  
      "username": "sansforensics",  
      "pwd": "x",  
      "uid": "1000",  
      "gid": "1000",  
      "user info field": "sansforensics",  
      "home folder": "/home/sansforensics",  
      "login shell": "/bin/bash"  
    }  
  ]  
}
```

grep: Search File Contents for a Pattern Match

Example

- `grep nmap access_log` (Search a file named *access_log* for any strings that match the term *nmap*.)

Useful Switches

- `-r` (Search recursively.)
- `-i` (Case-insensitive search.)
- `-o` (Print only the matched [non-empty] parts of a matching line.)
- `-v` (Exclude matches.)
- `-E` (Pattern match to an extended regular expression.)

Use Notes

- Very powerful for carving through data to find keywords

Grep is a command-line tool for searching data sets to find lines where there is a match with a predefined pattern. Grep can be used to search through single files, groups of files, or even entire filesystems.

The general syntax is:

```
grep [switches] PATTERN DATASET
```

The pattern can be a simple string ("foo") through to very complex regular expressions such as "(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)" to find IPv4 addresses.

Additional switches worth considering include:

- | | |
|------------------|---|
| <code>-F</code> | This disables any regular expression matching and only matches on simple strings. While it does reduce the functionality slightly, it can make using grep significantly faster. |
| <code>-a</code> | This forces grep to treat all input as ASCII. |
| <code>-A#</code> | Show # (number) lines after the matching line. |
| <code>-B#</code> | Show # (number) lines before the matching line. |
| <code>-C#</code> | Show # (number) lines before and after the matching line. |
| <code>-E</code> | This is extended regular expression matching. |
| <code>-e</code> | This is regular expression matching. This switch allows multiple patterns to be used, for example:
<code>grep -e "pattern1" -e "pattern2" filename.</code> |

Regular Expressions: A Quick Guide

Basic search syntax

- Use metacharacters to define search patterns.
- For example, `grep '^a.`' matches any line starting with an "a", followed by any other single character and then a space.

Escape special characters

- When you want to pattern-match for absolute values, remember to escape special characters.
- For example, `grep '\\.file'` matches on the period; otherwise, it is treated as a metacharacter.

Extended regular expressions

- You can expand the syntax with `grep -E` or `egrep`.
- An example of this is `egrep 'POST|GET' access.log`. This will match on either phrase.

Using regular expressions (regex) with `grep` is a powerful way to match complex signatures. Regex can take a bit of practice to get used to, so it is definitely worth practicing. You can use regular expressions with a variety of tools, but here we will concentrate on using it with the `grep` pattern match tool.

With `grep`, you can use regex in one of the following ways:

- Basic regular expression matching. This is the simplest way to use it. You run the `grep` command and provide a string to match. If the string contains metacharacters, they will be acted upon.
- Extended regular expression matching. This allows for more complex matching and uses either `grep -E` or the `egrep` command.

Basic Search and Metacharacters

For most searches, you will use a basic search function and let the metacharacters determine what the matches should look like. These are the most common metacharacters:

.	Matches any single character, except a newline.
^	Anchors the match at the start of a line.
\$	Anchors the match at the end of a line.
*	Matches the preceding element zero or more times.
+	Matches the preceding element one or more times.
?	Matches the preceding element zero or one time.
[]	Matches any one of the characters inside the brackets. A range of characters can be specified using a hyphen. An example would be [a-z].
	Acts as a logical OR between patterns.
()	Groups parts of a pattern together.

Escaping Special Characters

Since many characters have special meanings in regex, you might need to search for the literal characters. This is done using the backslash (\) to escape these metacharacters.

Example: To search for a literal period (.), you use \. in your regex: grep 'secret\.file' evidence.log

This will search the file "evidence.log" and return any exact matches for secret.file.

However, if you run it without the backslash (grep 'secret\.pdf' evidence.log), it would also match on strings like secretsfile as well as secret.file.

Extended Regular Expressions

By default, grep uses Basic Regular Expressions (BRE). For more advanced features, you can use Extended Regular Expressions (ERE) with the -E option, or you can use the egrep command.

ERE includes:

- + Matches one or more of the preceding elements.
- ? Matches zero or one of the preceding elements.
- | Alternation. Matches either the pattern before or the pattern after the |.
- (**) Grouping.

Examples: grep -E 'root|user' file.txt or egrep 'root|user' file.txt matches lines containing 'root' or 'user'.

grep -E 'a+' file.txt or egrep 'a+' file.txt will match lines containing a, aa, aaa, etc. (Any line that has at least one "a" character will meet the condition.)

As you can see, this is a very powerful way to perform pattern-matching, but it can also end up matching many things you don't want.

AI Use: Crafting Regex with ChatGPT

- Use natural language to build search strings.
- Make sure you test before trusting the results.

ChatGPT 4.0

I have a webserver access log in the combined log format. Can you provide me with a regular expression that I can use with grep to match all instances of the phrase webshell.cgi with a HTTP response code of 200?

Certainly! To match all instances of the phrase `webshell.cgi` with an HTTP response code of `200` in a web server access log using grep, you can use the following regular expression:

```
sh
grep 'webshell\cgi[^"]*' 200, access_log
```

One use case for AI models such as ChatGPT is to either generate regular expressions or validate your own strings. This can be an incredibly quick way of generating very complex regex.

In general terms, you should structure the prompt to include the following elements:

- **Context:** Explain what kind of threat intelligence you are dealing with. This could be URLs, IP addresses, file hashes, email addresses, log entries, etc.
- **Patterns:** Describe the patterns you expect to see in the data. Include specific examples of the data you want to match, as well as examples of similar data you do not want to match.
- **Use case:** Specify the use case for the regex. Are you filtering logs, searching in emails, analyzing network traffic, etc.?
- **Environment:** Mention the environment or tool where this regex will be used. Different tools (like grep, Python, Splunk, etc.) might have slightly different regex implementations.
- **Complexity:** Indicate if you need a simple or complex regex. "Simple" might mean matching a straightforward pattern, while "complex" could involve multiple conditions or nested patterns.

It is important, however, that you validate the advice provided. Also, remember that different AI/LLM platforms might return different answers.

Here is an example of a prompt that should have good results:

"I need help creating a regex for threat hunting in my network logs. Specifically, I'm looking to match suspicious URLs that may indicate command and control activity.

1. **Context:** The logs contain various types of entries, but I am focusing on IP addresses.
2. **Patterns:** Match IP addresses in the following ranges 54.12.12.100 – 54.12.13.100 but exclude 54.12.12.254 and 54.12.12.255
3. **Use Case:** I am using this regex to filter logs in a SIEM tool.
4. **Environment:** The regex will be used in Splunk.
5. **Complexity:** I need a comprehensive regex that can match the above patterns while minimizing false positives.

Please generate a regex based on these requirements."

When this example was passed to ChatGPT (on 2024-06-03), the response (shown below) suggested the following regular expression:

```
(54\.12\. (12\. (10[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-3]|[^2][^5][^4-5])|13\. (100|[0-9]|1[9][0-9])) )
```

When used in Splunk, ChatGPT further suggested using this format (note slight change of escape characters):

```
54\.12\. (12\. (10[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-3]|[^2][^5][^4-5])|13\. (100|[0-9]|1[9][0-9])) )
```



To create a regex for matching IP addresses in the specified range while excluding specific addresses, you can use the following regex pattern:

```
regex Copy code  
(54\.12\. (12\. (10[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-3]|[^2][^5][^4-5])|13\. (100|[0-9]|1[9][0-9])) )
```

Explanation of the regex:

1. `54\.12\.` : Matches the initial part of the IP addresses.
2. `12\.` : Matches the first octet 12 in the third block.
3. `(10[0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-3]|[^2][^5][^4-5])` : Matches the IP addresses in the range 54.12.12.100 to 54.12.12.253, while excluding 54.12.12.254 and 54.12.12.255.
 - `10[0-9]` : Matches 100 to 109.
 - `1[0-9]{2}` : Matches 110 to 199.
 - `2[0-4][0-9]` : Matches 200 to 249.
 - `25[0-3]` : Matches 250 to 253.
 - `[^2][^5][^4-5]` : Excludes 254 and 255 by ensuring the third octet is not 254 or 255.
4. `|13\. (100|[0-9]|1[9][0-9])` : Matches the IP addresses in the range 54.12.13.0 to 54.12.13.100.

Different AI Models Give Different Answers



You

I have a webserver access log in the combined log format. Can you provide me with a regular expression that I can use with grep to match all instances of the phrase webshell.cgi with a HTTP response code of 200?



Copilot

Certainly! To match all instances of the phrase `webshell.cgi` with a HTTP response code of 200 in your webserver access log using `grep`, you can use the following regular expression:

```
webshell\.cgi.*\s200\s
```

☰ Gemini ▾

+

⋮

I have a webserver access log in the combined log format. Can you provide me with a regular expression that I can use with grep to match all instances of the phrase webshell.cgi with a HTTP response code of 200?



Show drafts



Here is a regular expression that you can use with grep to match all instances of the phrase "webshell.cgi" with a HTTP response code of 200 in a combined log format webserver access log:

```
".*webshell\.cgi.* 200"
```

This is an example of one problem we often face with AI/LLM tools. Your exact output will vary depending on the exact wording of the prompt you provide and, even for identical prompts, the specific tool you use.

Largely this is because each platform has been trained on different data and using different learning patterns. Because of this, it is important that you avoid blindly trusting specific answers from an AI platform and always make sure you validate them before use in production or during a live incident.

On this page we can see that using the same query on Microsoft's Copilot produces a different answer than Google's Gemini, and both are different than the response from ChatGPT. This is true even though it is a reasonably simple regular expression request. All three are likely to match the string as requested, but they will have varying degrees of false positive rates.

Also, be aware that the models do change, and learn, over time. If you run the same prompts today, you might get very different output from the information shown here.

Working with Compressed Files: zcat

Example

- `zcat passwords1.txt.gz | grep gram` (This will read the file and pipe the output to grep for pattern matching.)

Useful Switches

- `-d` (Decompress the file.)
- `-r` (Recurse through directories.)
- `-h` (Display the help message.)

Use Notes

- Can be quicker than decompressing large files
- Only reliably works with zip/gzip formats

The cat command works well with plain text, but sometimes you want to review compressed files without fully decompressing the files first. The Gzip package comes with the zcat command to allow this.

The zcat command allows you to read compressed files, as long as the files were compressed with Zip or Gzip formats. It will often fail with 7z or xz compression.

Overall, zcat is very similar in use to the cat command. The standard syntax to read a single text file is:

```
cat [options] {filename}
```

Commonly used optional switches are:

- d, --decompress Decompress.
- h, --help Give this help.
- l, --list List compressed file contents.
- r, --recursive Operate recursively on directories.

Note: The help page displayed for zcat is the same as for gzip and gunzip, as all three tools are in the same package.

Working with Compressed Files: zgrep

Example

- `zgrep secret sensitivefiles.gz` (Search a compressed file named `sensitivefiles.gz` for any strings that match the term `secret`.)

Useful Switches

- `-i` (This is case insensitive.)
- `-o` (Print only the matched [non-empty] parts of a matching line.)
- `-v` (Exclude matches.)
- `-E` (Pattern match to a regular expression.)

Use Notes

- Very powerful for carving through data to find keywords.

Zgrep is a command-line tool for searching compressed data sets to find lines where there is a match with a predefined pattern. Unlike traditional grep, zgrep does not support recursive searches, although wildcard (*) searches do allow it to match against all files in a given location. Zgrep will pattern match in non-compressed files as well as compressed files.

The general syntax is:

```
zgrep [switches] PATTERN DATASET
```

The pattern can be a simple string ("foo") through to very complex regular expressions such as "(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]| [01]?[0-9][0-9]?)" to find IPv4 addresses.

Most switches are the same as with grep, although -r is not supported.

wc: Count the Number of Bytes, Lines, or Words in a File

Example

- `wc access_log` (Read access_log and report how many lines, words, and characters are in the file.)

Useful Switches

- `-l` (Show the number of lines.)
- `-c` (Show the number of bytes.)
- `-w` (Show the number of words.)
- `-m` (Show the number of characters.)

Use Notes

- This is effective for quickly assessing the volume of data you need to review.
- If no arguments are given, the output is in the following order: lines, words, characters.



Although it stands for "word count," wc is a command-line tool used to generate statistics from a given source. Used without any arguments, it produces newline count, word count, and byte count, although often it is used with a switch to limit the results.

The general syntax is:

`wc [switches] SOURCE`

This is an example of checking the passwd file to see how many lines, words, and individual characters it contains:

```
taz@asgard:~$ wc /etc/passwd
 47   86 2809 /etc/passwd
taz@asgard:~$
```

The input source can be standard input or a specific file. In DFIR, this is often used to quickly assess the volume of output from another command; for example, this would show how many times `secret.html` was called in an Apache weblog.

```
$ grep "secret\.html" /var/log/apache2/access.log | wc -l
173
```

sort: Takes Input and Sends to stdout in Order

Example

- `sort userdata.txt` (Reads userdata.txt and sends it to stdout in alphabetical order.)

Useful Switches

- `-r` (Reverse the sort order.)
- `-n` (Compare strings based on numeric value.)
- `-b` (Ignore leading blanks.)
- `-u` (Suppress all but one in each set where keys are equal.)

Use Notes

- An effective way of ordering data, especially when piped into other commands

The sort command takes input and writes a sorted concatenation to output. The input source can be a single file, multiple files, the output of other commands, or anything piped to stdin. Output can be stdout (displayed on the screen), piped to another application, or written to a file.

The standard syntax is:

```
sort {switches} {filename}
```

The switches allow for changes to the sort order (which defaults to A to Z or lowest to highest numbers), sorting on numbers or even ASCII code values. The `-u` switch is similar to piping sort to the `uniq` command.

uniq: Removes Duplicate Adjacent Lines

Example

- `uniq userdata.txt` (Read userdata.txt and remove any duplicate lines.)

Useful Switches

- `-d` (Only return the duplicate lines.)
- `-c` (Count the number of occurrences.)
- `-i` (Ignore the case.)

Use Notes

- `uniq` only compares adjacent lines. That means files must be sorted first for maximum value.



The `uniq` command compares lines and removes duplicate data. Note: this only works on adjacent lines, so input sources should always be sorted before the comparison.

Standard syntax is:

```
uniq {switches} filename
```

head: Show the First N Lines of a File

Example

- head access_log (Read access_log and display the first ten lines.)

Useful Switches

- -n # (Show # number of lines.)
- -c # (Show # number of bytes.)

Use Notes

- *Head* can be an effective way to check the structure of files before you run other commands.



The head command shows the beginning of a file. This defaults to the first ten lines but can be changed to an arbitrary number of lines or bytes from the start of the file.

Typical syntax is:

```
head {switches} {filename}
```

One of the more common DFIR uses for head is to check the first couple of lines of a file to make sure it is readable and determine the structure before running other commands. For example, running head against a log file can show the best way to use cut to extract specific data elements.

tail: Show the Last N Lines of a File

Example

- tail /var/log/dmesg (Read dmesg log and display the last ten lines.)

Useful Switches

- -n # (Show # number of lines.)
- -c # (Show # number of bytes.)

Use Notes

- Tail is frequently used to read the most recent events in log files, where data is appended to the end of the document.

Acting as the opposite of head, the tail command shows the final lines of a file. This defaults to ten lines but can be adjusted by the user to an arbitrary number of lines or bytes from the end of the file.

Typical syntax is:

```
tail {switches} {filename}
```

Tail is often used to read files where data is appended, so only the most recent entries are being returned to the user. A typical example of this is reading log files, such as the screenshot below looking at the /var/log/boot.log file:

```
root@midgard:/var/log# tail boot.log
      Stopping CUPS Scheduler...
[ OK ] Started GNOME Display Manager.
[ OK ] Stopped CUPS Scheduler.
[ OK ] Stopped CUPS Scheduler.
      Stopping CUPS Scheduler...
[ OK ] Started CUPS Scheduler.
[ OK ] Closed CUPS Scheduler.
      Stopping CUPS Scheduler...
[ OK ] Listening on CUPS Scheduler.
      Starting CUPS Scheduler...
```

file: Check the File's File Signature

Example

- `file ./susp` (Compare the file signature with known file types.)

Useful Switches

- `-z` (Look inside zip files/compressed archives.)

Use Notes

- This can be a quick way of determining file types.
- It is limited to known file signatures.
- Attackers can defeat this by modifying the file signature.

The file command checks the file signature of an object and compares it to a list of file types. This makes extensive use of the file signature to determine file types. An example of this is with Microsoft executable files that start with the ASCII characters MZ (hex string: 4d 5a), as shown in this screenshot:

```
$ xxd sample | head
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000 MZ.....
00000010: b800 0000 0000 0000 4000 0000 0000 0000 .....@.....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....0.....
00000030: 0000 0000 0000 0000 0000 0000 d800 0000 .....0.....
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468 .....!..L.!Th
00000050: 6973 2070 726f 6772 616d 2063 616e 6e6f is program canno
00000060: 7420 6265 2072 756e 2069 6e20 444f 5320 t be run in DOS
00000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000 mode....$.....
00000080: ac25 c5e5 e844 abb6 e844 abb6 e844 abb6 .%.D...D...D..
00000090: 1267 b2b6 e544 abb6 e844 aab6 8344 abb6 .g...D...D...D..
```

Typical syntax is:

```
file {switches} {filename}
```

Common File Signatures

File Type	Hex Signature	ASCII Signature
Linux ELF File	7F 45 4C 46	.ELF
PDF File	25 50 44 46 2D	%PDF-
.deb Package File	21 3C 61 72 63 68 3E 0A	!<arch>.
RPM Package File	ED AB EE DB	í«íÙ
Ext4 Inode	EF 53	íS
XFS Superblock	58 46 53 42	XFSB
PNG Graphic File	89 50 4E 47 0D 0A 1A 0A	.PNG....
Windows Executable	4D 5A	MZ



This page intentionally left blank.

cut: Slice Input to Get the Bits You Want

Example

- `cut -d: -f1,7 /etc/passwd` (Slice the passwd file using a colon [:] as a delimiter and select the first and seventh fields.)

Useful Switches

- `-d` (Use this as a delimiter to cut on.)
- `-f` (Select this field or combination of fields.)

Use Notes

- It provides a good way to minimize data before passing it to other tools for analysis.

The `cut` command is an excellent way to slice specific items out of larger blocks of data. It works by removing a section out of each line of a file. The key concepts are the delimiter to use and the field choice.

Delimiter: This defines what the `cut` command will use to mark the boundary of each slice. Any single character can be used, although if a space (or special characters like \$ and #) is used, it needs to be escaped. Escape options include using a \ character (such as \\$ to avoid the shell interpreting it) or enclosing in single or double quotes.

Field: This tells `cut` which slice to display. It can be a single field (e.g., `-f3`), multiple fields (e.g., `-f3,4,5`), or even a range of fields. The range can be between specific entries (e.g., `-f6-9`) or with just a stop or start field (e.g., `f-7` or `f3-`).

`Cut` works well as a precursor to minimize data for other tools. For example, this is looking at the `/etc/passwd` file and cutting on the login shell, then passing it to `sort` and `uniq` to identify how many occurrences of each there are.

```
(taz@dagger)-[~]
$ cut -d: -f7 /etc/passwd | sort | uniq -c
      1 /bin/bash
      5 /bin/false
      1 /bin/sh
      1 /bin/sync
      2 /usr/bin/zsh
  49 /usr/sbin/nologin
```

awk: Powerful Data Manipulation and Pattern Matching

Examples

- awk '/c99/ {print \$1 "\t" \$7}' access.log
Search the HTTP access log for strings matching c99, print the originator IP, and request string for any matches.
- awk '{print \$6}' access.log
Identify the HTTP request methods in a server access log.
- awk '/GET/ {++cnt} END {print "Total = ", cnt}' access.log
Count the number of GET requests in the HTTP access log.

Use Notes

- It is a very powerful tool, but it requires practice to use well.

Awk is actually a scripting language designed to manage actions against a stream of data (normally text-based). It is powerful and very complex, with its own user's guide,¹ and could have an entire course dedicated to getting the most out of it. For brevity, we will focus on using the awk command line to process and transform text-based input in this course.

As a command, awk can be run against files directly or used to process the output of other commands in a pipeline. This allows investigators to build advanced queries to quickly identify elements of interest in large data sets.

An example of this is an investigator using awk to check an Apache 2 access.log to find references to the C99 shell, then returning the source IP address and the request string used, filtered through uniq to remove duplicates.

```
root@midgard:/var/log/apache2# awk '/c99/ {print $1 "\t" $7}' access.log | sort | uniq
17.22.56.28    /c99.php
54.14.12.51    /c99.php
55.102.158.83  /c99.php
55.122.188.8   /c99.php
```

1. The GNU Awk User's Guide <https://for577.com/gnuawk>

xxd: Making Hexdumps

Examples

- `xxd unknownfile | head -n 10`
Convert `unknownfile` to hex and show the first ten lines.
- `xxd -s 3122 -l 256 diskimage.E01`
Show 256 bytes of hex, starting at byte `3122` from `diskimage.E01`.

Useful Switches

- `-s` Start X bytes into the file (seek).
- `-l` Show X bytes in total (length).
- `-r` Convert hexdump back into a binary format (reverse).

The binary `xxd` is a built-in tool that takes an input source (file or STDIN) and converts it to hexdump format. This is not the same as a pure hex editor, but it provides a quick way to read the underlying hex of a file. By default, output is sent to STDOUT, but it can also be redirected to a file.

An example is using `xxd` to check the file type by reviewing the file signature.

```
user@midgard:~$ xxd /usr/bin/ls | head
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000 .ELF.....
00000010: 0300 3e00 0100 0000 b06a 0000 0000 0000 ..>.....j...
00000020: 4000 0000 0000 0000 2014 0200 0000 0000 @.....@.....
00000030: 0000 0000 4000 3800 0d00 4000 1f00 1e00 ....@.8...@...
00000040: 0600 0000 0400 0000 4000 0000 0000 0000 .....@.....
00000050: 4000 0000 0000 0000 4000 0000 0000 0000 @.....@.....
00000060: d802 0000 0000 0000 d802 0000 0000 0000 .....@.....
00000070: 0800 0000 0000 0000 0300 0000 0400 0000 .....@.....
00000080: 1803 0000 0000 0000 1803 0000 0000 0000 .....@.....
00000090: 1803 0000 0000 0000 1c00 0000 0000 0000 .....@.....
```

Typical syntax is:

`xxd {switches} {input source}`

hexdump: Display File Contents in Hex (or octal Etc.)

Examples

- `hexdump -C unknownfile`
Show unknown file in hexadecimal with ASCII conversion.

Useful Switches

- `-C` Canonical hex + ASCII display, which is similar output to `xxd`.
- `-n [number]` Interpret only [number] bytes of input.
- `-s [number]` Skip [number] of bytes from the start.

Note: By default, hexdump on Linux displays output in little-endian format.

In practice, this is very similar to the `xxd` command in that it takes an input source and converts it to hexadecimal output. However, it can be used to display file contents in octal, decimal, or ASCII. The most significant difference is that `hexdump` shows the file content in endian format of the host operating system, unless the `-C` argument is used. On Linux, this nearly always means it is little-endian. With `-C`, the output is big-endian, followed by an ASCII representation of the file contents.

This is an example of using `hexdump` to read the `/usr/bin/ls` file as seen on the `xxd` page.

```
sansforensics@siftworkstation: ~
$ hexdump /usr/bin/ls | head
00000000 457f 464c 0102 0001 0000 0000 0000 0000
00000010 0003 003e 0001 0000 6ab0 0000 0000 0000
00000020 0040 0000 0000 0000 1420 0002 0000 0000
00000030 0000 0000 0040 0038 000d 0040 001f 001e
00000040 0006 0000 0004 0000 0040 0000 0000 0000
00000050 0040 0000 0000 0000 0040 0000 0000 0000
00000060 02d8 0000 0000 0000 02d8 0000 0000 0000
00000070 0008 0000 0000 0000 0003 0000 0004 0000
00000080 0318 0000 0000 0000 0318 0000 0000 0000
00000090 0318 0000 0000 0000 001c 0000 0000 0000
```

If you compare this with the `xxd` output, you can see that the order of the bytes is switched. The file signature for an ELF file is `7f 45 4c 46`, but the little-endian format of `hexdump` means this is rendered `457f 464c`.

This can be misleading, so if you aren't confident about the data source, it is often better to use the `-C` argument.

With -C specified, hexdump displays the output in 16 bytes, displayed in two 8-byte groups followed by an ASCII representation of the hexadecimal code. More importantly, it also renders the hex in a consistent format rather than using the host operating system's defaults.

This is an example of using hexdump -C to review the same file as before.

```
$ hexdump -C /usr/bin/ls | head
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00 | .ELF.....
00000010  03 00 3e 00 01 00 00 00  b0 6a 00 00 00 00 00 00 | ..>...j...
00000020  40 00 00 00 00 00 00 00  20 14 02 00 00 00 00 00 | @.....@...
00000030  00 00 00 00 40 00 38 00  0d 00 40 00 1f 00 1e 00 | .....@.8..@...
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00 | .....@...
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00 | @.....@...
00000060  d8 02 00 00 00 00 00 00  d8 02 00 00 00 00 00 00 | .....@...
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00 | .....@...
00000080  18 03 00 00 00 00 00 00  18 03 00 00 00 00 00 00 | .....@...
00000090  18 03 00 00 00 00 00 00  1c 00 00 00 00 00 00 00 | .....@...
```

This can be easier to read and understand, especially if you come from a Microsoft Windows background.

Typical syntax is:

```
hexdump {switches} {input source}
```

Hashing: MD5, SHA1, SHA256

Examples

- `md5sum filename` (Generate an MD5 hash of filename.)
- `sha1sum filename` (Generate a SHA1 hash of filename.)
- `sha256sum filename` (Generate a SHA256 hash of filename.)

Use Notes

- MD5 generation is often the fastest, but the hash is cryptographically weak.
- Different tools accept different hashes; consider generating all three.
- For most DFIR purposes, MD5 is enough to confirm that a file is identical or unchanged from a reference file.

In this context, a hash is a one-way cryptographic function that takes a variable-length input, performs some mathematical functions on it, and produces a fixed-length output string with the results. The concept relies on the fact that any changes to the input will result in changed output.

Hashing has multiple uses in security, but for our purposes, its primary use is to provide a representation of a given file. This allows investigators to check to see if the file has been changed at any time (any changes to the file contents will result in a new file hash) or to search public repositories to see if the file is known or has been reported.

A good example of this is searching VirusTotal (VT). Often, uploading malware samples will alert the attackers that their tools have been discovered, so investigators prefer to take a file hash and search VT to see if it is already there.

Output Lengths

Hashing produces fixed-length outputs. The most common functions produce output as follows:

MD5	32 hex characters (16 bytes or 8 bits)
SHA1	40 hex characters (20 bytes or 160 bits)
SHA256	64 hex characters (32 bytes or 256 bits)

Important Considerations

- MD5 is considered cryptographically weak. This means there are known attacks that allow an adversary to generate a hash collision, where two different inputs result in the same output. As a result, this hash is deprecated in most security tools and is not recommended for any cryptographical solution that attempts to protect the confidentiality of a system.

Although this is important to be aware of, it is less of an issue for file hash comparisons, so for DFIR uses, this may still be an acceptable hash algorithm for your use. This is because the known attacks involve very specific techniques (such as bit flipping at set offsets) rather than allowing arbitrary modification of a file's contents.

- Hashing takes time. Creating a hash of a small text file is almost instant with any hashing algorithm, but this becomes noticeable as the file gets bigger. Creating a SHA256 hash of a 1TB disk image is likely to take several hours, during which time no analysis can take place.

As a result of this, it is often more efficient to use MD5 for evidence integrity, but this will depend on your organizational policies and standards. You may also be constrained by the tools you use, as some products will only accept a specific type of hash. Where possible, taking all three hashes is useful, but this may be impractical.

The date Command: Print or Set the System Time

date [options]

Examples

- `date -u -d @1692564334`
This will convert the Unix epoch timestamp "1692564334" to a human-readable timestamp in UTC.

Useful Options

- `-d [STRING]` (Display the date referenced by STRING, not "now.")
- `-u` (Display the time in UTC.)

Use Notes

- This allows investigators to quickly convert between time formats.

The date command in Linux is used to either set the system time to a specified value or display the current system time. However, by passing it the `-d` argument, it can also convert the timestamp from one format to another. This is commonly used to convert between Unix epoch timestamps, which count in nanoseconds since midnight, January 1, 1970, to a more human-readable format.

Typical syntax is:

```
date [options] [output format options]
```

The output format can be specified in a variety of ways. The default is WEEKDAY MONTH DAY HH:MM:SS [AM/PM] TZ YEAR [shown below], but this can be modified by command-line options.

```
root@siftworkstation:~# date -u -d @1692564334
Sun Aug 20 08:45:34 PM UTC 2023
```

A full list of formatting options is available by running `date --help`.

Signaling processes with the kill command

kill [-signal] [pid | name]

Examples

- `kill 988` (This will send a TERM signal to the process with PID 988, shutting it down.)

Useful Options

- `-s` Send a specific signal name or number.
- `-1` Send the signal to all processes with a PID greater than 1.
- `-l` Print a list of signal names.

Notes

- This can be used for more than killing processes and is often abused by Linux rootkits.

Although it is often associated with a way of ending running processes, the `kill` command is actually used to send a "signal" to the running process. A signal can be specified by either a number or the signal name itself. If the command is used without specifying a signal, it defaults to signal 15, or SIGTERM, and this is interpreted as a request to terminate the running process.

The typical syntax to terminate a running process (in this example, PID 9990) is:

`kill 9990`

or

`kill -s 9 9990`

Most of the time, `kill` is used to send a SIGTERM or SIGKILL (signal 9) message—the difference is that SIGTERM is a "polite" request for the process to terminate and leads to a graceful shutdown, cleaning up files and so on. It can be ignored by the process and may not work on crashed processes. The more aggressive approach is to send a signal 9, SIGKILL, message, which cannot be caught or blocked by the process and thus forces a rapid shutdown.

Using SIGKILL can lead to problems, though, as the process is unable to clean up any files/handles and may lead to resource leaks.

In addition to SIGKILL/SIGTERM, a lot of other signals can be sent. The exact number does depend on the Linux distro in use, the shell in use, the Kernel version, etc. The POSIX standard defines 31 signals,¹ but most modern Linux platforms support more (e.g., Ubuntu 22.04 supports 62 signals).

Signals above 31 tend to be allocated to SIGTRMAX and SIGTRMIN calls. These are generally accepted to be user-defined signals, used for real-time interprocess communication. An example of this is that Ubuntu Linux treats signal 36 as SIGTRMIN+2, which is literally SIGTRMIN (34) plus 2.

In the early days of Linux (and its Unix ancestors), each signal had a well-defined meaning,² but this is less enforced in modern distros. An example of this is Signal 1 – SIGHUP. This is the "Hang Up" signal and was traditionally sent to a process when its controlling terminal exited. The original concept was that the phone or modem making the connection had literally hung up and severed the connection. Today, most shells will treat this as an instruction for the daemon to reload its configuration files. However, if sent to a running process, it is likely to be treated as a terminate signal.

Practically speaking, most POSIX-compliant signals will be treated as either a stop or terminate instruction by running processes, although they can have different error-handling routines in place. In addition to the `kill` command, there are keyboard sequences that can produce signals as well—the most common example is using `CTRL+C` in Linux, which sends a SIGINT signal and, by default, causes the process to terminate.

In the POSIX-complaint list, Signals 10 and 12 are "User Defined" (SIGUSR1 and SIGUSR2), and when sent to a process, they trigger a "user-defined condition." This concept of user-defined signal handling is frequently exploited by Linux rootkits as an easy way to hook/intercept interprocess communication.

This was most famously done by the diamorphine rootkit,³ which allowed the operator to send signal 63 to any PID to make the attack module invisible, or signal 64 to elevate the user account to root. This attack behavior (and the diamorphine rootkit itself) is still found in intrusions today and can be found as a component of advanced/stealth attack kits, such as with the Krasue⁴ malware family used to target Thai telecoms firms, discovered in 2022.

1. <https://for577.com/posixsignals>
2. <https://for577.com/signalipc>
3. <https://for577.com/diamorphine>
4. <https://for577.com/krasue>

Getting Help – man

man [command]

Examples

- `man ping` (This will display the manual pages for the ping command.)

Useful Options

- `-k [keyword]` Search the manual pages for entries that refer to the keyword.
- `-f [command]` Display a concise description of the command.

The `man` command is used to display the system's manual page for a given command. This is effectively rendering a help file providing guidance on how the command is used. The output can be very verbose and is normally presented in "pages," allowing you to search or scroll through the output.

Typical syntax is:

`man [command or function]`

An example of the output (for `man ping`) is shown below.

```
PING(8)                               iputils                               PING(8)

NAME
    ping - send ICMP ECHO REQUEST to network hosts

SYNOPSIS
    ping [-aAbBdDfhLnOqrRuVv46] [-c count] [-F flowlabel] [-i interval] [-I interface] [-l preload] [-m mark]
          [-M pmtdisc_option] [-N nodeinfo_option] [-w deadline] [-W timeout] [-p pattern] [-q tos] [-s packetsize]
          [-S sndbuf] [-t ttl] [-T timestamp option] {hop...} {destination}

DESCRIPTION
    ping uses the ICMP protocol's mandatory ECHO REQUEST datagram to elicit an ICMP ECHO RESPONSE from a host or gateway.
    ECHO REQUEST datagrams ("pings") have an IP and ICMP header, followed by a struct timeval and then an arbitrary number of
    "pad" bytes used to fill out the packet.

    ping works with both IPv4 and IPv6. Using only one of them explicitly can be enforced by specifying -4 or -6.

    ping can also send IPv6 Node Information Queries (RFC4620). Intermediate hops may not be allowed, because IPv6 source
    routing was deprecated (RFC5095).

OPTIONS
    -4
        Use IPv4 only.

    -6
        Use IPv6 only.

    -a
        Audible ping.

Manual page ping(8) line 1 (press h for help or q to quit)
```



Lab 1.2: Introduction to Linux Commands

Estimated time: 30 minutes

This page intentionally left blank.

FOR577.I

Linux Incident Response and Threat Hunting



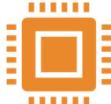
Package Management

SANS | DFIR

FOR577 | Linux Incident Response and Threat Hunting 84

This page intentionally left blank.

Linux Package Management



Default way to manage applications and OS



Helps users search, install, uninstall, and update applications



Command line or GUI based

At a high level, package management refers to how operating systems (and maintainers) manage and automate the process of installing, upgrading, and removing software applications on an endpoint. The term *packages* is used here to mean a collection of software, often in an archive file format, along with metadata that allows the management tool to understand how to install, configure, or remove the application. Package management tools exist on all operating systems (Windows has WinGet and the Microsoft Store, for example), but they are most often associated with Linux platforms.

Package management tools have been around since the earliest days of Linux, with dpkg existing in 1994.¹ Since around 1995, they have been able to automatically resolve dependencies. This means that when a package is installed, the management tool can check the metadata and find out what other packages are required. If any are missing, the tool either automatically resolves this by installing the missing packages or lets the user know and allows them to decide what actions should be taken.

1. The dpkg source code for Debian 0.93.15: <https://for577.com/dpkgsrc>

Package Managers and DFIR

Good

- Leave evidence on applications.
- Can be used to profile devices.

Bad

- There are lots of choices.
- Distro defaults can be overridden.

Package managers have good and bad points when it comes to DFIR investigations.

Good Points

- **Evidence:** Using package management tools often leaves considerable quantities of evidence on an endpoint and the network. This includes updating the repository lists, which can also include DNS lookups, scanning the existing installed software, and even filesystem modification events. Attackers have even been identified installing new applications on endpoints, possibly to ensure they can facilitate their chosen attack paths.
- **Profiling:** Even if the attackers haven't installed their own tooling, checking the package management tool can identify what applications were available to the attackers and what activities the user is likely to have carried out.

Bad Points

- **Too many choices:** There is a massive range of package management tools. Each one generates its own artifacts, and it is unrealistic to expect an incident responder to become an expert in how each one works. This creates a heavy reliance on learning the defaults for the distributions in use in each organization.
- **Defaults can be changed:** Unfortunately, it is pretty easy for administrators to change the defaults. This makes it essential for IR teams to ensure that they get a good understanding of the environment and how it is configured during the preparation stage of the IR cycle.

Package Manager Overview: Part I

apt

- Advanced package tool
- Runs on Debian devices
- Default on Ubuntu

dpkg

- Debian package format
- Low-level tool for Debian systems
- Rarely used directly now, often just manual installations

rpm

- RPM Package Manager
- Drives other package managers on Red Hat distros
- Found on RHEL/CentOS



Over the next three pages, we will look at some of the main package managers available. This is just a sample of the possible package managers, and it is important for responders to understand their environments.

apt

The Advanced Package Tool is the current default on Ubuntu systems. It acts as a front end for Debian packages and has largely supplanted direct installation via dpkg. It enhances the functionality of dpkg by managing relationships (dependencies). This means users don't need to worry about checking that they've installed all the linked packages an application needs; for correctly crafted packages, apt resolves this. Apt also sorts any packages to be installed, ensuring they are installed in the best possible order, minimizing the risk of issues.

dpkg

Dpkg is a tool used to install and manage software applications in Debian-based systems. It was one of the first package management tools in Linux and is still used to install stand-alone packages. Unlike modern, higher-level tools, dpkg only really supports users installing individual packages or .deb files. This means users are responsible for ensuring that dependencies are met and updates/upgrades must be manually installed. It is uncommon to see dpkg used as a package management tool today; more often, it's a sign that the user has downloaded a stand-alone package from the internet.

rpm

Originally the Red Hat Package Manager, this is now a recursive acronym for RPM Package Manager. This is the default package format for Red Hat-based distros, effectively equivalent to dpkg on Debian. Like dpkg, it is a low-level tool and rarely seen in general use, other than users downloading stand-alone .rpm packages for installation.

Package Manager Overview: Part 2

yum

- Yellowdog Updater, Modified
- Runs on top of rpm
- Found on RHEL/CentOS

dnf

- Dandified YUM
- Rewrite of YUM
- Default on Fedora

apk

- Alpine Package Keeper
- Lightweight tool
- Found on Alpine Linux



yum

Standing for Yellowdog Updated, Modified, yum is a high-level wrapper for the rpm tool. It manages dependencies, so the user simply installs the application they want, and yum will ensure the best chance of success. Yum has been the default on Red Hat Enterprise Linux and CentOS since version 5 and is widely used on related distros. Yum is a Python 2-based tool with some additional libraries to provide a command-line interface (CLI). As a result, distros will likely move away from this to avoid being tied to Python 2.

dnf

This package manager is an update to yum, called Dandified YUM, created for Fedora 22, moving away from running on Python 2 to the Libdnf framework. In an effort to speed up dependency management, dnf uses libssolv rather than internal resolver libraries. The tool also uses librepo to manage repositories and libcomps for file manipulation, further removing the reliance on Python.

apk

This is a lightweight package management tool used on Alpine Linux and designed for use in low-resource environments such as containers. Although the command syntax does vary from other package managers, the basic functionality is the same.¹

1. <https://for577.com/alpinepkg>

Package Manager Overview: Part 3

snap

- Developed by Canonical
- App store and package manager
- Uses a compressed filesystem format

synaptic

- GUI for APT
- Used on Debian systems
- Uses deb and rpm packages

zypper

- Zen/YaST Packages Patches Patterns Products
- Uses rpm-based packages
- Found on openSUSE and SUSE Linux Enterprise

snap

Snap is a store application developed by Canonical to manage package deployments. Often used at the command line, it is also possible to install a GUI Snap Store application. The Snap Store deploys packages called "snaps," which are self-contained packages designed to work in multiple environments. The file format is a self-contained compressed filesystem using SquashFS. The snap contains the application, libraries, and metadata to instruct the host on how to install the package. The files are decompressed and mounted at runtime, meaning they take up less disk space at the cost of slower start times.

Snap applications run in a sandbox environment with limited access to the host. This provides an element of security against malicious apps but can be bypassed by determined attackers.

synaptic

This is a graphical user interface for the apt package management tool. It is less commonly found now, as most users wanting a GUI use the Snap Store.

zypper

This is the command-line interface for YaST (Yet another Setup Tool) and has been found on SUSE-based distros since around 2006, when zypper was still in beta. Version 1.0 of zypper was released in 2008 along with OpenSUSE 11.1. Initially, zypper struggled with speed, as its dependency solver was reportedly very slow. However, this is being addressed with the implementation of a new solver based on libolv that, in benchmarks at least, outperforms RPM-based solutions.

Package Formats

Format	Distro
.deb	Debian based: Debian, Ubuntu, Kubuntu, Mint, LinuxMCE, Kali, Parrot, Vyatta
.rpm	Red Hat based: RHEL, CentOS, Amazon Linux, Rocky Linux, Fedora, Oracle Linux, OpenSUSE, SUSE Linux Enterprise Server
.apk	Alpine

This is a summary of the package formats commonly found on Linux distros.

There are two main formats in use:

1. .deb. This is used by Debian-based distros, most commonly Ubuntu.
2. .rpm. This is used by Red Hat-based distros, which is almost everything else in Linux. This includes Red Hat derivatives such as Rocky Linux, Oracle Linux, and Amazon Linux. Although SUSE uses its own package management tools, SUSE uses the RPM format packages.

There are some other formats you may encounter. Alpine Linux uses an APK format for its packages, which is really a digitally signed tar.gz file. Slackware also uses gzip'd tar files, although normally, these have the .tgz extension.

Package Management Defaults

Package Manager	Distros
apt	Ubuntu, Debian, Kali, Parrot, Knoppix, Raspberry Pi OS
Snap	Ubuntu
yum	Fedora <21, CentOS 5+, RHEL 5+, Oracle Linux, Amazon Linux
dnf	Fedora 22+
zypper	OpenSUSE, SUSE Linux Enterprise
apk	Alpine Linux

This slide identifies default package managers in a given Linux distribution.

It is important to note, however, that defaults are easy to change within a given format. For example, Ubuntu ships with the apt package manager, but users can, if they wish, use synaptic, aptitude, or even dpkg.

While it is theoretically possible, it is very unusual to see users running package managers for a different format. The dependencies inside each application make it challenging to install across distros. There are converters available, and for users who must install (for example) an RPM package on an Ubuntu system, the best approach is to convert it into a .deb file first rather than try to get yum working.

Command Comparisons

Common package management commands

Action	APT	DNF	ZYPPER	APK
Install PackageName	apt install PackageName	dnf install PackageName	zypper in PackageName	apk add PackageName
Update Package Database	apt update	dnf check-update	zypper ref	apk update
Remove PackageName	apt remove PackageName	dnf remove -- nodeps PackageName	zypper rm -RU PackageName	apk del PackageName
Upgrade all packages	apt upgrade	dnf update	zypper up	apk upgrade
List available upgrades	apt list -- upgradeable	dnf check-update	zypper lu	apk info (see notes)

Package management tools are used for common tasks, no matter what the distro is, and this table provides an example of the varied command syntax for the most common tasks.

Key points:

- All need to be run with superuser privileges to work properly.
- The dnf tool merges an update of the package database by showing the user what needs to be upgraded. With apt/zipper, this requires extra commands.
- The unusual nature of Alpine Linux (running in RAM) means the package management commands are slightly different. The use of apk info is the most obvious, as this returns information on all packages, installed and available ones. On other tools, it is possible to just return information on packages with an available update.

Package sources - Debian

Distro default packages: /etc/apt/sources.list

```
root@asgard:~# cat /etc/apt/sources.list | grep -Ev '^#[ ]|^$'
deb http://gb.archive.ubuntu.com/ubuntu/ jammy main restricted
deb http://gb.archive.ubuntu.com/ubuntu/ jammy-updates main restricted
deb http://gb.archive.ubuntu.com/ubuntu/ jammy universe
deb http://gb.archive.ubuntu.com/ubuntu/ jammy-updates universe
deb http://gb.archive.ubuntu.com/ubuntu/ jammy multiverse
deb http://gb.archive.ubuntu.com/ubuntu/ jammy-updates multiverse
deb http://security.ubuntu.com/ubuntu jammy-security main restricted
deb http://security.ubuntu.com/ubuntu jammy-security universe
deb http://security.ubuntu.com/ubuntu jammy-security multiverse
```

/etc/apt/sources.list.d/ can also be used—normally for third-party applications or where there are multiple repo sources.

Debian-based package management tools, such as apt, use centralized repositories to retrieve software. These are listed in one of two locations:

- /etc/apt/sources.list. This is a text file that typically contains the repositories used as defaults by the distro but can also include repositories manually added by the user. Generally, users should avoid manually adding repos in this way.
- /etc/apt/sources.list.d/. This folder contains ".list" files, which contain repository information for additional repos added by the user. Files here are created when the user runs commands such as apt-add-repository, and this is the preferred way to manage individual sources.

Here is the general format of a list file:

```
deb [options] uri distribution [component1] [component2] [...]
deb-src [options] uri distribution [component1] [component2] [...]
```

The fields are as follows:

- Deb indicates a binary package repository that provides precompiled software.
- Deb-src is used to indicate a source code repository.
- Options can include details such as architecture-specific details or any other parameters for the repo.
- The URI is the base URI for the repo.
- Distribution is the version codename (focal, jammy, etc.) used by the repository.
- The components comprise a space-delimited list showing the types of applications. This will use terms like main, contrib, non-free, multiverse, etc. Generally, this relates to the license models used. For example, "non-free" means software that is not compliant with the Debian Free Software Guidelines, "restricted" can include proprietary applications that are not supported by the distribution vendor, etc.

Debian (and Ubuntu) family systems use the following component terms:

- main: Officially supported free and open-source software
- contrib: Free software that depends on non-free software
- non-free: Software that is not compliant with the Debian Free Software Guidelines (DFSG)
- restricted: Non-open-source software supported by Ubuntu
- universe: Community-maintained free and open-source software
- multiverse: Software that is not free or supported under a free license

Licensed To: Kaleb Sego <kaleb_sego@outlook_com> June 12, 2025

Package sources – Red Hat Family

Repository list: /etc/yum.repos.d/ (multiple *.repo files)

```
[rhel-7-server-rpms]
name=Red Hat Enterprise Linux 7 - Server
baseurl=https://cdn.redhat.com/content/dist/rhel/server/7///os
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

RHEL



CentOS

```
[baseos]
name=CentOS Linux $releasever - BaseOS
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=BaseOS
&infra=$infra
gpgcheck=1
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
```

The Red Hat family also uses a centralized list for software packages, although this is normally stored as multiple .repo files in the /etc/yum.repos.d/ folder. Despite the name, this location is used by both yum and dnf.

The typical elements of a repo file are as follows:

- Repository ID - This is a unique identifier for the repository and is enclosed in square brackets.
- name - A human-readable name that's displayed when listing repos. Can include variables.
- baseurl - This is where the repo packages are located.
- mirrorlist - This is where any alternative URLs to download files are listed.
- enabled - A 1 shows the repo is enabled and will be used; a 0 means it is disabled.
- gpgcheck - A 1 means GPG signature checking will take place on packages; a 0 disables it.
- gpgkey - Specifies the location of the GPG key to be used for verification.

Other elements can include:

- Exclude - A list of packages to exclude from updates or builds.
- Include - A list of packages to include from the repository, ignoring others.
- Cost - The relative cost of using the repo. Lower costs are preferred.
- Priority - The priority of this repo (lower numbers are higher priority). This is not always respected.
- Sslverify - Enable or disable SSL certificate verification.

RHEL Variations – SUSE / OpenSUSE using Zypper

Repo files stored in /etc/zypp/repos.d/

```
[repo-oss]
name=Main Repository (OSS)
enabled=1
autorefresh=1
baseurl=http://download.opensuse.org/distribution/leap//repo/oss/
path=
type=rpm-md
priority=99
gpgcheck=1
gpgkey=http://download.opensuse.org/distribution/leap//repo/oss/repo/repodata/repomd.xml.key
```

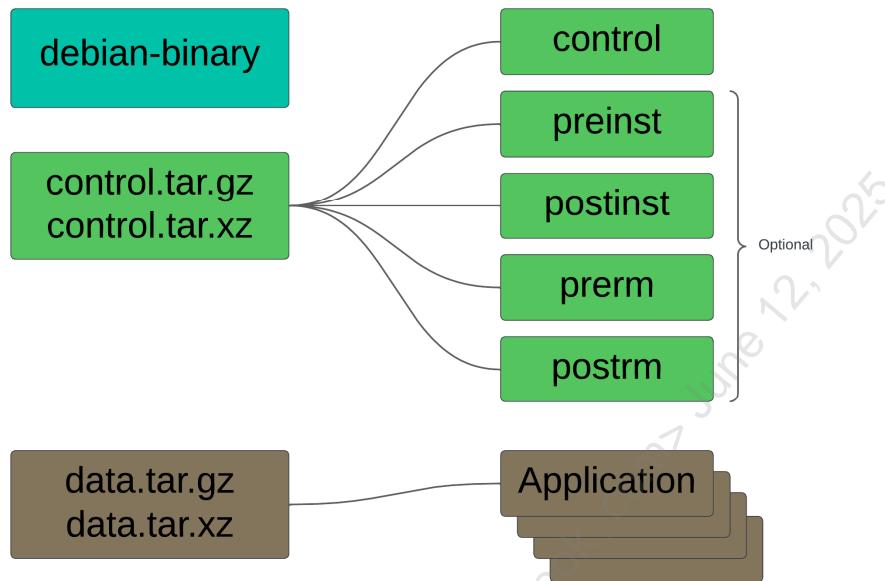


Zypper, used by SUSE Linux, is very similar to Red Hat, but the files are stored in /etc/zypp/repos.d/. The structure of the files is largely identical, but SUSE generally makes use of different options.

The additional elements commonly found in SUSE are as follows:

- autorefresh - This specifies if the repo metadata should be automatically refreshed on each access. A 1 means it will be refreshed each time. 0 means the user must perform a manual refresh.
- type - The type of repo. For example, rpm-md is RPM Metadata.
- path - The path to data within the repo.

Debian Package Structure



A Debian package is essentially an `.tar` archive designed for software installation, which means it has some predetermined components. These sub-file structures are there to support installers and provide easy installation/upgrade and removal of applications.

There are three sub-file structures in a Debian package:

- **debian-binary:** This is a plaintext file that simply contains the version of the Debian package format. As of most versions, this file contains a single line of text, such as "2.0", indicating the package format version. This file serves as a marker that the file is indeed a Debian package and helps the package management system process it correctly.
- **control.tar.gz or control.tar.xz:** This compressed archive contains metadata and control information about the package. Inside, you'll find various control files, including:
 - **control:** Contains package metadata such as the package name, version, dependencies, conflicts, package description, and other essential data that the package management system uses to handle the package.
 - **preinst, postinst, prerm, and postrm:** Optional scripts that are executed before installation (preinst), after installation (postinst), before removal (prerm), and after removal (postrm) of the package. These scripts can handle tasks such as configuring the package, making necessary system modifications, and cleaning up after a package is removed.
 - Other files might be included for specific package configuration or maintenance tasks.
- **data.tar.gz or data.tar.xz:** This archive contains the actual files to be installed on the system, organized in a directory structure that mirrors their intended destination on the target filesystem. For example, binaries that should be installed to `/usr/bin` are stored in a `usr/bin` directory within this archive.

Note: The control and data objects can have a range of file extensions. The `.tar.xz` and `.tar.gz` examples are just the most common; you can also see `.tar.zst`, `.tgz`, etc.

Analyzing .deb Files in Hex

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00000000	21 3C 61 72 63 68 3E 0A	!<arch>.debian-b
00000010	69 6E 61 72 79 20 20 20	inary 16506033
00000020	35 36 20 20 30 20 20 20	20 20 20 30 20 20 20 20
00000030	31 30 30 36 34 34 20 20	34 20 20 20 20 20 20 20
00000040	20 20 60 05 82 2E 30 0A	63 6F 6E 74 72 6F 6C 2E
00000050	74 61 72 2E 7A 73 74 20	31 36 35 30 36 30 33 33
00000060	35 36 20 20 30 20 20 20	20 20 30 20 20 20 20 20
00000070	31 30 30 36 34 34 20 20	31 30 37 33 39 34 36 20
00000080	20 20 60 0A 28 B5 2F FD	00 68 D4 26 03 FE 4E C8
00000090	36 11 22 F0 76 8A 3A 4C	60 1E 6D F1 FF 1F FB 50
000000A0	14 5D A4 4C 99 1B 5E 28	93 D6 65 59 A3 0E B3 33
000000B0	F0 0C 1C A6 3A FE 12	51 11 41 11 19 18 51 44 E3
000000C0	1C 72 D5 17 A0 85 10 AF	9D 84 73 0A 3F 55 5A C7
000000D0	18 5A 7B 4E 42 FF DB 88	02 69 05 23 4B 73 0F 6B
000000E0	1F 34 59 04 18 74 00 82	E1 0A 4F 13 AE 82 03 7B
000000F0	F5 88 D0 AA 55 03 E7 79	16 C3 A5 A2 61 FE 84 12
00000100	94 88 24 EB B2 4D 7C	7B 0E 9C 24 EA B1 9E B1 D1
00000110	5F E6 7F C9 30 3A 64 0C	F1 27 2C ED 43 78 4B 6B
00000120	B2 28 BF 74 5C 4F C3 52	AE E0 03 CC 4C A5 33 B3
00000130	96 61 4F 02 9E AC 88 CE	FB CF BD 73 BC B1 4D A8
00000140	79 EC 67 D2 EA D4 32	65 0F B4 31 D0 AA 60
00000150	D1 91 B7 88 5A BC 6B	B5 45 CE 65 D1 1B 2A 39 7E
00000160	0A 4A D4 EC 16 6E 99	0C 93 AF 70 5B C1 67 F0 20
00000170	03 CC 36 7D 12 DA 16 89	41 69 43 63 6E A9 E3 33

File signature

File mod timestamp

Version

Sub-file identifier

Sub-file timestamp

File size (Bytes)

One way to analyze packages is to use a hex editor, as shown here. This allows the analyst to validate the contents of the file and identify each sub-file.

It is worth noting that each sub-file has its own identifier (file name) and comes with some metadata that might be useful for an investigator—specifically, a timestamp and file size. The timestamp is normally the time the file was added to the archive (effectively a "file modification" timestamp, as covered in Section 2).

As each sub-file has its own timestamp, all timestamps should reflect the last modification time of the archive. Normally, this is when the archive package was created. If there is a mismatch between timestamps on an individual sub-file, this might indicate tampering or package corruption.

Viewing and Extracting the Contents of a Debian Archive

`ar -tv [package]`

```
sansforensics@siftworkstation: ~/examine
$ ar -tv pdftk-java_3.2.2-1_all.deb
rw-r--r-- 0/0 4 Feb 4 20:12 2021 debian-binary
rw-r--r-- 0/0 1472 Feb 4 20:12 2021 control.tar.xz
rw-r--r-- 0/0 940772 Feb 4 20:12 2021 data.tar.xz
```

Examine
the archive

Extract the
archive

`ar -xv [package]`

```
sansforensics@siftworkstation: ~/examine
$ ar -xv pdftk-java_3.2.2-1_all.deb
x - debian-binary
x - control.tar.xz
x - data.tar.xz
```

Debian packages (the .deb files) are in a standard archive format and can be analyzed with the `ar` command. This is a program for creating, modifying, and extracting objects from archives in a similar manner to the `tar` command. The main differences are conceptual—the `ar` command is mainly used to create static files, often applications, while `tar` is more traditionally used for archives and backups. Also, `ar` does not have a built-in mechanism for compression, while `tar` supports gzip, bzip2, and xz compression via command-line arguments. This means that `ar` archives can end up being significantly larger than their `tar` equivalents. Performance-wise, the biggest difference is that `ar` creates an index at the beginning of the archive. This is a plaintext index and speeds up random access to objects within the archive file itself.

Here are some useful command-line arguments for analyzing packages:

- t Display a table listing the contents of the archive.
- x Extract files from the archive. If a filename is given, this is the only file to be extracted. If no filename is provided, all files will be extracted.
- o Preserve original dates when extracting files.
- v Verbose mode.

Note:

The `ar` tool is not installed by default on Ubuntu and may require the `binutils` package to be installed before you can use it.

Listing Application Files with dpkg

dpkg -c [package]

```
sansforensics@siftworkstation: ~/examine
$ dpkg -c pdftk-java_3.2.2-1_all.deb
drwxr-xr-x root/root      0 2021-02-04 20:12 .
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/bin/
-rw-r--r-- root/root    169 2021-02-04 17:45 ./usr/bin/pdftk.pdftk-java
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/doc/
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/doc/pdftk-java/
-rw-r--r-- root/root     855 2021-02-04 20:12 ./usr/share/doc/pdftk-java/changelog.Debian.gz
-rw-r--r-- root/root   3982 2021-02-04 20:12 ./usr/share/doc/pdftk-java/copyright
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/man/
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/man/man1/
-rw-r--r-- root/root   7582 2021-02-04 20:12 ./usr/share/man/man1/pdftk.pdftk-java.1.gz
drwxr-xr-x root/root      0 2021-02-04 20:12 ./usr/share/pdftk/
-rw-r--r-- root/root  975561 2021-02-04 20:12 ./usr/share/pdftk/pdftk.jar
```

The `dpkg` command can also be used to do some analysis on Debian packages, although it doesn't provide a simple way to show the debian-binary or control sub-files without extracting them. Despite this limitation, it can be a quick way to view the contents of the data.tar sub-file and show all the objects that the package would try to install, as well as the location on disk they would be written to.

For DFIR purposes, the main command-line arguments are as follows:

- c/--contents List the contents of the data.tar.[gz|xz etc] sub-file.
- x/--extract Extract the contents of the data.tar.[gz|xz etc] sub-file to a specified location on disk.
- control Extract the control.tar.[gz|xz| etc] sub-file and its contents to a folder called DEBIAN.
- I/--info Display any metadata associated with the package.

Package Metadata with dpkg -I

```
sansforensics@shiftworkstation: ~/examine
$ dpkg -I info pdftk-java 3.2.2-1_all.deb
new Debian package, version 2.0.
size 942436 bytes: control archive:1472 bytes.
1329 bytes, 32 lines    control
342 bytes, 5 lines    md5sums
534 bytes, 25 lines * postinst          #!/bin/sh
273 bytes, 20 lines * preinst          #!/bin/sh
Package: pdftk-java
Version: 3.2.2-1
Architecture: all
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Johann Felix Soden <johfel@debian.org>
Installed-Size: 979
Depends: libbcprov-java, libcommons-lang3-java, default-jre-headless | java8-runtime-headless
Suggests: xpdf-utils
Breaks: pdftk (<< 2.02.5-)
Replaces: pdftk (<< 2.02.5-)
Provides: pdftk
Section: text
Priority: optional
Homepage: https://gitlab.com/pdftk-java/pdftk
Description: pdftk port to Java - a tool for manipulating PDF documents
If PDF is electronic paper, then PDftk is an electronic stapler-remover,
hole-punch, binder, secret-decoder-ring, and X-Ray-glasses. PDftk is a
simple tool for doing everyday things with PDF documents. Keep one in the
top drawer of your desktop and use it to:
- Merge PDF documents
- Split PDF pages into a new document
- Decrypt input as necessary (password required)
- Encrypt output as desired
- Fill PDF Forms with FDF Data and/or Flatten Forms
- Apply a Background Watermark
- Report PDF on metrics, including metadata and bookmarks
- Update PDF Metadata
- Attach Files to PDF Pages or the PDF Document
- Unpack PDF Attachments
- Burst a PDF document into single pages
- Uncompress and re-compress page streams
- Repair corrupted PDF (where possible)
```



This page intentionally left blank.

Manually Reading .deb File Contents

```
user@midgard:/cases$ cat debian-binary
2.0
user@midgard:/cases$ tar -tvf control.tar.zst
drwxr-xr-x root/root      0 2022-04-22 05:55 .
-rw-r--r-- root/root     33 2022-04-22 05:55 ./conffiles
-rw-r--r-- root/root    439 2022-04-22 05:55 ./control
-rw-r--r-- root/root 7367976 2022-04-22 05:55 ./md5sums
user@midgard:/cases$ tar -tvf data.tar.zst | head
drwxr-xr-x root/root      0 2022-04-22 05:55 .
drwxr-xr-x root/root     0 2022-04-22 05:55 ./etc/
drwxr-xr-x root/root     0 2022-04-22 05:55 ./etc/bash_completion.d/
-rw-r--r-- root/root    773 2022-04-22 05:55 ./etc/bash_completion.d/azure-cli
drwxr-xr-x root/root     0 2022-04-22 05:55 ./opt/
drwxr-xr-x root/root     0 2022-04-22 05:55 ./opt/az/
drwxr-xr-x root/root     0 2022-04-22 05:55 ./opt/az/bin/
-rwxr-xr-x root/root   111 2022-04-22 05:54 ./opt/az/bin/2to3-3.8
drwxr-xr-x root/root     0 2022-04-22 05:54 ./opt/az/bin/__pycache__/
-rw-r--r-- root/root  1580 2022-04-22 05:54 ./opt/az/bin/__pycache__/jp.cpython-38.pyc
```

Once the archive file is extracted with `ar -xov` (or similar), it is possible to manually analyze it, if required.

The `debian-binary` file is a small text file and can easily be read with `cat`. The remaining files are archive formats (normally compressed with gzip or Zstandard), and individual files can be extracted with the `tar` command.

This is an example of using `tar` to extract the MD5 hashes from the control archive (using `grep` to only return the binary hash), which can then be compared with installed applications:

```
user@midgard:/cases$ tar -xOf control.tar.zst ./md5sums | grep "usr/bin"
79df15f896208dd6085f0c28cc4e338d  usr/bin/az
user@midgard:/cases$ md5sum /usr/bin/az
79df15f896208dd6085f0c28cc4e338d  /usr/bin/az
```

The timestamps inside each archive are the last modified time of the file before it was entered into the archive. If you extract individual files, this will be maintained, while the creation, access, and metadata change timestamps will be set to the new file creation time.

Debian Packages and Digital Signatures

Individual packages are not signed.

Repositories manage signing.

- Release file contains checksums.
- Release file signed by repository's key.

Secure APT validates download before installing.

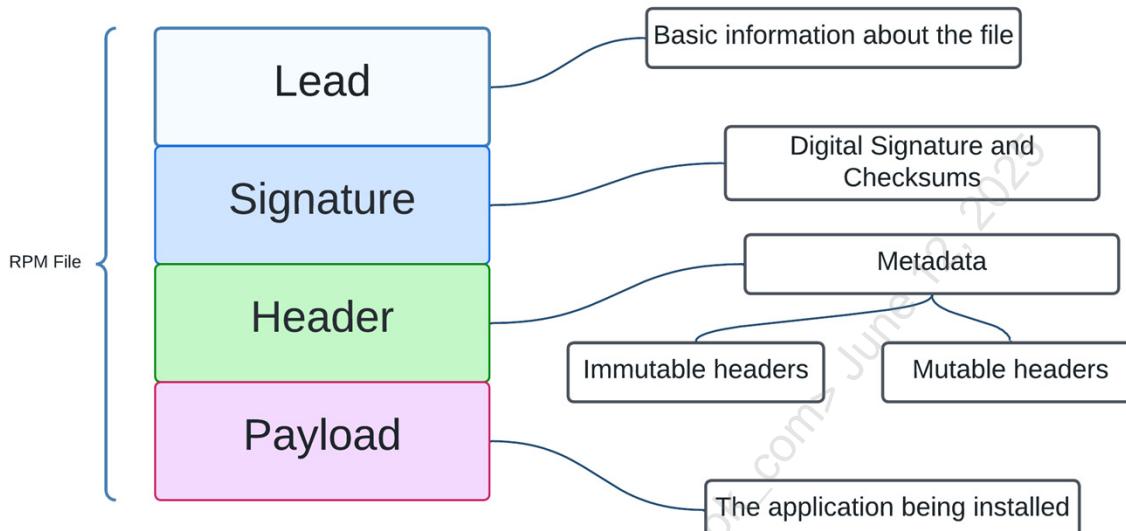
In general, .deb files are not digitally signed on an individual basis. The original reasons for this are largely lost to history, but the main ones used in current discussions concern efficiency—it is resource-intensive to sign and validate each package, and it creates complexity around what is a largely decentralized environment.

Ultimately, for Debian packages, the trust is placed on the repository, which means that individually downloaded packages are harder to validate.

For a Debian package downloaded from a repository (for example, with `apt install packagename`), the sequence is broadly as follows:

- The release file on the repository contains a checksum of the file being downloaded.
- The release file is digitally signed using the repository's key.
- When `apt` (or `apt-get`) installs the package, the digital signature on the release file is checked, and if it is valid, the checksum is trusted.
- If the file checksum matches, it is installed.

Red Hat rpm Package Structures



In the Red Hat family (which also includes SuSE), the .rpm file format also follows a set structure.¹ The structures are as follows:

- **Lead:** A legacy structure for compatibility that includes basic information about the package, such as its type (binary or source) and package name. Though largely obsolete for package management purposes, the lead remains for backward compatibility.
- **Signature:** This section contains information used to verify the integrity and authenticity of the package. It includes digital signatures and checksums of the package content. The RPM system uses this data to ensure that the package has not been tampered with since its creation.
- **Header:** The header is divided into two parts: the immutable and the mutable headers. These contain most of the metadata about the package, including:
 - Description and summary: Brief information about the package.
 - Version and release numbers: For identifying the package version.
 - Dependencies: Lists prerequisites needed by the package for proper installation and operation, as well as any packages that conflict with it.
 - License: The software license under which the package is distributed.
 - Install scripts: Scripts to be executed before or after the package installation or removal.
 - File list: A list of files and directories included in the package.
- **Payload:** The payload contains the actual files and directories to be installed on the system. Typically, gzip, bzip2, or xz compression is used to save space. The payload is extracted into the filesystem upon package installation.

1. <https://for577.com/rpmformat>

Analyzing .rpm Files

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00000000	ED AB EE DB	i<ü....ýawscli
00000010	03 00 00 00 00 FF 61 77 73 63 6C 69 2D 31 2E 31 34 2E 35 30 2D 35 2E 65 6C 38 00 00	-1.14.50-5.16..
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 00 00 00 00
00000040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 05
00000050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060	8E AD E8 01 00 00 00 00 00 00 09 00 00 14 E4	ž.è.....ä ->...ó..
00000070	00 00 00 3E 00 00 00 07 00 00 14 D4 00 00 00 10
00000080	00 00 01 0C 00 00 00 07 00 00 00 00 00 00 02 18
00000090	00 00 01 0D 00 00 00 06 00 00 02 18 00 00 00 01
000000A0	00 00 01 11 00 00 00 06 00 00 02 41 00 00 00 01
000000B0	00 00 03 E8 00 00 00 04 00 00 02 84 00 00 00 01
000000C0	00 00 03 EA 00 00 00 07 00 00 02 88 00 00 02 18
000000D0	00 00 03 EC 00 00 00 07 00 00 04 A0 00 00 00 10
000000E0	00 00 03 EF 00 00 00 04 00 00 04 B0 00 00 00 01
000000F0	00 00 03 F0 00 00 00 07 00 00 04 B4 00 00 10 20
00000100	89 02 15 03 05 00 5D 1A 70 B5 05 B5 55 B3 84 83	¾.....].pu.uU„f
00000110	C6 5D 01 08 71 EA 0F FB 06 ED F0 3E 6B 2C 1E 28	E]..qé.ü.iö>k..(
00000120	BD 62 09 5B 44 49 9A 4C 24 78 01 EE DD 75 9C 6E	¾b.[DIŠL\$x.iÝuœn
00000130	F2 F5 69 5D 3F 17 3B 14 8B FC 8B A7 1F 81 40 4B	ðöi]?.;<ü<\$.@K
00000140	3E 85 2E E5 1D 08 5C 12 02 8E 4D 05 A4 28 A9 23	>....å..žM.¤(©#
00000150	7B 0B A4 FA 97 C6 32 85 1A 95 D2 B8 B5 AA CA 87	(.mú-E2....ò,µ²Ê‡
00000160	EE D6 DE 1F 33 6C 79 A4 7D 29 EA D1 60 77 AB E0	iÖþ.3lyñ])éÑ>w«à
00000170	9D 1B 7B 3C AD 2F CD 64 05 74 80 CC E0 81 B8 7D	..{</Id.teia.,,

The Red Hat Package Manager file uses a binary format that effectively wraps around a compressed tape archive format file (referred to as a cpio file).

Files have four components defined in the RPM format.

- Lead:** This is a 96-byte section that includes the file metadata. The first four bytes are the RPM file signature of ed ab ee db. The next two bytes denote the major and minor file version numbers. Bytes 6 to 7 form a 16-bit number, which indicates the package type. There are only two options currently available: 0 for binary files and 1 for source files. This is followed by two bytes for the architecture, as returned by the `file` command. Bytes 10 to 75 are used to store the package name, version, and release number, with null (0) padding as required. Bytes 76 to 77 indicate the OS (currently, always a 1 for Linux). Bytes 78 to 79 indicate signature type, with a 5 referring to "header-style" signatures. The remaining 16 bytes are currently unused.
- Signature:** Version 3.0 format signatures (denoted by a type 5 in the lead) use the same structure as the header. This includes a file signature of 8e ad e8 followed by a single byte to show the header version. This region holds digital signatures for the included files, which can be in MD5 or PGP format. This area does not have a fixed length.
- Header:** Next comes the header. The header also starts with 8e ad e8 and includes information about the package, including name, version number, included file list, installation/dependency data, and other information as needed to manage the file install.

An example of what this data can look like is shown here:

000019B0	00 04 DB D4 00 00 00 01 43 00 61 77 73 63 6C 69 ..ÛÛ....C.awscli
000019C0	00 31 2E 31 34 2E 35 30 00 35 2E 65 6C 38 00 55 .1.14.50.5.e18.U
000019D0	6E 69 76 65 72 73 61 6C 20 43 6F 6D 6D 61 6E 64 niversal Command
000019E0	20 4C 69 6E 65 20 45 6E 76 69 72 6F 6E 6D 65 6E Line Environmen
000019F0	74 20 66 6F 72 20 41 57 53 00 54 68 69 73 20 70 t for AWS.This p
00001A00	61 63 6B 61 67 65 20 70 72 6F 76 69 64 65 73 20 ackage provides
00001A10	61 20 75 6E 69 66 69 65 64 0A 63 6F 6D 6D 61 6E a unified.command
00001A20	64 20 6C 69 6E 65 20 69 6E 74 65 72 66 61 63 65 d line interface
00001A30	20 74 6F 20 41 6D 61 7A 6F 6E 20 57 65 62 20 53 to Amazon Web S
00001A40	65 72 76 69 63 65 73 2E 00 00 00 00 5C DD E0 37 ervices....\Ýà7
00001A50	70 70 63 36 34 6C 65 2D 30 32 2E 6D 62 6F 78 2E ppc64le-02.mbox.
00001A60	63 65 6E 74 6F 73 2E 6F 72 67 00 00 00 38 9B 68 centos.org...8>h
00001A70	43 65 6E 74 4F 53 00 43 65 6E 74 4F 53 00 41 53 CentOS.CentOS.AS
00001A80	4C 20 32 2E 30 20 61 6E 64 20 4D 49 54 00 43 65 L 2.0 and MIT.Ce
00001A90	6E 74 4F 53 20 42 75 69 6C 64 73 79 73 20 3C 62 ntOS Buildsys <b
00001AA0	75 67 73 40 63 65 6E 74 6F 73 2E 6F 72 67 3E 00 ugs@centos.org>.
00001AB0	55 6E 73 70 65 63 69 66 69 65 64 00 68 74 74 70 Unspecified.http
00001AC0	3A 2F 2F 61 77 73 2E 61 6D 61 7A 6F 6E 2E 63 6F ://aws.amazon.co
00001AD0	6D 2F 63 6C 69 00 6C 69 6E 75 78 00 6E 6F 61 72 m/cli.linux.noar
00001AE0	63 68 00 00 00 00 00 00 00 00 CC 00 00 03 3E ch.....i....>

There is a concept of "mutable" and "immutable" headers in RPM, although since Version 4, the package format has fewer mutable header regions. The primary difference is that an immutable header is metadata that shouldn't change once the package has been built. This includes information such as the list of files it includes, checksums, etc.

Mutable headers can include last access times, installation counts, and other information that is subject to change and relevant to the package management system's database.

4. Payload: The final section is the payload. This is a gzipped cpio archive with a CRC checksum.

The Header Structure

The header structure is used for both the signature and header areas. This structure consists of three parts: an 8-byte header record, including the file signature; an array of one or more index records; and then 0 or more bytes of data associated with the index records.

Manual File Extraction

```
user@midgard:/cases$ rpm2cpio aws-cli.rpm > aws-cli.rpm.cpio
user@midgard:/cases$ cpio -i -tv < aws-cli.rpm.cpio | grep README
-rw-r--r-- 1 root      root      17754 Mar  1 2018 ./usr/share/doc/awscli/README.rst
-rw-r--r-- 1 root      root      12014 May 16 2019 ./usr/share/doc/awscli/colorama_README.rst
8026 blocks
user@midgard:/cases$ cpio -i --to-stdout ./usr/share/doc/awscli/README.rst < aws-cli.rpm.cpio | head
=====
aws-cli
=====

.. image:: https://travis-ci.org/aws/aws-cli.svg?branch=develop
:target: https://travis-ci.org/aws/aws-cli
:alt: Build Status

.. image:: https://badges.gitter.im/aws/aws-cli.svg
:target: https://gitter.im/aws/aws-cli
```

RPM files can be analyzed using the `rpm2cpio` and `cpio` command-line tools. These tools can be installed on Debian systems (such as Ubuntu) and RHEL-based distros. This can be an effective way for investigators to analyze files without risking accidental execution.

This is best carried out in a sequence.

1. Convert the `.rpm` file to a `.cpio` file with `rpm2cpio`. The syntax for this is:
`rpm2cpio RPMFILENAME > CPIOFILENAME`.
2. Use `cpio` to determine what files are present in the archive and identify any of interest. In the example on the slide, the analyst is using `grep` to search the output for `README` files. The syntax for this is:
`cpio -i -tv < CPIOFILENAME` (optionally `| grep SEARCHSTRING`).
3. Finally, use `cpio` to extract the file(s) of interest. In the example on the slide, the analyst has decided to pipe the output to the screen and limit it to the first ten lines with the `head` command. The syntax for this is:
`cpio -i --to-stdout FILEOFINTEREST < CPIOFILENAME`.

Files can be piped to `stdout` (normally displayed in the terminal) or saved to disk, depending on the investigator's requirements.

Hunting Through Package Management Data: Guidelines

Check installed applications for unusual entries

- Random or short (1-2 character) application names
- Installation during the intrusion
- Unusual repository use

Check packages for inconsistent data

- Timestamps
- Digital signatures
- Pre/post-installation scripts

Review individual elements

- Extract files and review



When it comes to threat hunting through package management tools, or validating them during the scoping phase of IR, you have some guidelines to consider:

- Review all the installed applications on the device. This is going to be noisy—often there will be hundreds of files installed. Look for applications with random character names, often a characteristic of malware, or applications with unusually short (or long) names. The idea is that attackers will randomize names to make threat hunting difficult, and short names are often used by attackers in a rush. Also, if you have any useful threat intelligence, look for specific application names that relate to the known indicators.
- Check for the repositories being used. It is possible that an attacker with sufficient privileges could add a malicious/unwanted application repository that allows them to install their own tools.
- Check the individual packages and ensure they contain consistent information. Focus on items such as the internal file timestamps. With Red Hat (RPM) files, the signatures should be intact and valid, but where possible, you should check whether the signature is an expected one.
- Review any pre- or post-installation scripts to check for unusual behavior.
- Finally, you might have to extract the installer files and review them individually.



Lab 1.3: Package Management Review

Estimated time: 30 minutes

This page intentionally left blank.

FOR577.I

Linux Incident Response and Threat Hunting



Threat Intelligence and Host-Based Threat Hunting

SANS | DFIR

FOR577 | Linux Incident Response and Threat Hunting 110

This page intentionally left blank.



Understanding the Adversary

Threat Intelligence

This page intentionally left blank.

What Is Cyber Threat Intelligence (CTI)

"Threat intelligence is data that is collected, processed, and analyzed to **understand a threat actor's motives, targets, and attack behaviors.** Threat intelligence enables us to make faster, more informed, data-backed security decisions and change their behavior from reactive to proactive in the fight against threat actors."

[Source: CrowdStrike; emphasis changed from original]



There are lots of definitions. The one on the slide is from CrowdStrike,¹ and although there are many others, this covers the main points well. For our purposes, we will focus on the bits in bold. We use CTI to try and understand what attackers are trying to do, what they will attack, and how they will attack it.

Used properly, threat intelligence provides us with insight into attackers' tactics, techniques, and procedures. In turn, this allows us to proactively block techniques, detect attacks earlier in the cycle, and hunt our environments for malicious activity.

1. What Is Cyber Threat Intelligence? [Beginner's Guide]: <https://for577.com/what-cti>

CTI Sources

External

- Commercial CTI providers
- Intelligence sharing/special interest groups
- Public feeds

Internal

- Inbound filtering (mail/proxy)
- Investigations

For ease of discussion, we've classed CTI sources into two groups based on commonalities.

External CTI Sources

This covers all the different ways you can get intelligence from sources outside your organization. This covers a range of sources, and some examples include:

- Monitoring Twitter for announcements and discoveries
- Subscribing to free CTI feeds, such as AlienVault's OTX feed¹ or Mandiant Threat Intelligence free version²
- Feeds provided by MSSP or security vendors, such as IBM X-Force, Rapid7's Integrated Threat Feed, and so on
- Membership of security forum, special interest group, information sharing group, which can be based on business sector, geographical region, and so on
 - The most common are the Information Sharing and Analysis Center (ISAC) groups.³
- A paid-for threat intelligence feed, often from a specialist CTI organization

The main advantage of externally sourced threat intelligence is that the often-challenging technical analysis is carried out by someone else. This reduces the need to have in-house expertise in a range of technologies.

The biggest disadvantage is that it is often generic intelligence produced for a range of consumers rather than being specific to your organization or needs.

Internal CTI Sources

This covers threat intelligence you collect inside your organization using your own staff and analyzing data you have collected yourself.

Examples include:

- Analyzing inbound emails to identify phishing URLs or malware families
- Analyzing network sensors (proxy, DNS) to identify C2 traffic or attempts to transfer malicious files
- Intelligence driven from incident response investigations and indicators of compromise/attack (IOC/IOA)

The biggest advantage to this is that the threat intelligence you generate here is very specific and targeted, often providing you with information you can immediately act on. Generating intelligence related to attacks as they are happening means you can immediately use it.

The main disadvantage is that it can be resource intensive, both in the need to have good monitoring and response tools and also the need to have sufficiently skilled staff available. This can include network forensic specialists, reverse engineers, and endpoint incident responders.

1. AlienVault OTX <https://for577.com/otx>
2. Mandiant Threat Intelligence Free <https://for577.com/man-free>
3. ISAC summary by ENISA <https://for577.com/enisa-isac>

CTI Terms: Indicators, Tactics, Techniques, and Procedures



Indicators of Compromise (IOCs)

Signs that a system has been compromised



Indicators of Attack (IOAs)

Signs that a specific attack is underway



Tactics, Techniques, and Procedures (TTPs)

How the attacker behaves
What tools the attacker uses

To help ensure we use a common vocabulary during this course, some terms need to be defined.

Indicators of Compromise (IOC)

This term is used to mean the signs or forensic artifacts that show a system has been compromised by an attacker. Examples include specific scheduled tasks being created, the presence of known malware, or specific strings stored in memory (called "mutexes" or "mutants").

Indicators of Attack (IOA)

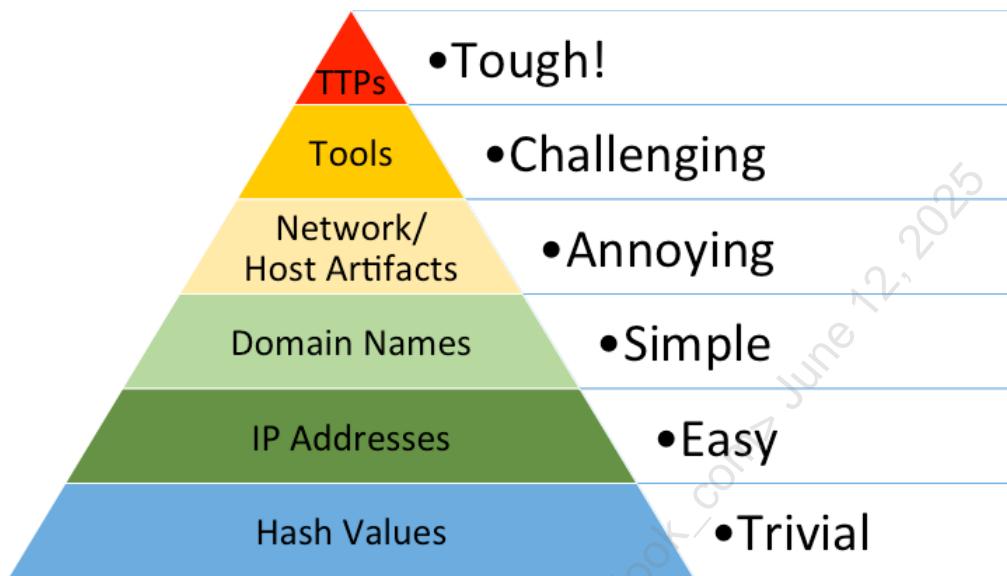
Very similar and often used interchangeably with IOC, IOA is used to mean the forensic artifacts that show a specific attack is underway. This should differentiate from a general attack; for example, password spraying would not normally be considered an indicator of an attack. Examples of good IOA include activity linked to specific domains or IP addresses or probes against specific services.

Tactics, Techniques, and Procedures (TTPs)

This is a breakdown of the attacker's behaviors, tool use, and methodologies to help defenders gain insights and begin to predict where the adversary might be. Examples of TTPs include things like identifying how an attacker names their tools, the techniques they use for privilege escalation/lateral movement, and so forth. This can be very challenging for an attacker to change and is often the best type of threat intelligence to collect.

Note on language: Lots of organizations avoid using the term *indicator of compromise*, as it may carry contractual or regulatory reporting issues; instead, they describe everything as an IOA.

CTI Indicators: The Pyramid of Pain



With all indicators or TTPs, it is important to understand how valuable they are, and this is based on the difficulty the attackers will have in changing them once they are known to the defenders.

David Bianco summarized this relationship in the now-famous *Pyramid of Pain*.¹ This conceptual model shows the things we often discover easily and quickly (hash values, IP addresses, etc.) and are also very straightforward for the attackers to change. However, once we move up the pyramid, we encounter things that require significant effort to change.

From an operational incident response perspective, this often means the lower-level indicators are useful during an ongoing incident before the attackers are aware a response has taken place or for threat hunting to see if a previous attack went unnoticed.

Hash Values

Hashes, or checksums, are generated by running a known algorithm against the file to create a unique hexadecimal number that identifies the file. This is then used to identify duplicates of the file (same hash) or if the file has been changed (the hash changes).

With IOCs, the hash is often a representation of malicious files and can be used to search resources such as VirusTotal to find matches.

Unfortunately for defenders, the hash is trivial to change, as changing anything inside the file will cause the file's hash to change. An attacker can add nulls or junk code or even modify internal comments, and every change will create a new hash.

As a result, while this is often an easy-to-discover threat intelligence element, it has a very limited lifespan.

IP Addresses

During an attack, we often see elements of remote access, command and control communications, downloading malware from staging sites, and data exfiltration. All of this is driven by IP addresses, making them a fundamental indicator for most incidents.

Sometimes, attackers hard-code an IP address (C2 or downloader) into their tooling. This makes it slightly more challenging for them to adapt—they need to reconfigure the tool and then deploy it into the target organization.

This remains low on the pyramid, though, because it is generally easy for an attacker to change IP addresses if one is believed to be compromised. In addition, lots of attacks come from platforms with inherently changeable IPs, such as Tor or other anonymizing networks, or from content delivery networks/cloud providers.

Domain Names

Related to IP addresses are domain names. This is slightly harder for an attacker to change because some registration has to take place. There are still enough domain registrar organizations that will allow quick registration with minimal questions asked or standards required.

Frequently, attackers will compromise an innocent victim and use their domain to stage attacks; this makes it a bit harder for them to quickly change domain names.

Generally, domain names remain static during an attack but change between attacks. However, sub-domain names can and often do change frequently, with some attacks cycling between thousands in short periods.

Network/Host Artifacts

Moving up the pyramid, we come to some artifacts that are going to require some effort for the attacker to change.

This artifact group covers elements that will require the attackers to reconfigure their tooling to change. Examples include elements like the user-agent string in the attackers' tools or mutexes stored in memory while malicious code runs.

While it may seem that this is trivial to change, as it often impacts a multitude of tooling, it does start to present hurdles for the attacker.

Tools

This level of the pyramid is where incident responders can actually make life hard for the attackers. Identifying the tools of choice and ideally being able to block them or prevent the attackers' access can present a major hurdle.

Most of us—attackers and defenders—get used to a specific way of doing things and using specific tools for tasks. We can definitely use other tools, but it slows us down, increases our error rate, and starts to force behavior changes. Sometimes, it forces attackers to create entirely new tools, which can eliminate some lower-skilled attack groups.

TPPs

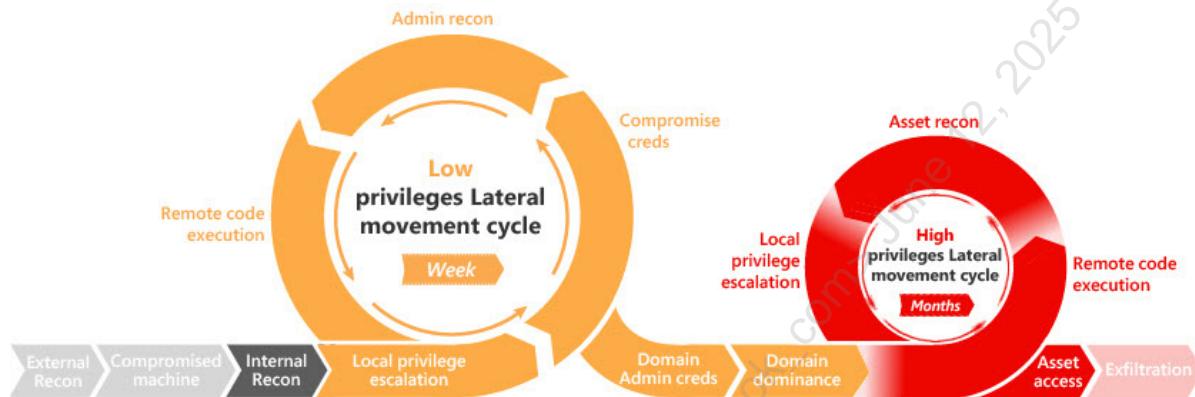
This is the apex of the pyramid. Here, we are looking at attacker behaviors, not specific tool choices, and so on. This is harder for responders to identify well but very challenging for attackers to change.

Examples of this category include things like using pass-the-hash attacks in Windows or writing SSH keys to the Authorized Keys file in Linux. Being able to detect the attacker's tactics is a significant step for the responders and generally forces the attacker to learn new techniques or give up.

1. Pyramid of Pain <https://for577.com/pyramid>

CTI:Attack Life Cycles

Threat intelligence allows us to map activity to the attacker life cycle.

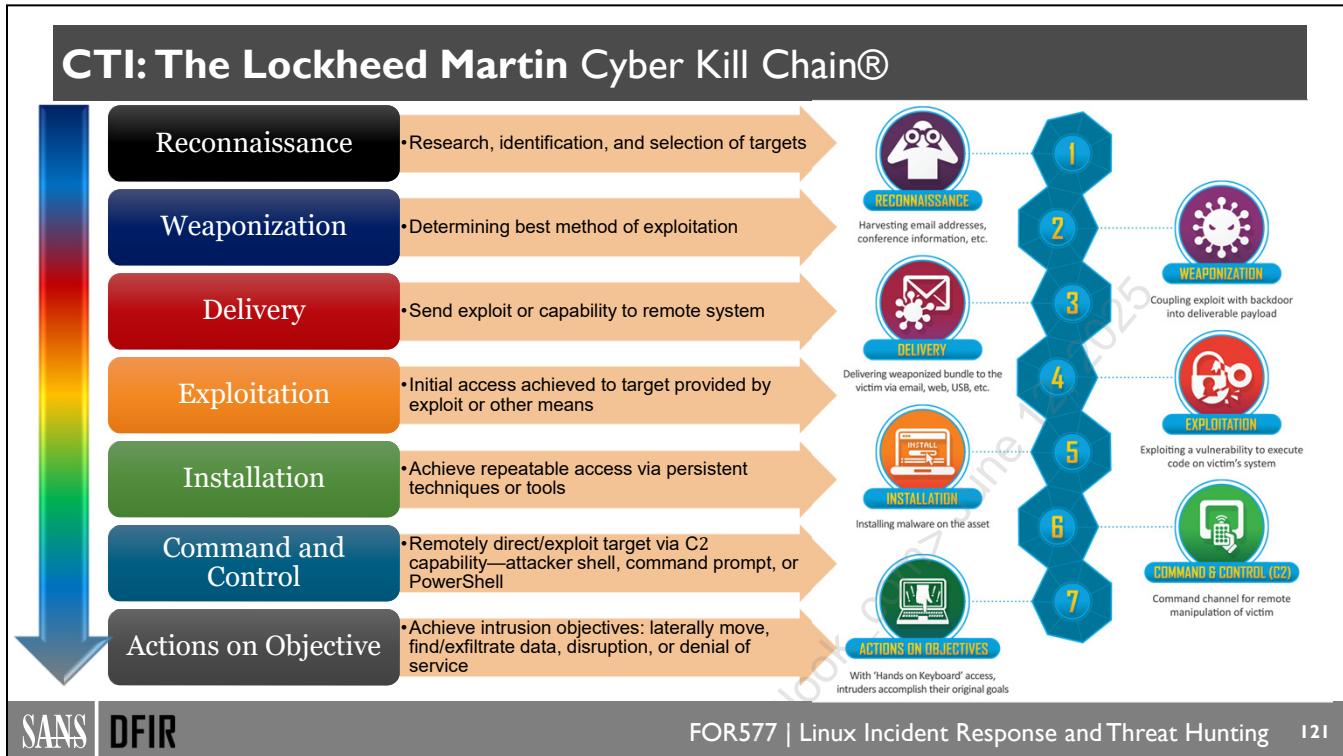


Threat intelligence attempts to map attacker techniques, tactics, and procedures to the attack life cycle. Nearly all attacks progress through a series of common steps to accomplish adversary objectives. Understanding these steps allows defenders to craft opportunities to discover attacks while in progress. The model on this slide is sourced from the documentation on Microsoft Advanced Threat Analytics, and it does a commendable job of demonstrating the cyclical nature of attacks as attackers perform network discovery and elevate privileges to achieve greater access.

- **Initial Compromise:** Most initial compromises are not persistent, and the level of access achieved by an adversary at this stage is generally very fragile. If a response team can eliminate an adversary before establishing a foothold on the network, then the survivability of the adversary drops to nearly zero. Unfortunately, this can be extremely hard to accomplish because initial reconnaissance and exploit delivery leave few discernable artifacts that rise above the noise of constant attacks against the network.
- **Low Privileges Lateral Movement Cycle:** During this phase, an attacker must maintain persistence within the environment and expand their foothold, gaining access to additional systems. The placement of persistent backdoors is common. Large amounts of lateral movement paired with credential dumping are seen during this phase, as high-level credentials are required to achieve most attacker goals.
- **High Privileges Lateral Movement Cycle:** Once high-level credentials are achieved, the attack shifts from mass credential collection to asset collection as the adversary extends their influence within the environment and prepares to accomplish their ultimate objectives. Since high-level credentials are (sadly) often discovered early in the attack, the phase can comprise the longest length of time of the intrusion. Tracking unusual credential usage, lateral movement, and abnormal system access are important capabilities in finding attacker activity in this phase.

- **Asset Access and Data Exfiltration:** Adversaries leave many footprints as they search for and collect data of interest. The process of finding specific data on a remote network is challenging. You might have a hard time doing it on your own systems; imagine what it would be like to try and find an important file out of millions of files across thousands of hosts! Once assets and data are identified, an attacker must find a way to exfiltrate it. This is often easier said than done because moving a large quantity of data outside the network is likely to be caught by network monitoring. It is common for attackers to find and utilize a staging system in order to accomplish this goal. An alternative to data exfiltration is the setup of destructive attacks like ransomware. Similar to data exfiltration, the most sophisticated attackers can spend weeks or even months learning about a network before deploying their final payloads.

The flip side of this life cycle is that the incident response team is charged with identifying intrusion activity. The best teams employ intelligence-driven incident response, using the identification/scoping and containment/intelligence stages of the incident response cycle to understand an adversary both during and after an incident. By collecting intelligence at each phase of the attack life cycle, defenders can more easily counter adversary activity and employ mechanisms, both automated and manual, to find them in the future. The latter point is important as it implies the same adversary will likely be seen again, and the knowledge gained during this intrusion can allow faster mitigation of future attacks.



The Cyber Kill Chain® is an important model used in threat intelligence. It helps categorize the sequence of actions occurring in most attacks and provides a framework for organizing detection indicators. It is best described by one of its authors, Mike Cloppert:

Security Intelligence: Attacking the Cyber Kill Chain

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

On Indicators

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

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

Atomic indicators are pieces of data that are indicators of adversary activity on their own. Examples include IP addresses, email addresses, a static string in a covert command and control (C2) channel, or fully qualified domain names (FQDNs). Atomic indicators can be problematic because they might or might not exclusively represent activity by an adversary. For instance, an IP address from where an attack is launched could very likely be an otherwise legitimate site. Atomic indicators often need vetting through analysis of available historical data to determine whether they exclusively represent hostile intent.

Computed indicators are those that are, well, computed. The most common among these indicators are hashes of malicious files, but they can also include specific data in decoded custom C2 protocols. Your more complicated IDS signatures might fall into this category.

Behavioral indicators are those that combine other indicators (including other behaviors) to form a profile. Here is an example: Bad Guy 1 likes to use IP addresses in West Hackistan to relay email through East Hackistan and target our sales folks with trojaned MS Word documents that discuss our upcoming benefits enrollment, which drops backdoors that communicate to A.B.C.D. Here, we see a combination of computed indicators (geolocation of IP addresses, MS Word attachments determined by a magic number, and base64 encoded in email attachments), behaviors (targets sales force), and atomic indicators (A.B.C.D C2). Already, you can probably see where we're going with intelligence-driven response. What if we can detect, or at least investigate, behavior that matches what I described previously?

One likes to think of indicators as conceptually straightforward, but the truth is that proper classification and storage have been elusive. I'll save the intricacies of indicator difficulties for a later discussion.

Adversary Behavior

The behavioral aspect of indicators deserves its own section. Indeed, most of what we discuss in this installment centers on understanding *behavior*. The best way to behaviorally describe an adversary is by how they do their job. After all, this is the only discoverable part for an organization that is strictly CND (some of our friends in the US government likely have better ways of understanding adversaries). That job is compromising data, and therefore, we describe our attacker in terms of the anatomy of their attacks.

Ideally, if we could attach a human being to each and every observed activity on our network and hosts, we could easily identify our attackers and respond appropriately every time. At this point in history, that sort of capability passes beyond a pipe dream into ludicrous. However mad this goal is, it provides a target for our analysis: We need to push our detection closer to the adversary. If all we know is the forged email address an adversary tends to use in delivering hostile emails, assuming this is uniquely linked to malicious behavior, we have a mutable and temporal indicator upon which to detect. Sure, we can easily discover when it's used in the future, and we are obliged to do so as part of our due diligence. The problem is that this can be changed at any time on a whim. If, however, the adversary has found an open mail relay that no one else uses, then we have found an indicator closer to the adversary. It's much more difficult (though, in the scheme of things, still somewhat easy) to find a new open mail relay to use than it is to change the forged sending address. Thus, we have pushed our detection closer to the adversary. Atomic, computed, and behavioral indicators can describe more or less mutable/temporal indicators in a hierarchy. We, as analysts, seek the most static of all indicators at the top of this list but often must settle for indicators further from the adversary until those key elements reveal themselves.

That this analysis begins with the adversary and then dovetails into defense makes it very much a security intelligence technique, as we've defined the term. Following a sophisticated actor over time is analogous to watching someone's shadow. Many factors influence what you see, such as the time of day, the angle of the sun, etc. After you account for these variables, you begin to notice nuances in how the person moves, observations that make the shadow distinct from others. Eventually, you know so much about how the person moves that you can pick him out of a crowd of shadows. However, you never know for sure if you're looking at the same person. At that point, for our purposes, it doesn't matter. If it looks like a duck and sounds like a duck, it hacks like a duck. Whether the same person (or even group) is truly at the other end of behavior every time is immaterial if the profile you build facilitates predicting future activity and detecting it.

Attack Progression, or the Kill Chain

We have found that the phases of an attack can be described by six sequential stages. Once again, loosely borrowing vernacular, the phases of an operation can be described as a "kill chain." This is a linear flow—some phases might occur in parallel, and the order of earlier phases can be interchanged—but rather how far along an adversary has progressed in their attack, the corresponding damage, and the investigation that must be performed.

Recon

The reconnaissance phase is straightforward. However, in security intelligence, often, this is manifested not in port scans, system enumeration, or the like. It is the data equivalent: browsing websites, pulling down PDFs, and learning the internal structure of the target organization. A few years ago, I never would've believed that people went to this level of effort to target an organization, but after witnessing it happen, I can say with confidence that it does. The problem with activity in this phase is that it is often indistinguishable from normal activity. There are precious few cases where one can collect information here and find associated behavior in the delivery phase matching an adversary's behavioral profile with high confidence and a low false positive rate. These cases are truly gems; when they can be identified, they link what are often two normal-looking events in a way that greatly enhances detection. The weaponization phase might or might not happen after reconnaissance; it is placed here merely for convenience. This is the one phase the victim doesn't see happen but can very much detect. Weaponization is the act of placing a malicious payload into a delivery vehicle. It's the difference between how a Russian warhead is wired to the detonator versus how a US warhead is wired in. For us, it is the technique used to obfuscate shellcode, how an executable is packed into a trojaned document, and so forth. Detection of this is not always possible, nor is it always predictable, but when it can be done, it is a highly effective technique. Only by reverse engineering of delivered payloads is an understanding of an adversary's weaponization achieved. This is distinctly separate and often persistent across the subsequent stages.

Delivery

Delivery is rather straightforward. Whether it is an HTTP request containing SQL injection code or an email with a hyperlink to a compromised website, this is the critical phase where the payload is delivered to its target. I heard a term just the other day that I really like: "warheads on foreheads" (courtesy of the US Army).

Exploitation

The compromise phase will possibly have elements of a software vulnerability, a human vulnerability known as "social engineering," or a hardware vulnerability. Although the latter are quite rare by comparison, I include hardware vulnerabilities for the sake of completeness. The compromise of the target might itself be multiphase or more straightforward. As a result, we sometimes have the tendency to pull apart this phase into separate sub-phases or peel out *compromise* and *exploit* it as wholly separate. For simplicity's sake, we'll keep this as a single phase. A single-phase exploit results in the compromised host behaving according to the attacker's wishes directly as a result of the successful execution of the delivered payload, for example, if an attacker coaxes a user into running an EXE attachment to an email, which contained the desired backdoor code. A multiphase exploit typically will involve the delivery of shellcode whose sole function is to pull down and execute more capable code upon execution. Shellcode often needs to be portable for a variety of reasons, necessitating such an approach. We have seen other cases where, possibly through sheer laziness, adversaries end up delivering exploits whose downloaders download other downloaders before finally installing the desired code. As you can imagine, the more phases involved, the lower an adversary's probability of success.

This is the pivotal phase of the attack. If this phase is completed successfully, what we as security analysts have classically called "incident response" is initiated: Code is present on a machine that should not be there. However, as will be discussed later, the notion of incident response is so different in intelligence-driven response (and the classic model so inapplicable) that we have started to move away from using the term altogether. The better term for security intelligence is *compromise response* because it removes ambiguity from the term *incident*.

C2: Maintain Presence

The command and control phase of the attack represents the period after which adversaries leverage the exploit of a system. A compromise does not necessarily mean C2, just as C2 doesn't necessarily mean exfiltration. In fact, we will discuss how this can be exploited in CND but recognize that successful communications back to the adversary *often* must be made before any potential for impact to data can be realized. This can be prevented intentionally by identifying C2 in unsuccessful past attacks by the same adversary, resulting in network mitigations or fortuitously when adversaries drop malware that is somehow incompatible with your network infrastructure, to give but two examples.

In addition to the phone call going through, someone has to be present at the other end to receive it. Your adversaries take time off, too, but not all of them. In fact, a few groups have been observed to be so responsive that it suggests a mature organization with shifts and procedures behind the attack more refined than that of many incident response organizations.

Actions on Objectives

The actions on objectives phase is conceptually very simple: For most advanced persistent threats (APTs), this is when the data, which has been the ultimate target all along, is taken. Previously, I mentioned that gathering information about the environment of the compromised machine doesn't fall into the exfiltration phase. The reason for this is that such data is being gathered to serve but one purpose, either immediately or longer term, to facilitate collection and theft of the target information: the source code for the new O/S, the new widget that cost billions to develop, and access to the credit cards or personally identifiable information (PII). Adversary objectives could also fall into the category of denial of service, data destruction, or more. We will also lump lateral movement with compromised credentials, filesystem enumeration, and additional tool dropping by adversaries broadly into this phase of the attack. However, an argument can be made that situational awareness of the compromised environment is, technically, exfiltration.

Security Intelligence Using the Kill Chain Successfully

(This is included with permission from Mike Cloppert as originally published on the SANS Computer Forensics Blog.)

The *persistence* in APT intrusions is manifested in two ways: maintaining a presence on your network and repeatedly attempting to gain entry to areas where presence is not established. The repeatability of these activities inevitably involves consistent attributes because resource constraints typically prevent adversaries from acting differently every time they set foot in your environment. With a way to model intrusions and align these common attributes, network defenders can take advantage of persistence to profile their adversaries, informing strategic response, analysis efforts, and resource investment.

A single intrusion, as we have already discussed, can be modeled as seven phases. Within each of these phases of an intrusion is a highly dimensional set of indicators—computer scientists would call them "attributes"—that together uniquely define that intrusion. For example, a C2 callback domain is an indicator attribute; talktome.bad.com is the corresponding value of the indicator. The targeting used (reconnaissance), how the malicious payload is obscured (weaponization), the path the payload takes (delivery), the way the payload is invoked (exploit), where the backdoor is hidden on the system (installation), the protocol used to call back to the adversary (C2), and habits of the adversary once control is established (actions on intent) are all categorical examples of these indicators. It is up to the analyst to discover the significant or uniquely identifying indicators in an intrusion.

In some cases, there are common indicators—for example, the last-hop email relay used to deliver a message will be significant in most like intrusions, excluding webmail. In others, the attributes can be unique and surprising—a piece of metadata, a string in the binary of a backdoor, and a predictably malformed HTTP request to check for connectivity.

Be aware that adversaries shift tactics over time. A campaign is not static, nor are the key indicators or corresponding values. We've seen adversaries use the same delivery and C2 infrastructure for years, whereas others will shift from consistent infrastructure in the delivery and C2 phases to highly variable infrastructure in the delivery phase but with consistent targeting and weaponization techniques. Some adversaries will have consistent key indicators, such as tool artifacts in the delivery and weaponization phases, but the specific indicator values might change over time. Without constant and complete analysis of sophisticated intrusions, knowledge of campaigns becomes stale and ineffective at predicting future intrusions.

Gathering Intel through Kill Chain Completion

To have the data set necessary to link intrusions and identify key indicators, analysts must understand all phases of every sophisticated intrusion. Initial detection of an intrusion might occur at any point across the kill chain. Even if the attack is unsuccessful, detection is just the first step.

Classic incident response methodology assumes a system compromise. In the situation where a detection happens after the installation and/or execution of malicious code, adversaries have successfully executed many steps in their intrusion. As the intrusion progresses forward in the kill chain, so the corresponding analysis progresses backward. Analysts must reconstruct every prior stage, necessitating not only the proper tools and infrastructure to do so but also deep network and host forensic skills. Less mature response teams will often get stuck in the delivery to installation phases. Without knowledge of what happened earlier in an intrusion, network defenders will be unable to define campaigns at these earlier phases, and response to intrusions will continue to happen post-compromise because this is where the detections and mitigations are. When walls are hit in analysis that prevents reconstruction of the entire chain, these barriers represent areas for improvement in instrumentation or analytical techniques. Where tools do not already exist for accurate and timely reconstruction, development opportunities exist. Here is but one area where having developers on staff to support incident responders is critical to the success of the organization.

As response organizations mature and can more fully build profiles of intrusion campaigns against them, they become more successful at detection prior to compromise. However, just as a post-compromise response involves a significant amount of analysis, the unsuccessful intrusion attempts matching APT campaign characteristics also require investigation. The phases executed successfully by the adversary must still be reconstructed, and the phases that were not must be synthesized to the best ability of the responders. This aspect is critical to identifying any TTP change that may have resulted from a successful compromise. Perhaps the most attention-grabbing example is the identification of zero-day exploits used by an APT actor at the delivery phase before the exploit is invoked.

Synthesis clearly demonstrates the criticality of malware reverse engineering skills. It is likely that the backdoor that would have been dropped, even if it is of a known family, using a known C2 protocol, also contains new indicators further defining the infrastructure at the disposal of adversaries. Examples include indicators such as C2 callback IP addresses and fully qualified domain names. Perhaps minor changes in the malicious code would produce new unique hashes, or a minor version difference results in a slightly different installation filename that could be unique. Although antivirus is typically a bad example of detection in the context of APT intrusions, there are times when it can be of value for older variants of code. For instance, how many reading this analyze emails that are detected by their perimeter antivirus system? If the detection is for a particular backdoor uniquely linked to an APT campaign, the email could contain valuable indicators about the adversary's delivery or C2 infrastructure that might be reused later in an intrusion that your antivirus system does not detect.

Detecting campaigns enable resilient detection and prevention mechanisms across an intrusion and engage CND responders earlier in the kill chain, reducing the number of successful intrusions. It should be obvious, but it bears repeating that a lack of specific indicators from a single intrusion prevents the identification of key indicators from sequential intrusions. A lack of key indicators results in an inability to define adversaries, and an inability to define adversaries leaves network defenders responding post-compromise to every intrusion. In short, the inability to reconstruct intrusions should be considered an organizational failure of CND, and intelligence-based detections prior to system compromise a success. Defining campaigns, as demonstrated here, is one effective way to facilitate success.

Reference:

- Sourced with permission from Mike Cloppert and originally published on the SANS Computer Forensics Blog at <https://for577.com/cyberkillchain> and officially branded by Lockheed Martin at <https://for577.com/kh03e>

Host-Based Threat Hunting

Using CTI to find the hidden attacker



This page intentionally left blank.

What Is Threat Hunting?



Proactively looking for adversaries



Reduces attacker dwell time



Improves monitoring

There are lots of definitions of threat hunting, and it can vary between organizations. We will use the term to mean the activities an organization carries out to proactively look for possible attacks. This is significantly different from lots of our defensive security controls, which tend to be more passive, firing alerts if a predefined threshold is met.

Threat hunting is a way organizations can look to see if controls have failed (or have been bypassed) by an attacker, making them otherwise undetectable. Good threat hunting is the best way to reduce attacker dwell time, identify control weaknesses, and, crucially, identify monitoring gaps. If it isn't possible to answer the threat hunt, that's a good sign that there is a monitoring gap.

Types of Threat Hunt



Hypothesis-Led

Intelligence-Led

At a very basic level, there are two types of threat hunt. You can modify this to suit your organizational needs, and it is a good idea to run both types.

- **Hypothesis-Led Hunting.** With this approach, you start by forming a hypothesis on something an attacker might do. This combines well with Red Team/Adversary Emulation attacks, and the MITRE ATT&CK framework is very useful.
- **Intelligence-Led Hunting.** This approach takes targeted information about a known threat to your organization and then hunts across the data to find any evidence it may have happened.

It is important to note that both hunts are equally important. Avoid thinking of this as one is better than the other.

Threat Hunt Requirements



Data



Analysis



Time

Generally speaking, three elements are required for an effective threat hunt.

Data

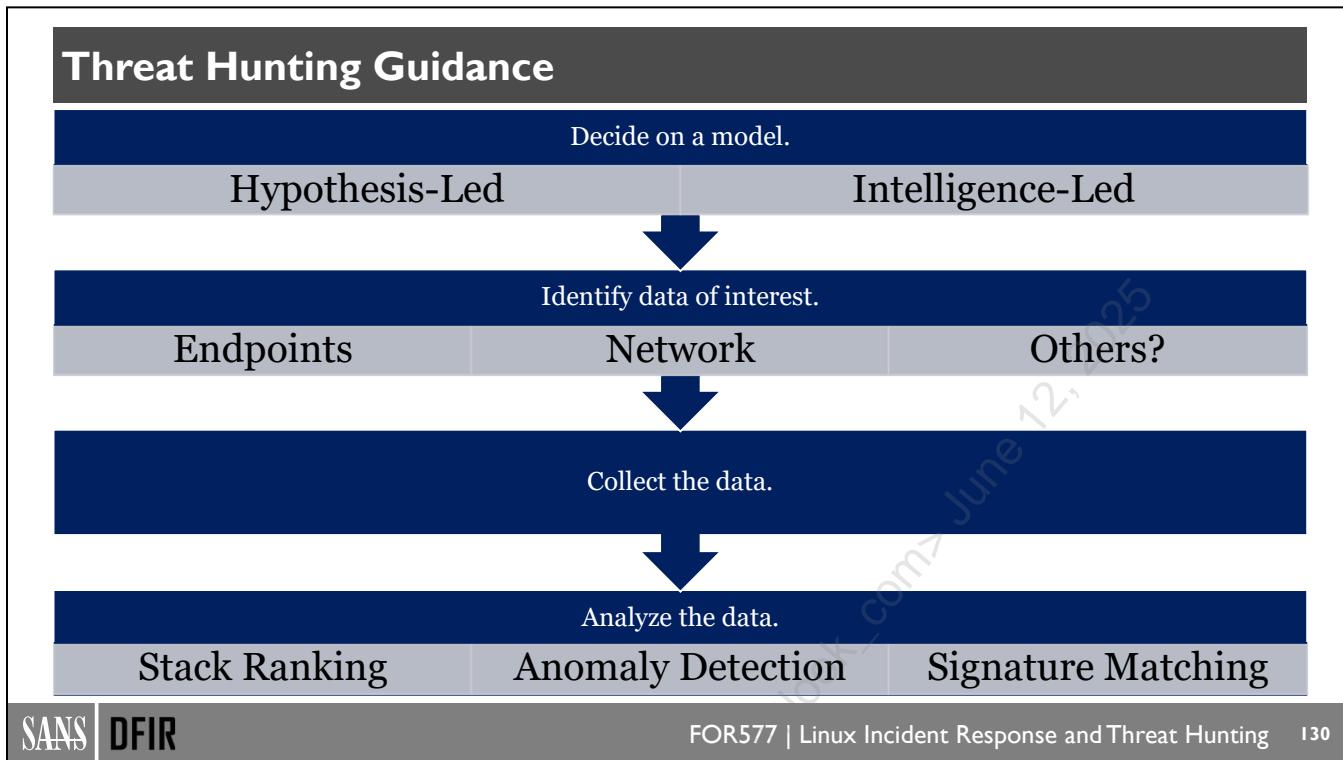
The hunt needs to be able to access the data it requires. This can be network, endpoint, or cloud data. If this isn't available, it should be identified as a hunt finding and steps taken to improve access.

Analysis of the Data

Once the data has been collected, it needs to be analyzed. The techniques used will vary depending on the data, often using some form of stack ranking to identify outliers or unusual activity. This can require specialist knowledge to maximize value.

Time

Time is required to allow the analysis to take place but also to ensure that the data collection covers enough time for attacker behavior to stand out. As an example, if there is a C2 beacon active that fires every 12 minutes, it might need several hours or even days' worth of traffic before the beacon stands out.



At a high level, there are four steps to threat hunting.

Step 1: Decide on a model.

Every threat hunt needs to be based on some form of intelligence. The threat hunter needs to have some knowledge, understanding, or idea to hunt for. This can range from the simplistic "know normal, find evil" approach or be more complex, involving deep attack chains. The two most common examples are:

- Hypothesis-led threat hunts. In this model, the threat hunters analyze potential attacker behavior, using a variety of sources of information, and try to build a hypothesis of what an attack may involve. The hunters should try to be very broadminded when creating an attack scenario here, and examples could range from "attackers may attempt to log in outside normal hours" to complex examples like "an attacker may log in, switch to the root user, install a rootkit, search for sensitive data, and exfiltrate via FTP."
- Intelligence-led hunts. Here, the threat hunters are working on actionable threat intelligence. This can be collected through internal means or provided from an external source. Examples include analyzing an ongoing attack to find indicators or receiving a call from law enforcement or a national agency warning you of adversary activity.

Step 2: Identify the data of interest.

Once the model has been determined, you need to decide on what data to collect. There are a lot of choices here, and your goal should be to find the most relevant data you are able to collect. Some common types include:

- Endpoint data. This can be audit logs, server logs, authentication logs, user profiles, file hashes, file timestamps, file and folder locations, SSH keys, AV alerts, user creation events, and more.
- Network data. This is often very useful and covers NetFlow data, traffic, and other information being sent over the wire.
- Others. This is really a catch-all for everything else and can include cloud platform API logs, container syscalls, service provider billing, and so forth.

Step 3. Collect the data.

It sounds simple, but this can be the hardest step. Once you know the data source you are interested in, you need to collect the data in the most suitable manner. It isn't possible to give specific guidance here because it does depend on the case, but examples can include using EDR to collect from endpoints, reviewing data in the SIEM, forwarding NetFlow data to an analysis tool, or running queries against cloud platforms.

Step 4: Analyze the data.

Once you have data, you need to review it to see if it matches the criteria from your model. Common techniques for this include but are not limited to the following.

- Stack ranking. This is based on the principle that odd things should be unusual in your environment and uses statistical analysis to find the things that are least frequent. This can be a very good approach if you have a standard build in your environment where individual change is rare or tightly controlled.

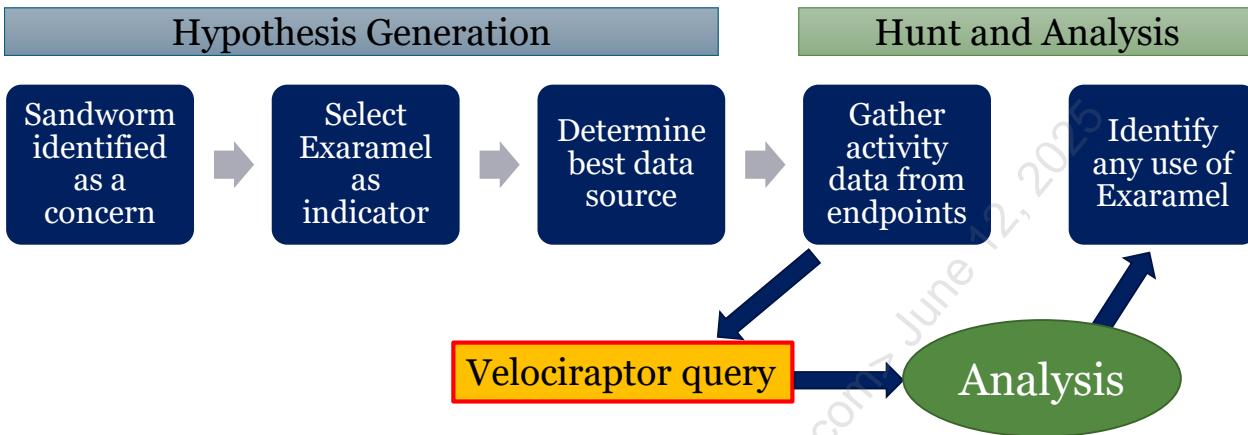
This is an example where the hunt team has collected file hashes for binaries in a web server folder from 12 systems. Most files can be found on every system and are unlikely to be malicious. The final hash is an outlier and would be worth further investigation.

MD5 Hash	Count
c9aaaf6b8b4b47df9253f43c3f344c5b2	12
6312632c0218b75c460af7ad774d0514	12
7a84493663d77c39e4642e898a837ce1	11
932f13a92a8b8bdb52f62bb0232026c6	10
ddb177af1d4f6eec1c3324a3faadc7c	10
1789f488f53f69796b7ac8c602858280	10
c7601e407716898a1a0e947dffcb89dc	1

- Anomaly detection. Similar to the above, this involves reviewing the data to find things that are abnormal in other ways. This can include finding Sysadmins logging in outside their normal working day or from locations they have never been to. Other examples include checking for crash reports, unexpected restarts, or even unusual emails.
- Signature matching. The final example is using known attacker signatures to hunt through the data. The most common way is to search for known malicious hashes, commands, or specific tools. Signatures can also include known command and control IP addresses, attacker domain names, and even tool names. A very common example of a signature to look for is the /usr/bin/whoami command (often referred to simply as whoami). This is frequently used by attackers early during the attack and very rarely used by normal users.

As you can see, the different techniques do overlap, and you may want to use a combination of methods to get the best result. With threat hunting, it is important that you keep an open mind and be willing to assess the data in multiple ways.

Example Threat Hunt: Hypothesis-Led Hunt



Our first example is a hypothesis-led hunt. To show how this might work, we have used a free endpoint detection and response (EDR) tool called Velociraptor as our example, and this is covered later in the course.

Hypothesis Generation

It is important to make sure that this hunt has a valid, effective, and well-thought-out hypothesis.

First, the organization identifies a threat actor/group they feel presents a threat. This analysis identifies SANDWORM/APT28 as a group of concern.

Next, the hunt team reviews the MITRE ATT&CK framework¹ for any useful information on this group to determine a good indicator to hunt for.

After reviewing the data and based on previous hunts/incidents, it is determined that the use of Exaramel would be a good indicator. The hunt team made this decision because the tool isn't something that should exist in the environment, and its presence is a strong sign of compromise.

Finally, a decision is made as to what data will best serve this hunt. One option might be to collect the file hash of every executable and compare that with known entries for Exaramel. This may have a high false negative rate, as file hashes can be easily changed, but it is a good starting point. Alternative hunts could include looking through audit logs or other records of execution.

Hunt and Analysis

The second phase is where the hunt takes place.

First, the data should be collected from the endpoints. As the enterprise uses Velociraptor, the hunt team may decide to use a file finder to collect MD5 hashes and match them with known examples. This is an example of what a hunt might look like:

State	Hunt ID	Description	Created	Started	Expires	Scheduled	Creator
PAUSED	H.CKI42AGJ7N458	Sandworm group - Exaramel Tool	2023-10-09T18:06:34Z		2023-10-16T18:04:39Z		admin

Overview

Results

Total scheduled: 0
Finished clients: 0
Download Results

Available Downloads

name	size	date

(Note: the exact syntax would need to be tested for your environment, and this specific hunt may be very resource-intensive).

Once the data has been collected, it can be analyzed to determine if any data matches the available threat intelligence.

OSPath	Inode	Mode	Size	ATime	MTime	CTime	Keywords	Upload	Hash	FlowId	ClientId	Fqdn
/home/sansfo_rensics	214748 4136	4096	2023-10-09T18:11:37Z	2023-10-09T18:11:32Z	2023-10-09T18:11:32Z					F.CKI46I	C.2ccc27ad	local
										TJU28GK	a2f61b79	host
/home/sansfo_rensics/.bash_aliases	420	473	2023-10-09T17:00:18Z	2023-05-10T16:58:35Z	2023-05-10T16:58:35Z					F.CKI46I	C.2ccc27ad	local

MD5 : 539254e39b9e2bc8cba82bffc99ed3a
SHA1 : 739cb6840219d2b7cd3891bee8190393a192b4b2
SHA256 : 00

1. Sandworm group on MITRE <https://for577.com/sandworm>

Generating a Hypothesis

There is no
"one true way"

- Find what works for you.
- Try lots of things.

Consider likely
threat actors

- Industry.
- Geography.
- Assets/intellectual property and so on.

Use an "assume
breach" model

- Threat hunting looks for attacks that bypassed your controls.
- Find alternative ways to detect.

With a hypothesis-led threat hunt, the first step is to generate the hypothesis you want to hunt! Generating this first idea can often feel daunting. The most important thing to remember is that it is better to come up with any idea to start your hunt, rather than spend time trying to decide if you have the right or best idea. The following are some approaches you can take to build a hypothesis:

Threat Actor Focus

With this approach, you analyze likely threat actors based on attributes such as your industry sector, geographical region, or what assets you have. Examples include being in the Financial Services sector and having advanced pharmaceutical research. However, don't think these are the only sectors targeted. Sometimes an attacker simply wants to use your infrastructure.

This can involve using threat intelligence but is different from an intelligence-led hunt, as you don't have any specific TI to work from. Instead, you are creating a hypothesis that a specific threat actor might have targeted you and might use certain techniques to gain access.

Assumed Breach

An alternative approach is to work on the assumption that you've been breached and then look for likely signs. In this model, you could decide that your EDR/AV tools have been bypassed and then hunt for techniques that might allow it. Or you could assume that your administrator's credentials have been compromised and you can hunt through login data to see if anything unusual stands out.

AI-Assisted Generation: Generating the Hypothesis

ChatGPT 4o ▾

↑

My organization believes they are being targeted by APT28. Based on publicly available reporting, including the MITRE ATT&CK groups page, what would you recommend was the best threat hunting hypothesis?

Searched 5 sites

Given that your organization believes it is being targeted by APT28 (also known as Fancy Bear), a suitable threat hunting hypothesis should focus on the tactics, techniques, and procedures (TTPs) commonly associated with this group. Based on publicly available reporting, including the MITRE ATT&CK framework, the following hypothesis would be effective:

Hypothesis: APT28 is leveraging spear-phishing attacks with malicious attachments or links to compromise credentials and deploy custom malware, followed by the use of network device vulnerabilities to maintain persistence and facilitate data exfiltration.

Hypothesis: APT28 is leveraging spear-phishing attacks with malicious attachments or links to compromise credentials and deploy custom malware, followed by the use of network device vulnerabilities to maintain persistence and facilitate data exfiltration.

Supporting Rationale:

1. **Phishing and Credential Harvesting:** APT28 is known for conducting spear-phishing campaigns using malicious attachments and links to harvest credentials. They often register domains that closely resemble legitimate ones to deceive targets and capture login details (CrowdStrike) (Strapi).



You can also look to use an AI/LLM model to generate your threat-hunting hypothesis. This can be an effective way to provide a natural language description of the issue you are concerned about and then allow the AI platform to generate ideas to work on.

In this example, we have an organization concerned about APT28, and ChatGPT 4o is being used to generate the hunting plan. The advantage of this is that most AI/LLM platforms will provide additional information such as:

- A rationale for the hypothesis
- High-level hunting guidance
- Source material links so you can validate the idea

This can be used to build documentation and provide assurance statements to executive leadership. To show you what this can look like, the information below is the output from ChatGPT 4o when asked this question on 2024-05-30. Note: AI/LLMs are constantly updating their understanding and models, and you might get different answers.

ChatGPT Output

Hypothesis: APT28 is leveraging spear-phishing attacks with malicious attachments or links to compromise credentials and deploy custom malware, followed by the use of network device vulnerabilities to maintain persistence and facilitate data exfiltration.

Supporting Rationale:

Phishing and Credential Harvesting: APT28 is known for conducting spear-phishing campaigns using malicious attachments and links to harvest credentials. They often register domains that closely resemble legitimate ones to deceive targets and capture login details (CrowdStrike) (Strapi).

Exploitation of Network Devices: The group has been reported exploiting vulnerabilities in network devices, such as Cisco routers, to gain access and deploy malware. Notable vulnerabilities include CVE-2017-6742, which they have used to compromise network infrastructure globally (CISA).

Custom Malware Deployment: APT28 utilizes sophisticated malware like X-Agent, X-Tunnel, and other proprietary tools to maintain persistence and conduct further operations. These tools are cross-platform, affecting both traditional computing environments and mobile devices (CrowdStrike).

Brute Force and Password Spraying: They have also been observed conducting brute force and password spraying attacks to gain unauthorized access to various accounts and systems. These techniques are typically employed to penetrate weakly protected accounts and expand their foothold within the targeted network (MITRE ATT&CK).

Recommended Threat Hunting Activities:

- Monitor Email Gateways: Look for indicators of spear-phishing attempts, such as emails with suspicious attachments or links and domains resembling legitimate ones.
- Inspect Network Traffic: Pay close attention to unusual outbound traffic, especially connections to known APT28 command-and-control (C2) servers or domains.
- Vulnerability Scanning: Regularly scan network devices for known vulnerabilities, particularly those related to Cisco routers, and ensure they are patched promptly.
- Analyze Authentication Logs: Search for signs of brute force or password spraying attacks, such as repeated failed login attempts followed by successful access, especially from external IP addresses.

(End of output)

This provides a range of ideas and starting points that your threat-hunting team can use to track any suspicious activity.

AI Assisted Generation: Creating Searches

Based on publicly available reports and the MITRE ATT&CK framework, provide the Splunk query string to search for evidence of Fysbis malware in a Linux environment. Each device feeding into the SIEM has auditd enabled to the CIS benchmarks Level 2.

- To search for evidence of Fysbis malware in a Linux environment using Splunk, you can create a query that leverages `auditd` logs. Fysbis (also known as Linux/Trojan.Fysbis) is known for its command and control capabilities and can be identified through its behavior, such as specific process creation and network activity.

Here is a Splunk query string that you can use to search for indicators of Fysbis malware:

```
splunk
index=your_index sourcetype=auditd
| eval cmd=mv,exe
| search cmd IN ("wget", "curl") AND (exe IN ("*/tmp/*", "*/dev/shm/*", "*/.cache/*"))
| stats count by host, exe, cmd, pid, uid, user
| where count > 10
```

Hypothesis

Hunt query

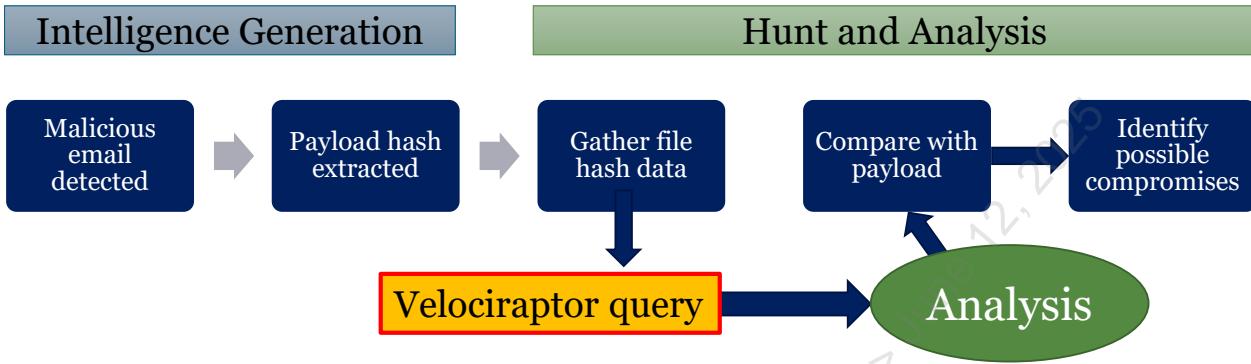
Hypothesis generation is the first part of building a hunt. The next step is to determine how to *run* the hunt, and AI/LLM can assist here as well. This is another opportunity to use natural language queries to build detailed technical responses.

In the example on this slide, the threat hunter is looking to build some hunts for the Fysbis malware family in their Linux environment. The hunt is based on publicly available data. They have specified that auditing is enabled to the standards of CIS Benchmarks Level 2. This allows the LLM to estimate what log data is likely to be available. The threat hunter has also specified in what format they want the search query, thus helping speed up the transition from hypothesis to hunt.

While the output will always vary by AI/LLM platform, you can generally provide a very simple, plaintext explanation of what you want. There are some important considerations, though:

- The more information you can provide, the more accurate and useful the response will be. If you are using a locally built platform, it may already understand your environment, but, if not, you need to be very clear about what your data is likely to look like.
- You really need to test the output. While using AI definitely helps, there will be situations where the response contains errors or inaccuracies. Also, you will always need to tailor this to your environmental specifics. This includes index names, formatting, etc. Avoid blindly copying and pasting the responses without an understanding of what the tool is telling you.

Example Threat Hunt: Intelligence-Led Hunt



The second example is using internally generated threat intelligence as the basis for a threat hunt.

Intelligence Generation

This can come from multiple sources. In our example, an employee reports a suspicious email that has made it through filtering. The email contains a link; if the users visit the link, a browser exploit fires, and the attackers download malware to the user's device (e.g., a beacon or C2 implant).

The security operations center (SOC) has analyzed the mail and updated the filters to prevent any new mail from being delivered. It has checked the mail server to remove any existing copies, and the proxy/firewall has been configured to prevent users from visiting the malicious site. However, there is a risk that copies might have been delivered to users before it was identified, so a threat hunt is called for.

First, the hunt team gathers the indicators they want to hunt for. In this example, it is going to be the SHA1 hash of the malicious download. Once the hashes are identified, the hunt team uses the EDR tool to scan the environment to determine if it is present anywhere.

In this hunt, any detections are likely to indicate a possible compromise and should lead to direct incident response.

Using Threat Intelligence: Introducing YARA

```

rule Sofacy_Fysbis_ELF_Backdoor_Gen2
{
    meta:
        description = "Detects Sofacy Fysbis Linux Backdoor"
        author = "Florian Roth"
        reference = "http://researchcenter.paloaltonetworks.com/2016/02/a-look-into-fysbis-sofacys-linux-backdoor/"
        date = "2016-02-13"

    strings:
        $s1 = "RemoteShell" ascii
        $s2 = "basic_string::_M_replace_dispatch" fullword ascii
        $s3 = "HttpChannel" ascii

    condition:
        uint16(0) == 0x457f and filesize < 500KB and all of them
}

```

YARA is a powerful, open-source tool for identifying and classifying files based on patterns and signatures. It is primarily used for malware analysis and threat intelligence, but it can also be used for other purposes, such as finding specific files or detecting vulnerabilities in software. YARA uses a flexible syntax that allows users to define their own rules and signatures, and it can search files for these rules at high speed, making it a valuable tool for incident response and threat hunting.

YARA works by defining a set of rules, which are essentially patterns or regular expressions that describe the characteristics of a file. These rules can be based on a wide range of attributes, such as filenames, file sizes, and specific strings of code or data. Once the rules are defined, YARA can scan a set of files or a directory for matches and generate reports on any files that meet the criteria.

In the example on this slide, the matched condition requires:

- It must have the file signature for a Linux ELF binary: Hex 45 7F. The full file signature is Hex 7F 45 4C 46. This is a hex 7F followed by the letters ELF.
- The file must be smaller than the specified maximum size.
- Matching all three of the specific strings identified is required.

The goal of an IOC is to create a signature that is specific enough to limit false positives at scale while being broad enough to still match different variants of the same malware sample. This is a difficult balance and is why different IOCs have wildly different efficacy rates.

Using YARA

yara [OPTIONS] RULES_FILE TARGET

```
user@midgard:~/samples$ yara ../*.yar -r ~/samples
Sofacy_Fysbis_ELF_Backdoor_Gen2 /home/user/samples/userapp
Sofacy_Fybis_ELF_Backdoor_Gen1 /home/user/samples/mailsweeper
Sofacy_Fysbis_ELF_Backdoor_Gen3 /home/user/samples/minesweeper_for_linux
Sofacy_ELF_Anti_Forensics /home/user/samples/time
Sofacy_Fybis_ELF_Backdoor_Gen1 /home/user/samples/door_app
```

Note: Using wildcards (*) can result in errors.

Running YARA from the command line is relatively easy, and it provides a good way to quickly scan a large number of objects for known malicious content. The command syntax is yara, followed by any options, then a rules file and a target.

The options include:

-C --compiled-rules	RULES_FILE contains rules already compiled with yarac.
-c --count	Print only the number of matches.
-d <identifier>=<value>	Define the external variable.
--fail-on-warnings	Treat warnings as errors; has no effect if used with --no-warnings.
-f --fast-scan	This is a fast-matching mode.
-h --help	Show help.
-i <identifier> --identifier=<identifier>	Print rules named <identifier> and ignore the rest.
--max-process-memory-chunk=<size>	When scanning process memory, read the data in chunks of the given size.
-l <number> --max-rules=<number>	Abort scanning after matching a number of rules.
--max-strings-per-rule=<number>	Set the maximum number of strings per rule (default=10000). If a rule has more than the specified number of strings, an error will occur.
-n --negate	Print not satisfied rules only (negate).
-N --no-follow-symlinks	Do not follow symlinks when scanning.
-w --no-warnings	Disable warnings.
-m --print-meta	Print metadata.
-D --print-module-data	Print module data.
-e --print-namespace	Print rules' namespace.

-S --print-stats	Print rules' statistics.
-s --print-strings	Print matching strings.
-L --print-string-length	Print the length of matching strings.
-g --print-tags	Print tags.
-r --recursive	Recursively search for directories. It follows symlinks.
--scan-list	Scan files listed in FILE, one per line.
-z <size> --skip-larger=<size>	Skip files larger than the given <size> in bytes when scanning a directory.
-t <tag> --tag=<tag>	Print rules tagged as <tag> and ignore the rest.
-p <number> --threads=<number>	Use the specified <number> of threads to scan a directory.
-a <seconds> --timeout=<seconds>	Abort scanning after a number of seconds has elapsed.
-v --version	Show version information.



Lab 1.4: Consuming Threat Intelligence

Estimated time: 30 minutes



FOR577 | Linux Incident Response and Threat Hunting 142

This page intentionally left blank.

DIGITAL FORENSICS

	
FOR498 Digital Acquisition and Rapid Triage GBFA	FOR500 Windows Forensic Analysis GCFE
	
FOR518 Mac and iOS Forensic Analysis & Incident Response GIME	FOR585 Smartphone Forensic Analysis In-Depth GASF

INCIDENT RESPONSE & THREAT HUNTING

	
FOR508 Advanced Incident Response, Threat Hunting & Digital Forensics GCFA	FOR509 Enterprise Cloud Forensics and Incident Response GCFR
	
FOR572 Advanced Network Forensics: Threat Hunting, Analysis, and Incident Response GNFA	FOR577 LINUX Incident Response and Threat Hunting
	
FOR608 Enterprise-Class Incident Response & Threat Hunting GEIR	FOR610 REM: Malware Analysis Tools & Techniques GREM
	
FOR710 Reverse-Engineering Malware: Advanced Code Analysis	SEC504 Hacker Tools, Techniques & Incident Handling GCIH

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

Wherever he steps, wherever he touches, whatever he leaves, even without consciousness, will serve as a silent witness against him.
—Sherlock Holmes

AUTHOR CONTACT

 twake@sans.org
<https://twitter.com/tazwake>

SANS INSTITUTE

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

DFIR RESOURCES

 digital-forensics.sans.org
Twitter: @sansforensics

SANS EMAIL

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

This page intentionally left blank.