# STEP 1: Install Sysmon (System Monitor)

1. Instructions to install Sysmon:

<https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>

Extract the .zip file (contains Sysmon.exe and Sysmon64.exe).

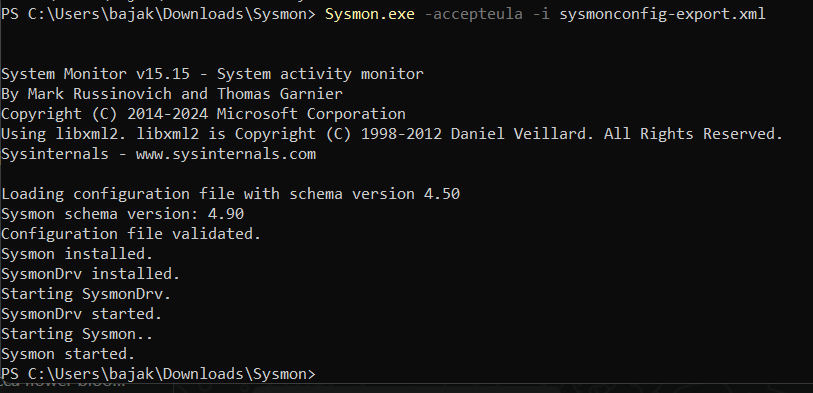
1. Download a Config File

<https://github.com/SwiftOnSecurity/sysmon-config>

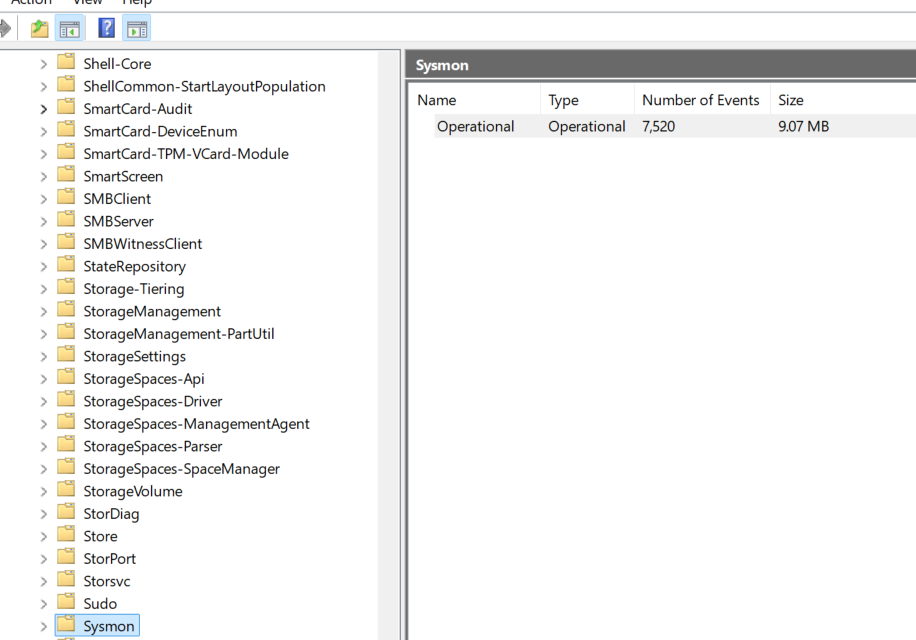
Extract and locate the config file: sysmonconfig-export.xml.

1. Install Sysmon with the Config

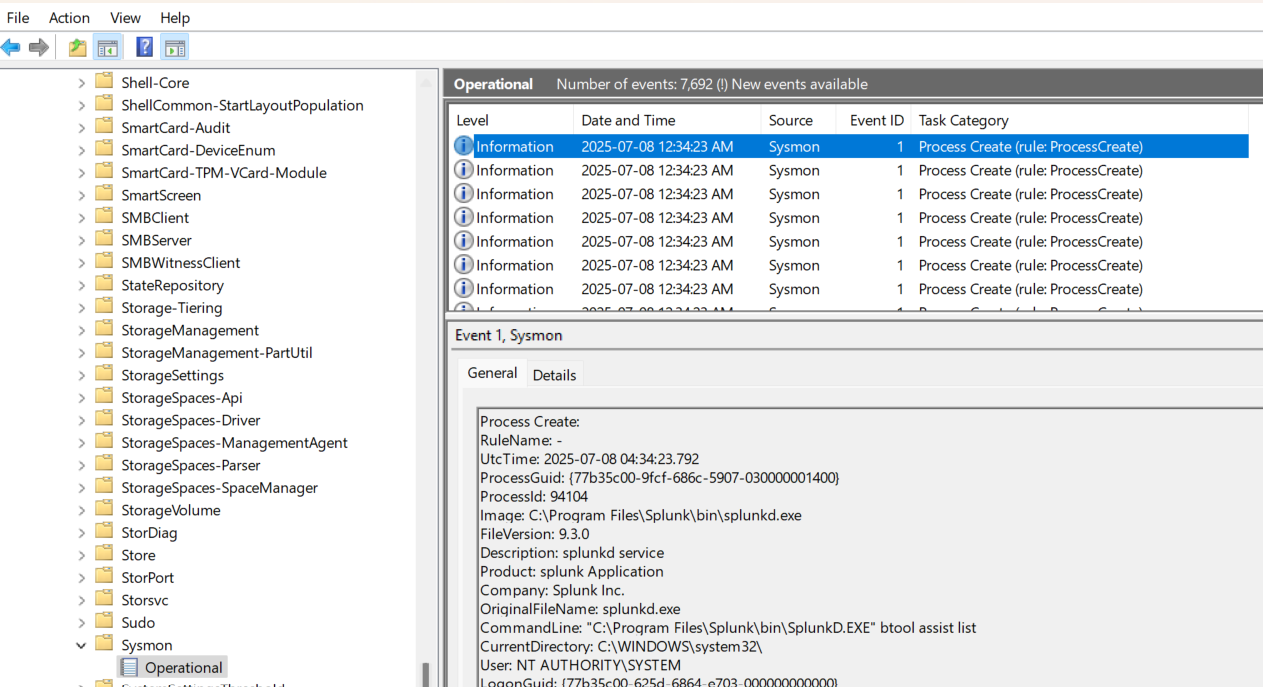
Sysmon64.exe -accepteula -i sysmonconfig-export.xml



Able to see Sysmon folder in Event viewer after installation



Confirmation that Sysmon is running on windows host machine.



# STEP 2: Install and Configure Splunk Universal Forwarder

## Download the Splunk Universal Forwarder.

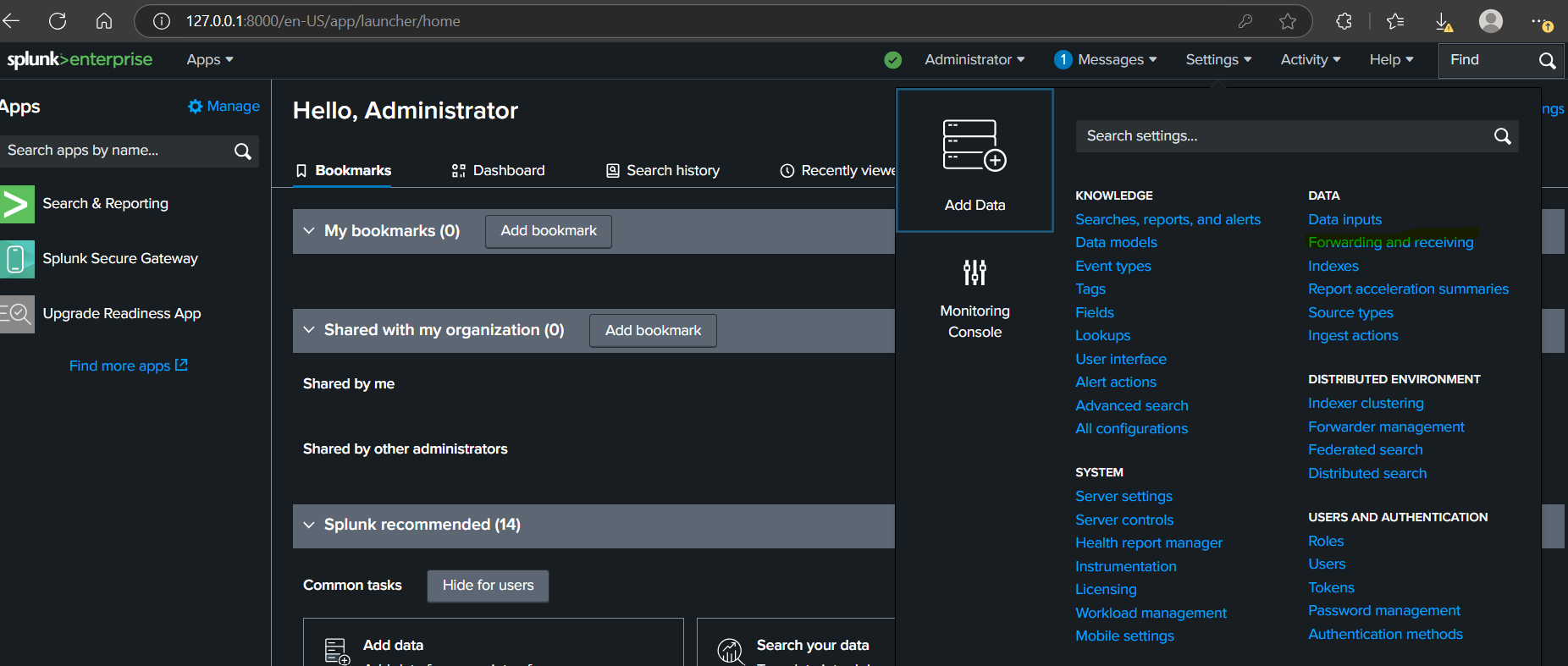
Go to:  
<https://www.splunk.com/en_us/download/universal-forwarder.html>

Download the version matching your system (Windows 64-bit .msi).

**Note**: if Splunk instance is already downloaded on the host machine, connect UF to Splunk Enterprise Instance.

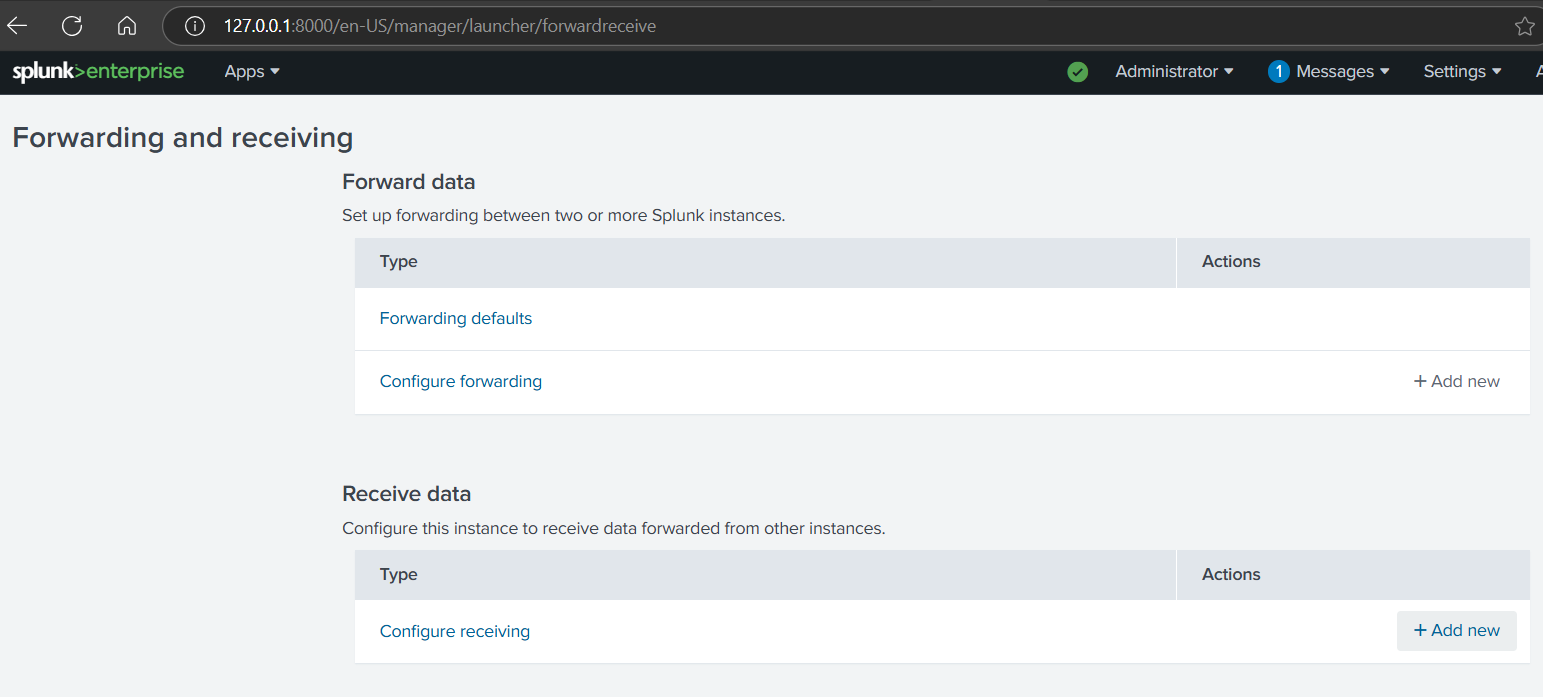
## Connect UF to Your Splunk Enterprise Instance

Let’s now configure your **Splunk Universal Forwarder (UF)** to send logs to your **Splunk Enterprise trial instance**.

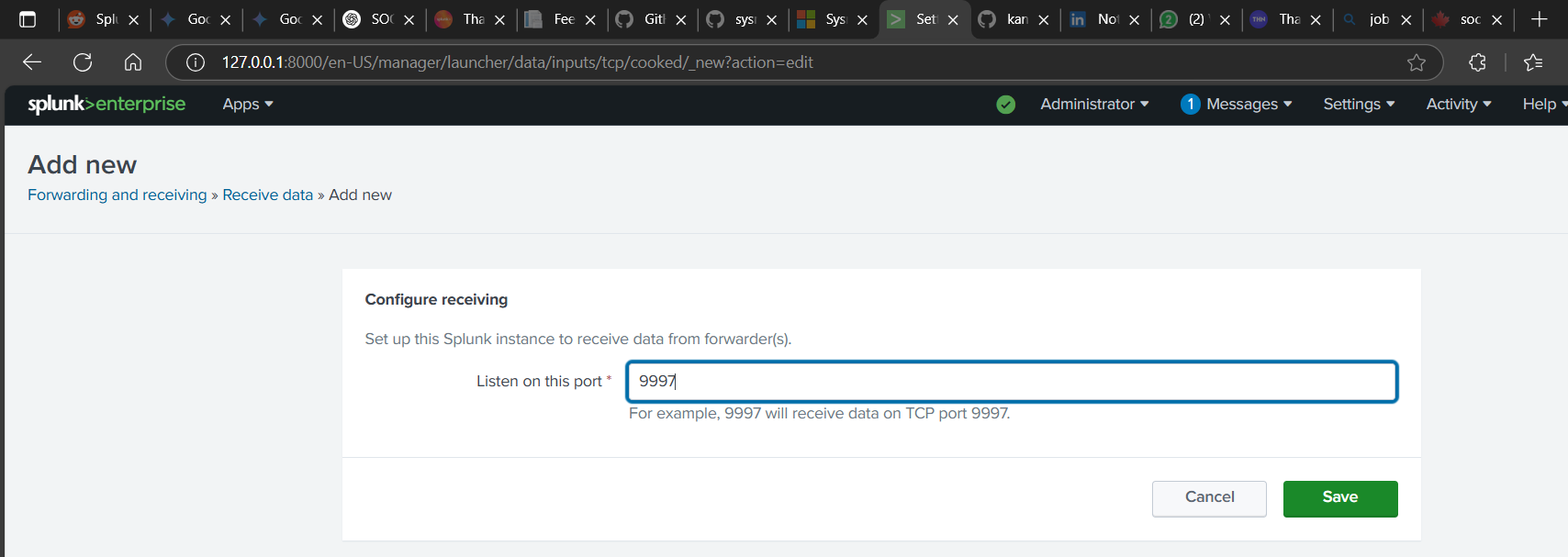


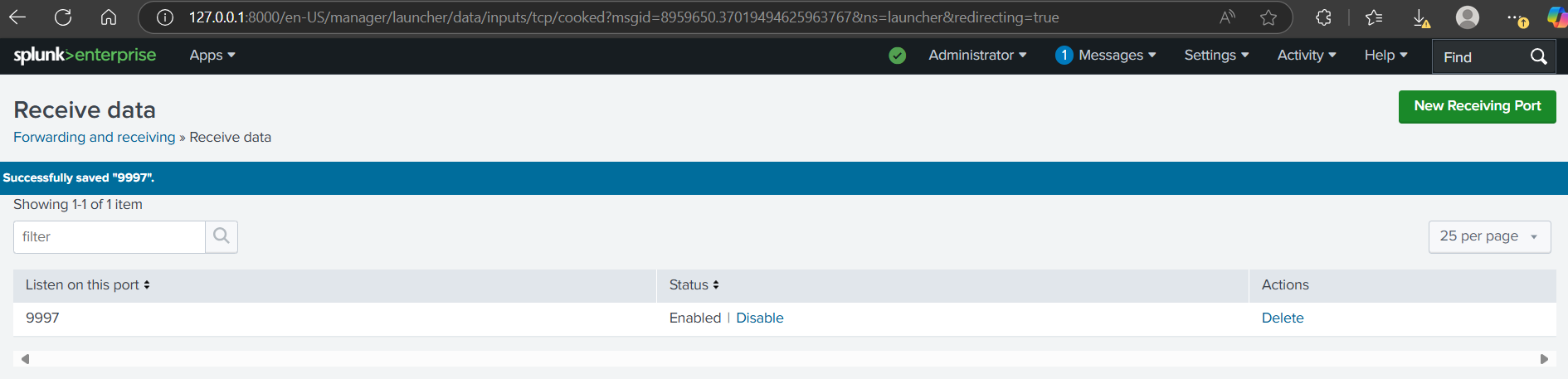
1. Login to Splunk Web (usually http://localhost:8000)
2. Go to:  
   Settings > Forwarding and Receiving > Receive data
3. Click Add New under “Receive Data on Port”.
4. Enter 9997 (default UF-to-Splunk port), then click Save.

🧪 Now your Splunk is listening for data from forwarders on port 9997.



Click on Add New button under Receive data





## Configure the Forwarder to Splunk

On the system where **Sysmon and the Universal Forwarder** are installed:

1. Open **Command Prompt as Administrator**
2. Go to the UF bin folder:

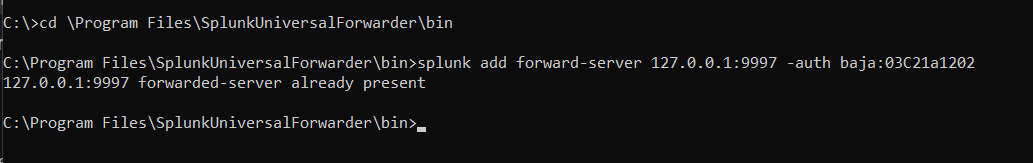
cd "C:\Program Files\SplunkUniversalForwarder\bin"

splunk add forward-server <SPLUNK\_ENTERPRISE\_IP>:9997 -auth admin:changeme

* Replace <SPLUNK\_ENTERPRISE\_IP> with the IP address of your host machine (if same machine: 127.0.0.1)
* Replace admin:changeme with your actual Splunk Enterprise credentials

✔️ You should see:

Successfully added forward-server



# STEP3: Configure inputs.conf to Collect Logs

**Step-by-Step to Create inputs.conf**

1. Open **Notepad as Administrator**
   * Click Start → Search for **Notepad**
   * Right-click → **Run as administrator**
2. Paste this content (or a variation):

[default]

[default]

host = SwaBha

[WinEventLog://System]

disabled = 0

index = winlog

[WinEventLog://Application]

disabled = 0

index = winlog

[WinEventLog://Microsoft-Windows-Sysmon/Operational]

disabled = 0

index = sysmon

[WinEventLog://Security]

disabled = 0

index = winlog

**note**: please make sure what ever the indexes being specified in inputs.conf file has to be created and exist on splunk enterprise instance.

Also make sure the Splunk UF has sufficient permissions by changing the service **from NT SERVICE\SplunkForwarder to Local System account**. **This allows the UF to run with broad permissions**, including reading protected event logs like Sysmon.

You're saying:

* ✅ **Security** → collect logs like login attempts (e.g. EventCode 4624, 4625)
* ✅ **System** → OS-level events (service failures, system crashes)
* ✅ **Application** → app-related logs
* ✅ **Sysmon/Operational** → detailed process, network, file, and registry activity

**🔄 Then what happens?**

1. The UF **reads those logs** from Windows Event Viewer.
2. It **forwards them over port 9997** to your Splunk Enterprise instance.
3. Splunk **indexes them** so you can search, build dashboards, detect threats, etc.

# STEP 4: Restart the Splunk Forwarder

Save the file as:

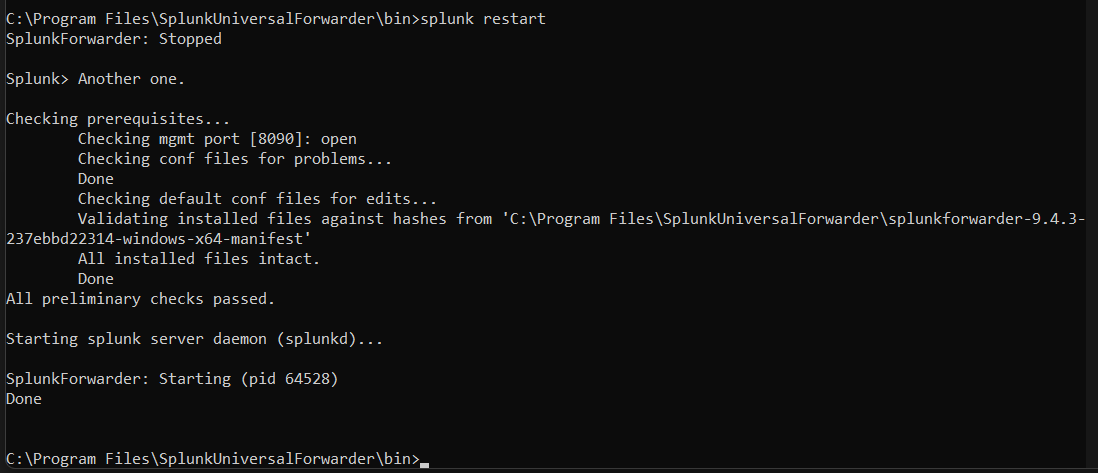
C:\Program Files\SplunkUniversalForwarder\etc\system\local\inputs.conf

Make sure to choose **"All Files (*.*)"** in the Save dialog so Notepad doesn't save it as inputs.conf.txt

After creating the file restart the Forwarder

Using power shell: Restart-Service splunkforwarder

Or in CMD: splunk restart



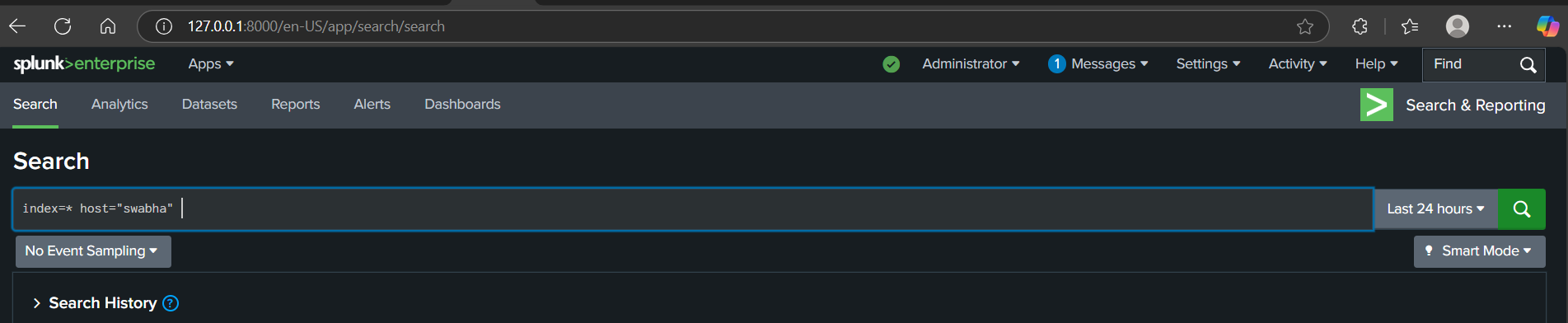
# STEP5: Verify Data in Splunk Enterprise

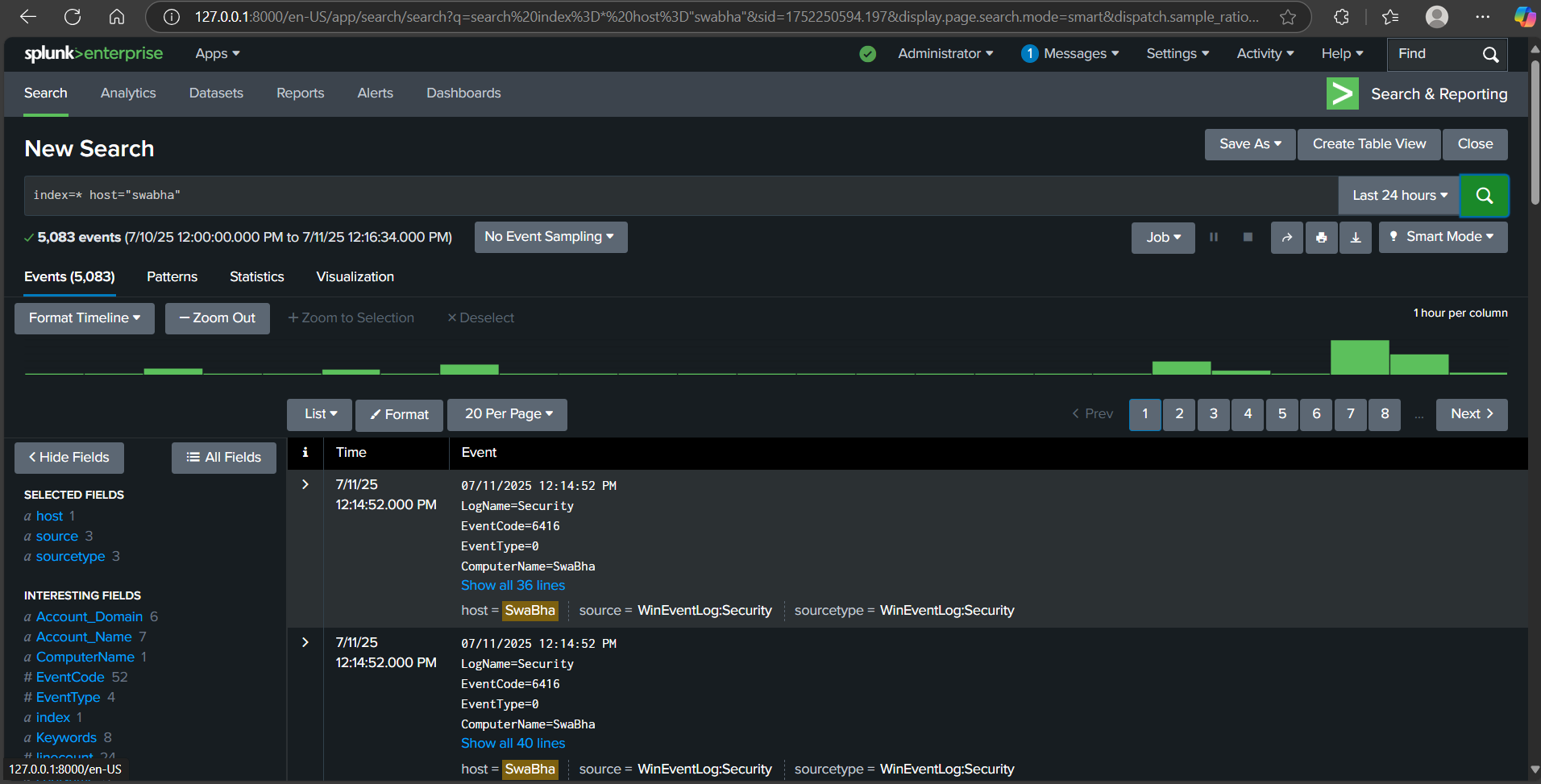
**To Verify It’s Working**

In your **Splunk Web UI**, run:

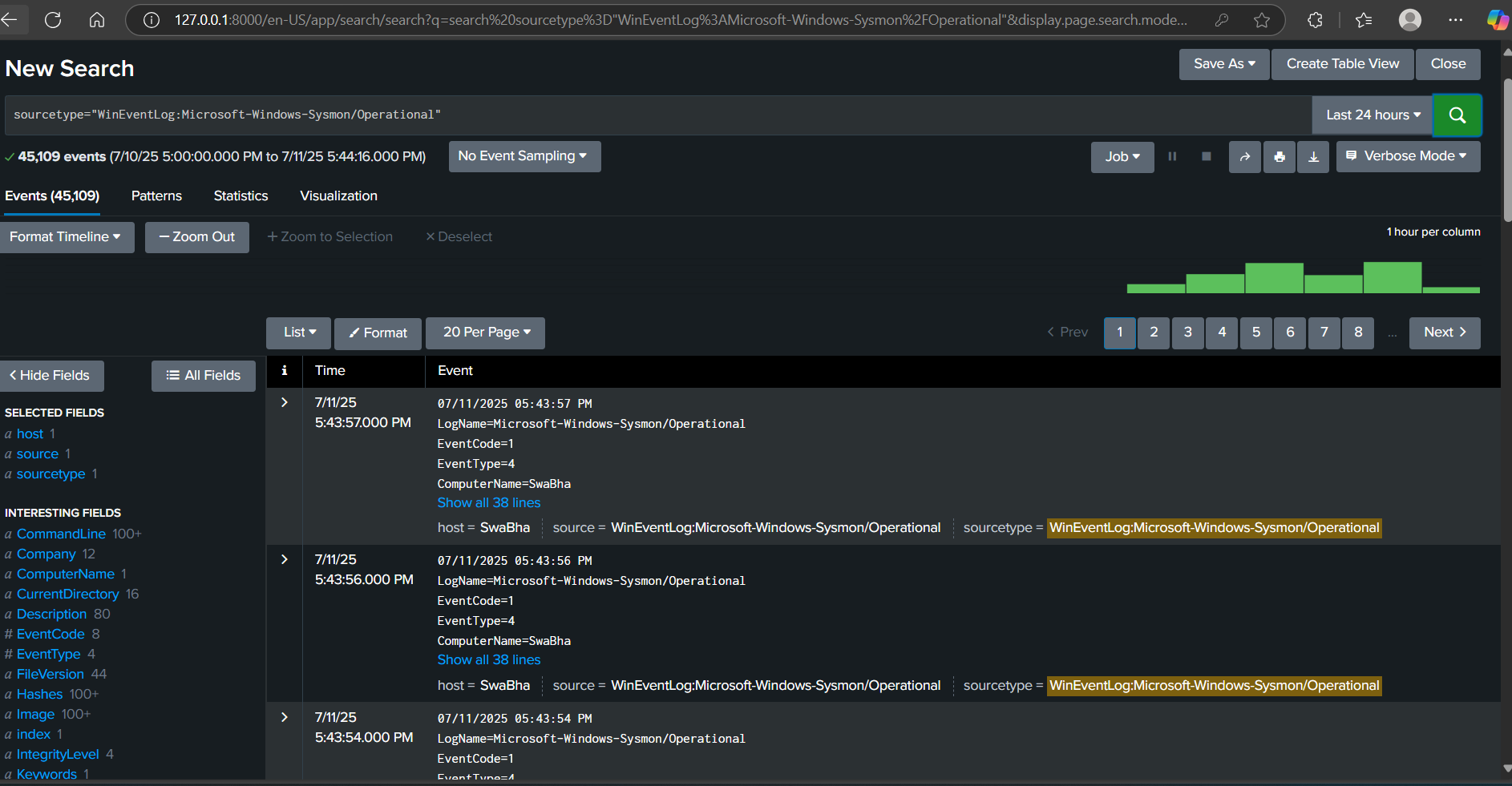
1. Check all forwarded logs: index=\* host=YOUR-HOSTNAME

You should start seeing Windows and Sysmon logs coming in.

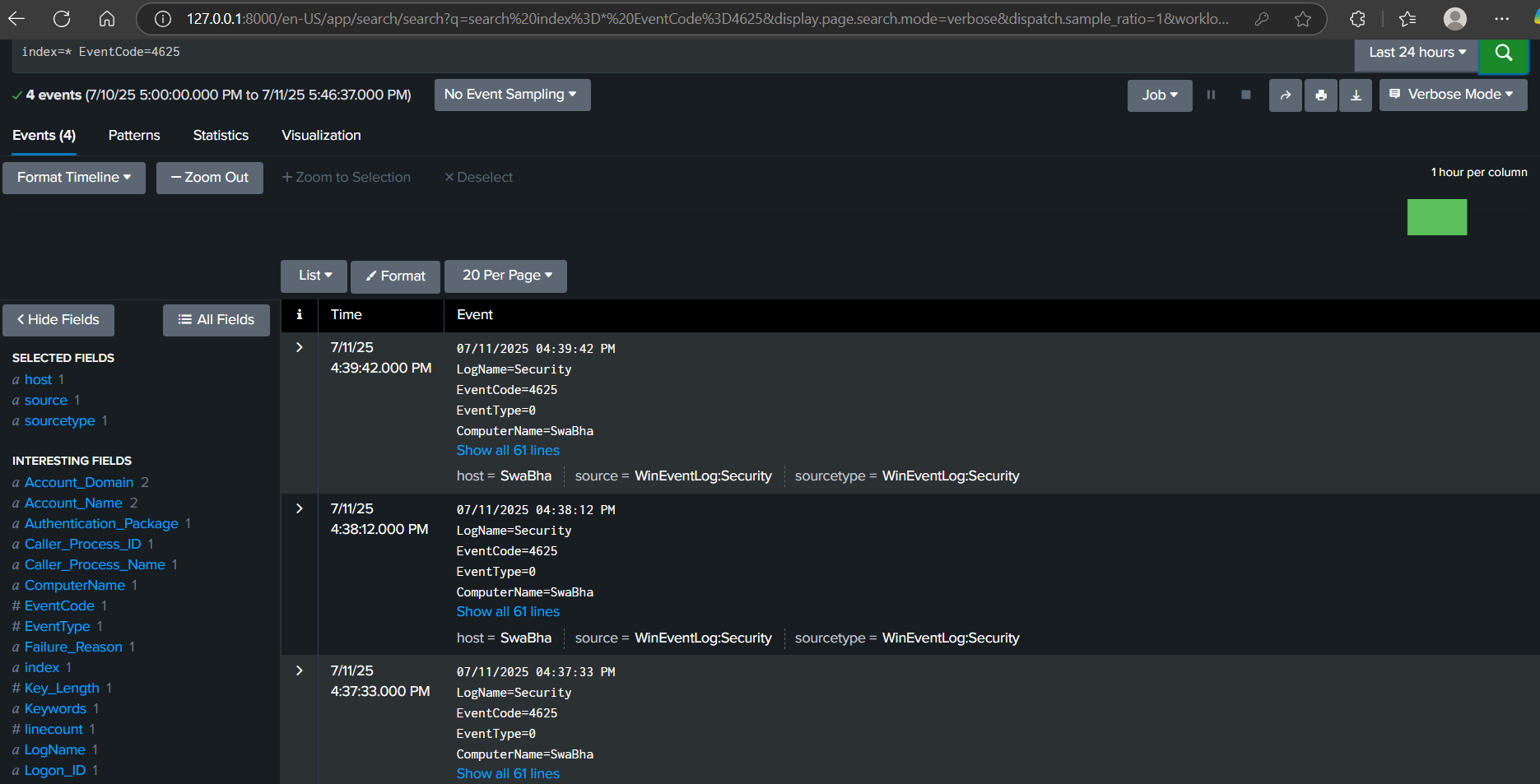




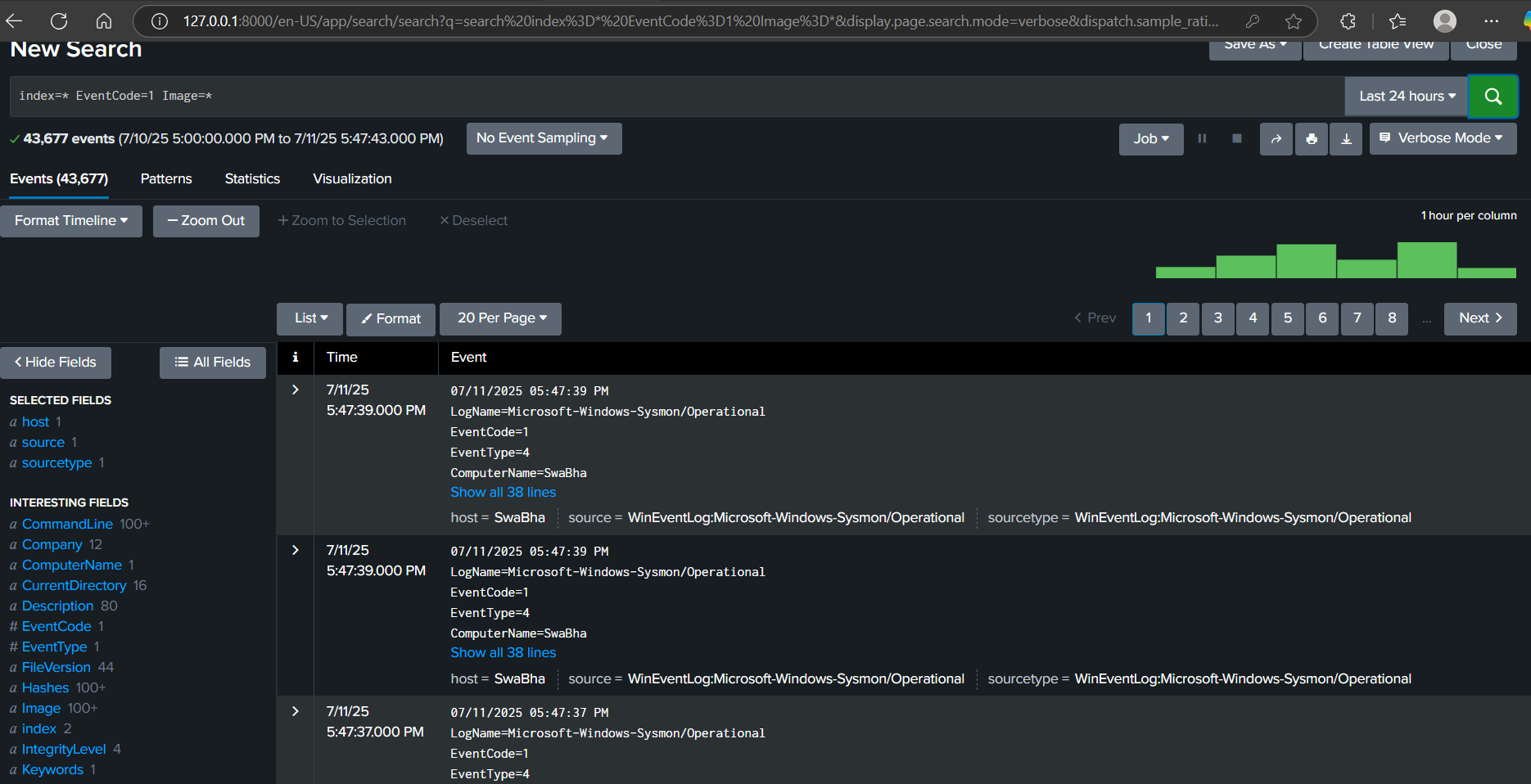
1. Check Sysmon logs specifically: sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational"



1. Check for failed logins (EventCode 4625): index=\* EventCode=4625



1. Check for process creation (Sysmon EventCode 1): index=\* EventCode=1 Image=\*



# Category: Authentication & Access

## Multiple Failed Login Attempts (Brute-force Attempt)

**The Threat/Technique (Brute-Force / Password Spraying and Compromise):**

* Attackers frequently attempt to gain unauthorized access by guessing credentials.
* **Brute-force attacks** involve systematically trying many passwords for a single username.
* **Password spraying attacks** involve trying one common password against many usernames.
* When multiple failed login attempts are quickly followed by a successful login for the *same account*, it's a strong indicator that one of these automated attacks (or even a persistent manual attempt) has succeeded, meaning the account might be compromised.

**I verified:**

1. **Simulation:** You manually simulated a brute-force or password spraying attempt by making multiple failed logins for an account, followed by a single successful login for that same account.
2. **Endpoint Visibility:** You confirmed Windows generated the appropriate **Event ID 4625 (failed)** and **Event ID 4624 (successful)** logs in the Security Event Log.
3. **Splunk Detection:** You successfully used a Splunk search query with the transaction command to correlate these individual login events into a sequence, identifying specific instances where an account successfully logged in shortly after experiencing multiple failures.

**Purpose:** Detect suspicious authentication failures (e.g., brute-force or mistyped passwords)

**Simulate:**

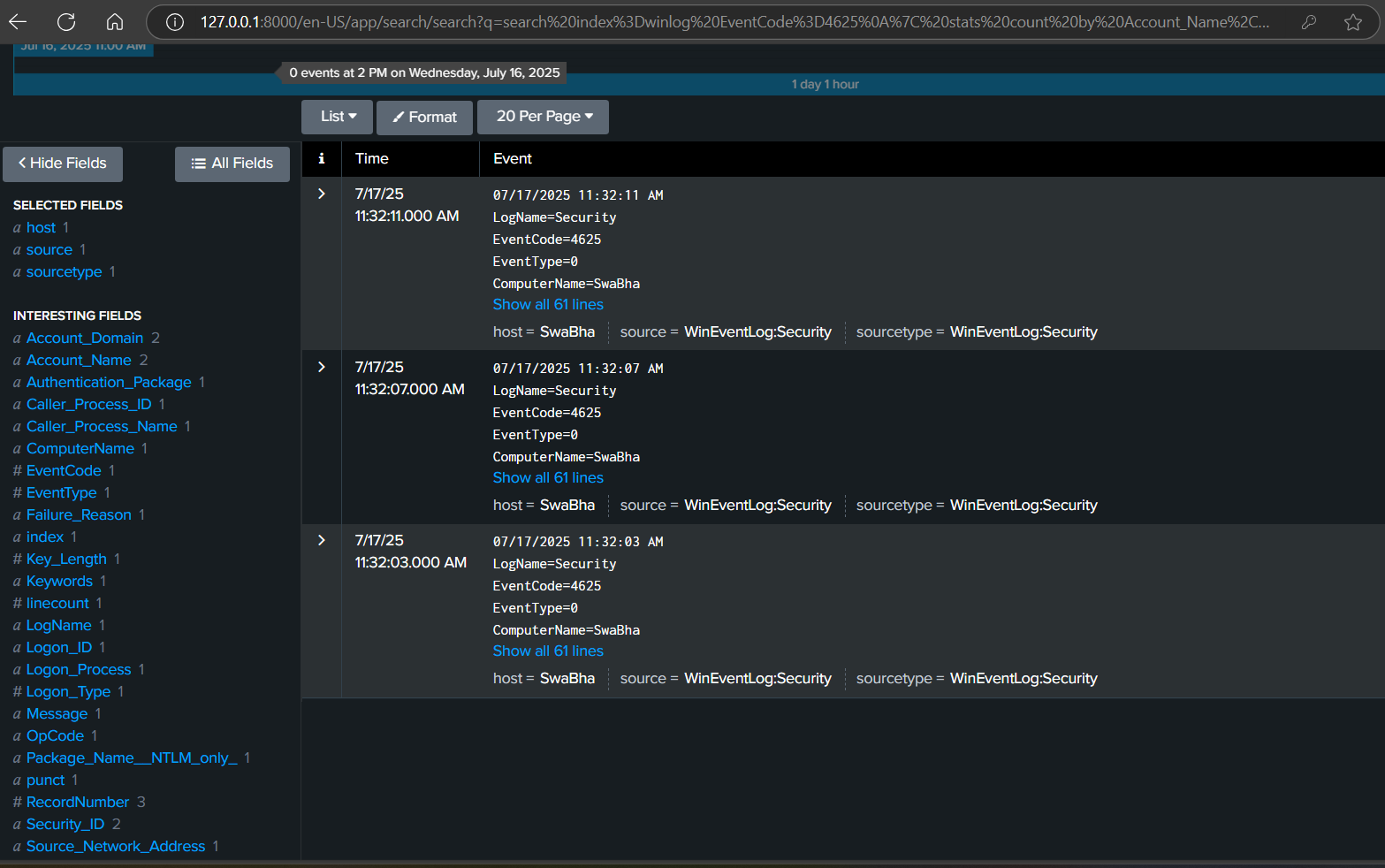
1. Lock your screen (press Win + L)
2. Attempt to log in with the wrong password 4–5 times
3. Log in with the correct password

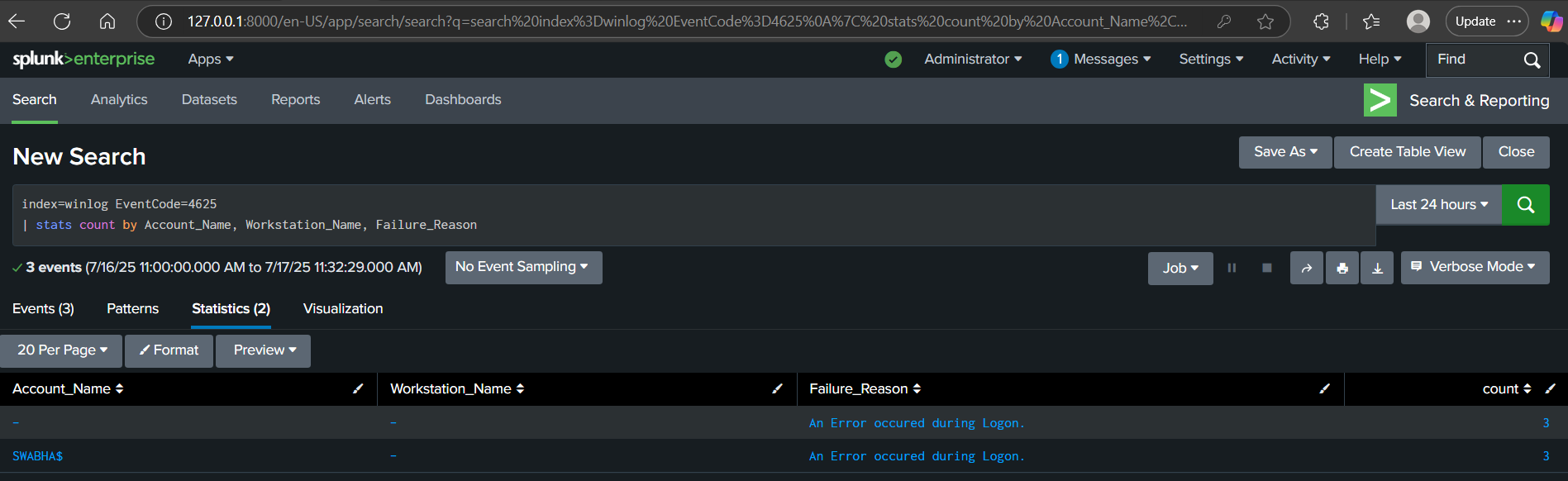
**Detect in Splunk: index=winlog EventCode=4625**

**| stats count by Account\_Name, Workstation\_Name, Failure\_Reason**

**What to Look For:**

* High number of 4625 events for one user
* FailureReason = "Unknown user name or bad password"





## Successful Login After Several Failures

**The Threat/Technique (Brute-Force / Password Spraying and Compromise):**

* Attackers frequently attempt to gain unauthorized access by guessing credentials.
* **Brute-force attacks** involve systematically trying many passwords for a single username.
* **Password spraying attacks** involve trying one common password against many usernames.
* When multiple failed login attempts are quickly followed by a successful login for the *same account*, it's a strong indicator that one of these automated attacks (or even a persistent manual attempt) has succeeded, meaning the account might be compromised.

I verified:

1. **Simulation:** You manually simulated a brute-force or password spraying attempt by making multiple failed logins for an account, followed by a single successful login for that same account.
2. **Endpoint Visibility:** You confirmed Windows generated the appropriate **Event ID 4625 (failed)** and **Event ID 4624 (successful)** logs in the Security Event Log.
3. **Splunk Detection:** You successfully used a Splunk search query with the transaction command to correlate these individual login events into a sequence, identifying specific instances where an account successfully logged in shortly after experiencing multiple failures.

**Purpose:** Detect successful login after failed attempts (may indicate password spray success)

**Simulate:**

1. Repeat above failed logins
2. Then log in successfully

**Detect in Splunk:**

(index=winlog EventCode=4625 OR EventCode=4624)

| transaction Account\_Name maxspan=5m

| search EventCode=4625 AND EventCode=4624

| table \_time, Account\_Name, EventCode, Workstation\_Name

**What to Look For:**

* Sequence of 4625 followed closely by 4624 from same user/IP

# Category: Process Execution & Behavior

## **PowerShell with Encoded Command (Obfuscated Execution)**

**The Threat/Technique (Obfuscated Execution):**

* Attackers frequently use **obfuscation** to conceal the true nature of their commands. One very common way to do this with PowerShell is using the -EncodedCommand parameter.
* Instead of writing their malicious script directly on the command line (e.g., powershell.exe -Command "Download-Malware..."), they **Base64-encode** the entire script.
* They then execute PowerShell like this: powershell.exe -EncodedCommand <base64\_string>. PowerShell will automatically decode the string and run the hidden script. This makes the command line appear as a jumbled string of characters, making it harder to detect based on simple signatures or manual review.

**In essence, for this task, you verified:**

1. **Simulation:** You ran a PowerShell command using -EncodedCommand (powershell -EncodedCommand UwB0AGEAcgB0AC0AUwBsAGUAZQBwACAAMQAwAA==) to simulate an attacker's obfuscated execution.
2. **Endpoint Visibility:** You confirmed Sysmon accurately captured this process creation (Event ID 1), including the powershell.exe image and the entire CommandLine containing the EncodedCommand.
3. **Splunk Detection:** You successfully searched in Splunk for these specific Sysmon events, demonstrating your ability to identify PowerShell executions that are attempting to hide their true purpose through encoding.

**Simulate:**

powershell -EncodedCommand UwB0AGEAcgB0AC0AUwBsAGUAZQBwACAAMQAwAA==

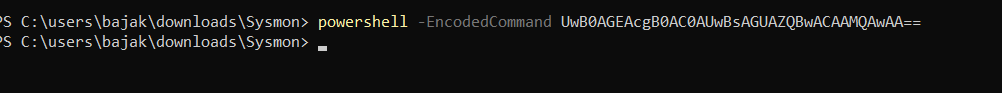
Detect in Splunk:

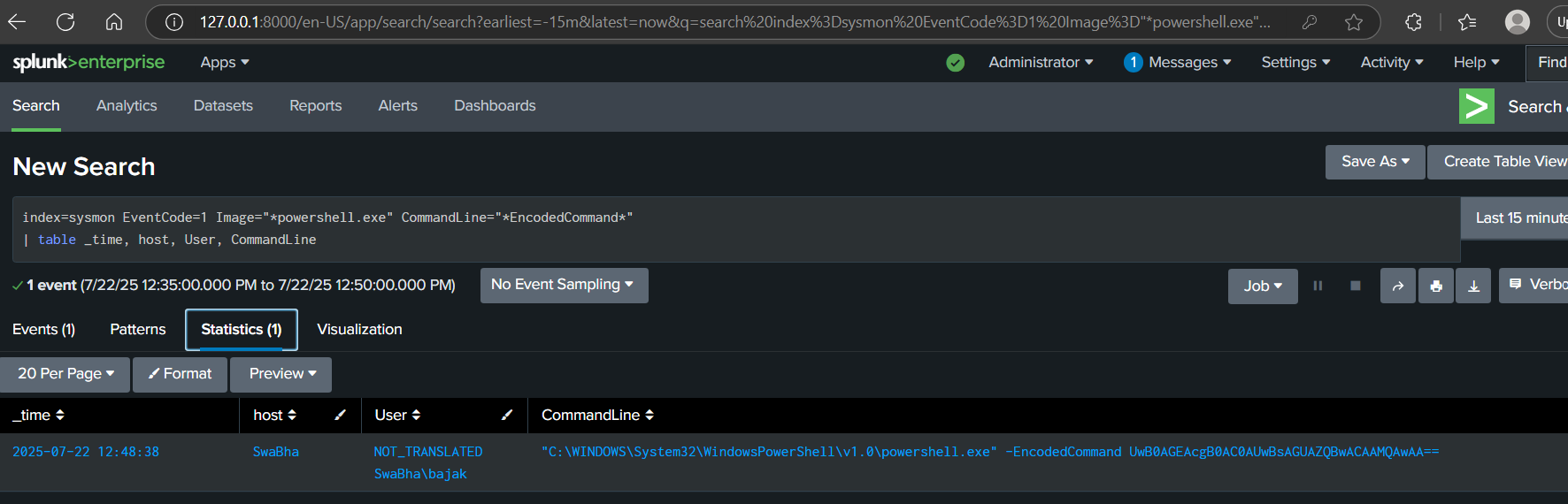
index=sysmon EventCode=1 Process\_Name="powershell.exe" CommandLine="\*EncodedCommand\*"

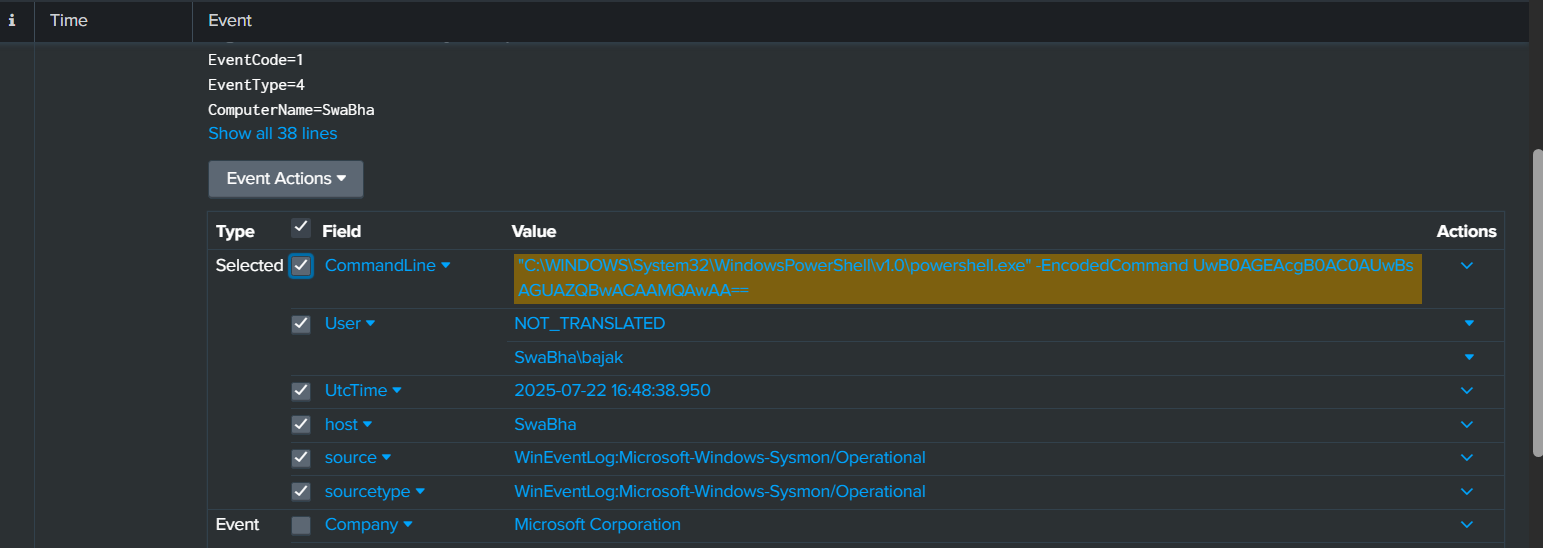
| table \_time, host, User, CommandLine

**What to Look For:**

* PowerShell running encoded commands (common in attacks)







## **Office Application Spawning PowerShell (Macro-like Behavior) and General Obfuscation**

**The Threats/Techniques (Initial Access & Evasion):**

* **Office Application Spawning PowerShell (Macro Behavior):** This is a primary initial access vector. Attackers embed malicious code (often VBA macros) within seemingly harmless Office documents (Word, Excel, PowerPoint). When an unsuspecting user opens the document and enables macros, the VBA code executes, and a frequent next step is to launch a powershell.exe process to run further malicious commands, download malware, or establish persistence. This bypasses many traditional file-based detections.
* **PowerShell with EncodedCommand (General Obfuscation):** This is an **evasion** technique. Instead of writing out their malicious PowerShell script in plain text on the command line (which security tools could easily detect), attackers encode it (e.g., using Base64). They then execute powershell.exe with the -EncodedCommand parameter, which decodes and runs the hidden script. This makes the command line appear benign or unintelligible, bypassing simple signature matches.

I **verified:**

1. **Simulation:** You effectively simulated two common attack behaviors:
   * An Office application (conceptually via a macro) launching powershell.exe.
   * The use of powershell.exe with the -EncodedCommand parameter to hide the true intentions.
2. **Endpoint Visibility:** You confirmed Sysmon accurately captured these process creations (Event ID 1), including the powershell.exe image, its parent process (if applicable, e.g., an Office app), and the full command line with the EncodedCommand.
3. **Splunk Detection:** You successfully used a Splunk search to find these specific Sysmon events, demonstrating your ability to detect both the spawning of PowerShell and the use of obfuscation techniques.

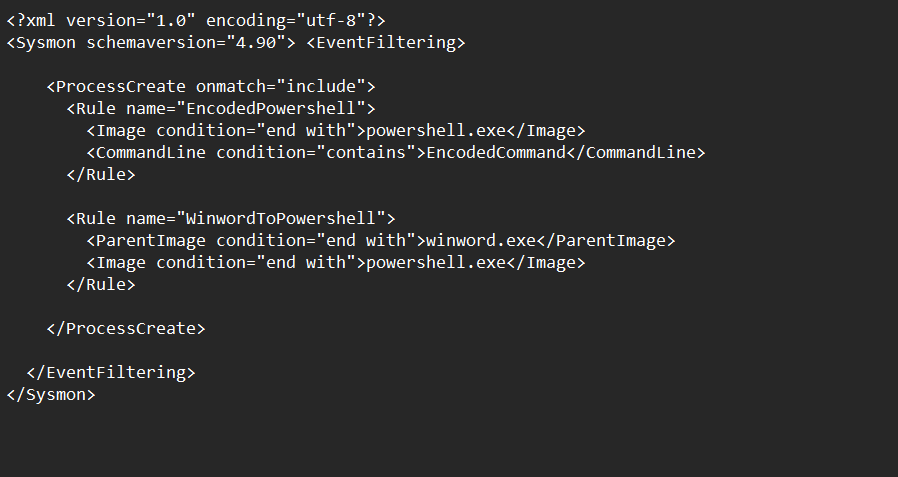
**Simulate:**

* Attack Scenario 1: Office Application Spawning PowerShell (via Macro, etc.)
* Attack Scenario 2: PowerShell with EncodedCommand (General Obfuscation)

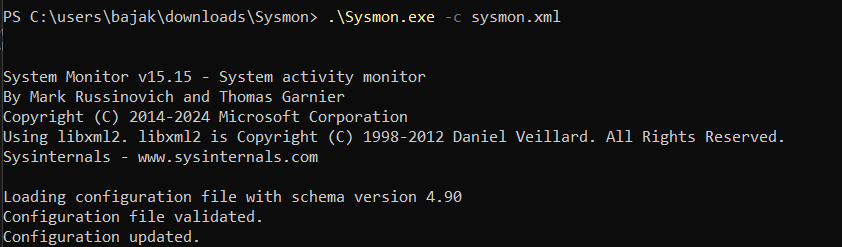
**What to Look For:**

* Word or Excel spawning a script runner — suspicious macro activity

Configure sysmon.xml file as shown

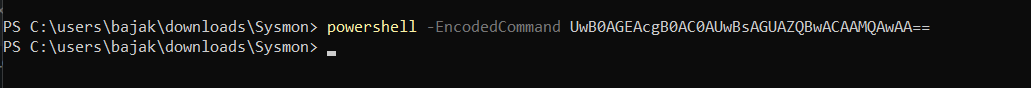


Run the below command to update the Sysmon.xml file



Detect in Splunk:

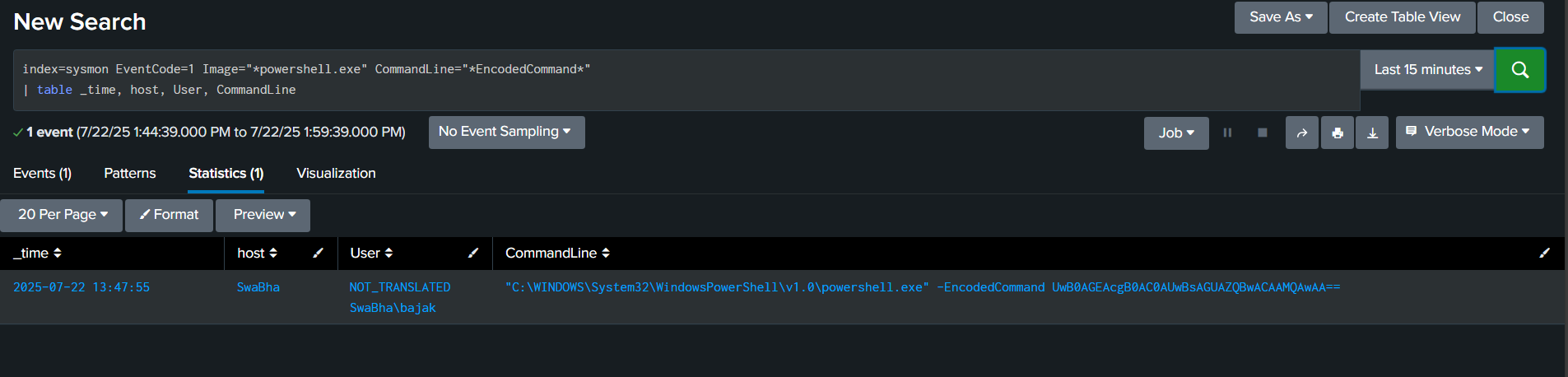
Run the following command as an admin in powershell

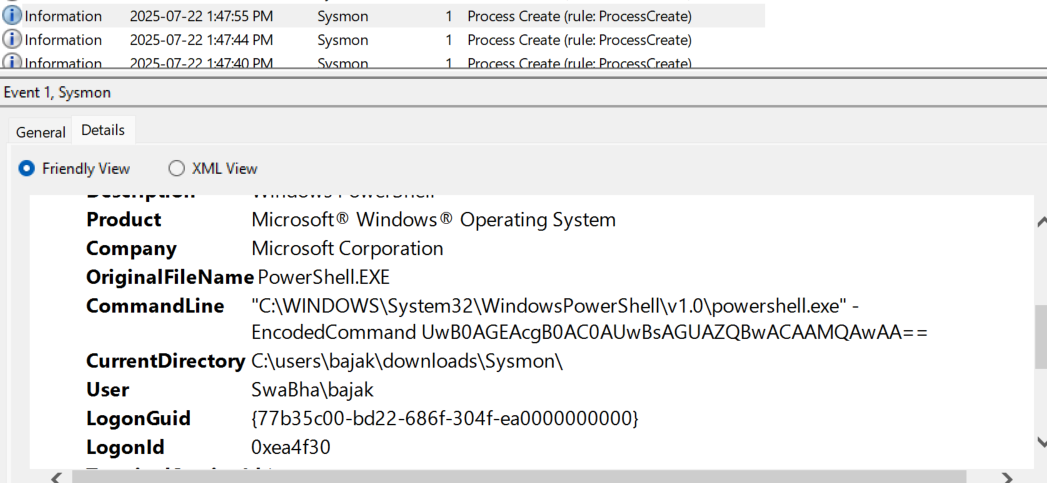


Run the search for the "EncodedCommand" event:

index=sysmon EventCode=1 Image="\*powershell.exe" CommandLine="\*EncodedCommand\*"

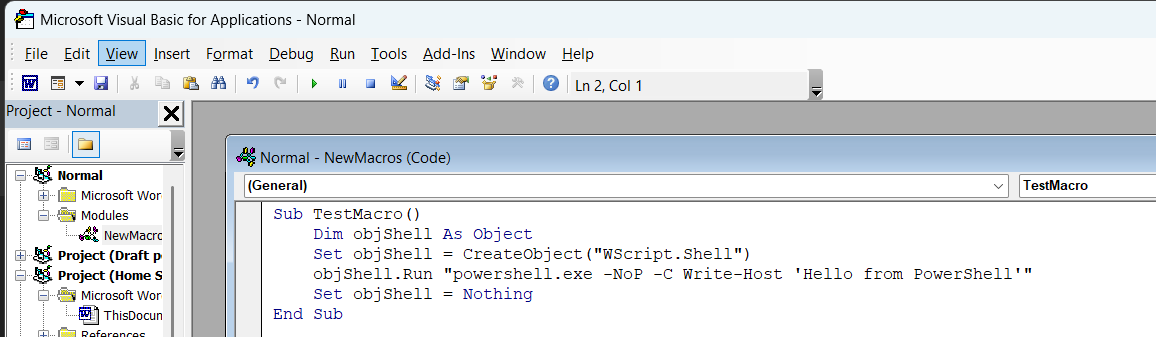
| table \_time, host, User, CommandLine





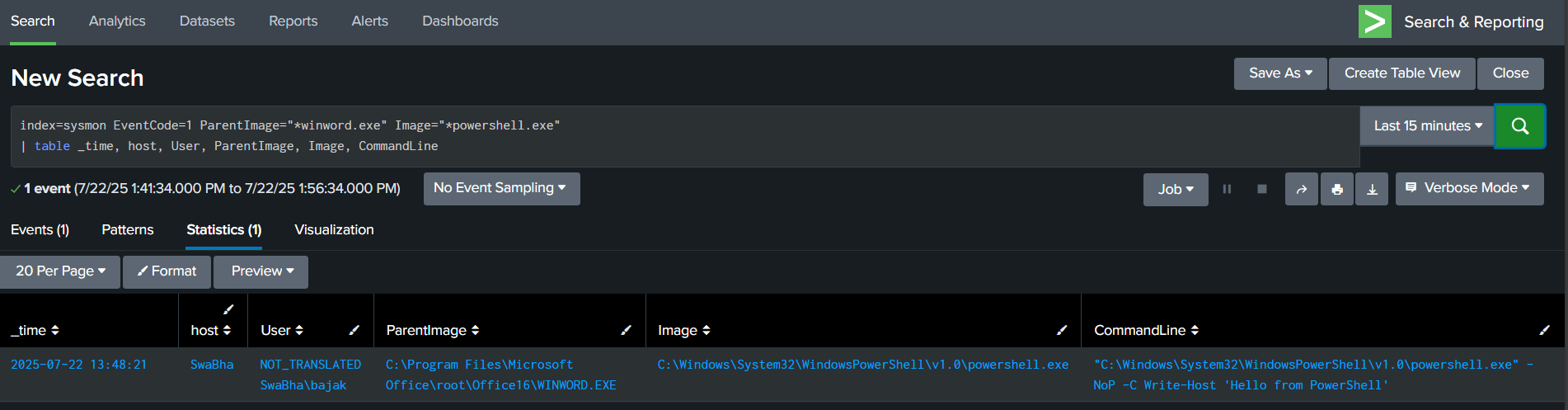
Run the search for the "Word Macro" event:

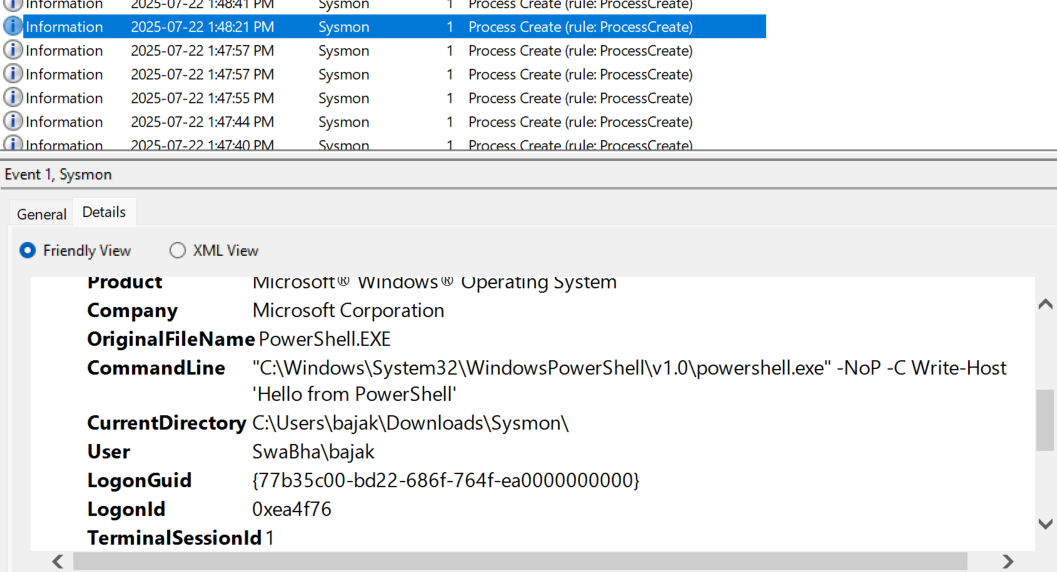
Create a Macro in word document as below and save it as .docm file



index=sysmon EventCode=1 ParentImage="\*winword.exe" Image="\*powershell.exe"

| table \_time, host, User, ParentImage, Image, CommandLine





# Category: Network Activity

## PowerShell Making External Network Connections

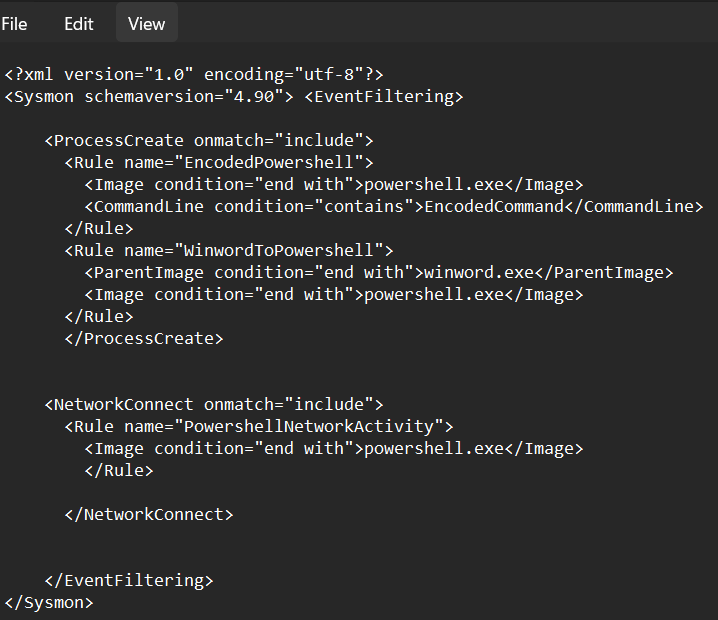
I verified**:**

1. **Simulation:** You directly used the Invoke-WebRequest cmdlet in PowerShell to make an outbound connection to a simulated external server (http://1.1.1.1).
2. **Sysmon Logging:** You confirmed that Sysmon correctly generated an **Event ID 3 (Network Connection)** log, accurately showing powershell.exe as the source and the external IP as the destination.
3. **Splunk Detection with Context:** You successfully searched in Splunk for these EventCode=3 events by powershell.exe, and importantly, you were able to extract and display the CommandLine field to verify the exact Invoke-WebRequest command that was executed.

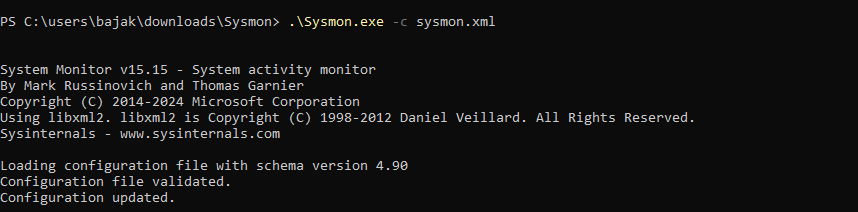
**The Threat/Technique (Direct Malicious Outbound Communications):**

* As discussed, attackers need to establish outbound network connections for **Command and Control (C2)**, **data exfiltration**, or **downloading additional malicious tools/payloads**.
* **Invoke-WebRequest** is a powerful, built-in PowerShell cmdlet designed for making HTTP/HTTPS requests (like downloading files or interacting with web APIs). Its direct use by attackers is a common tactic because it's native to the operating system and often not flagged by traditional antivirus software.

Configure Sysmon.xml file

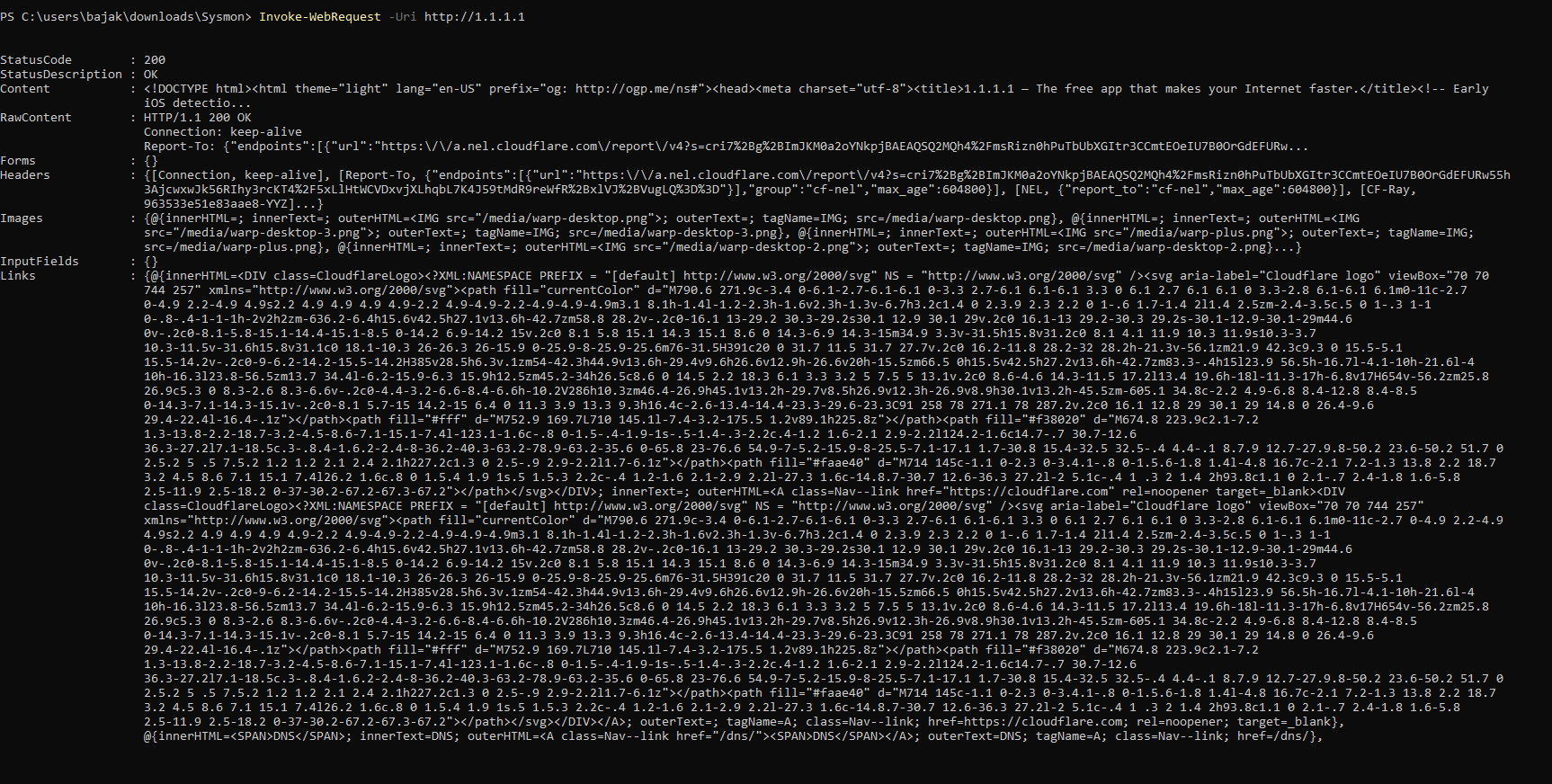


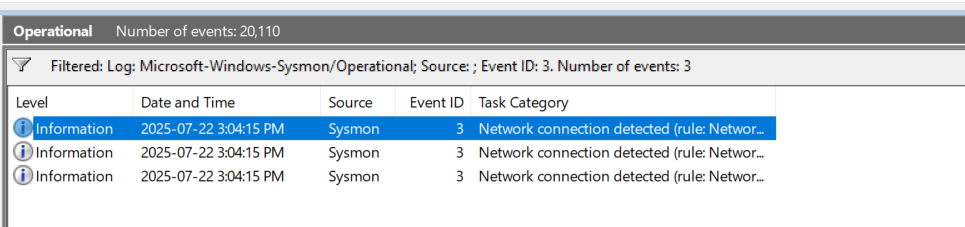
Run the following command to update the configuration file after changes



**Simulate:**

Invoke-WebRequest -Uri <http://1.1.1.1>





Detect in Splunk:

index=sysmon EventCode=3 Image="\*powershell.exe"

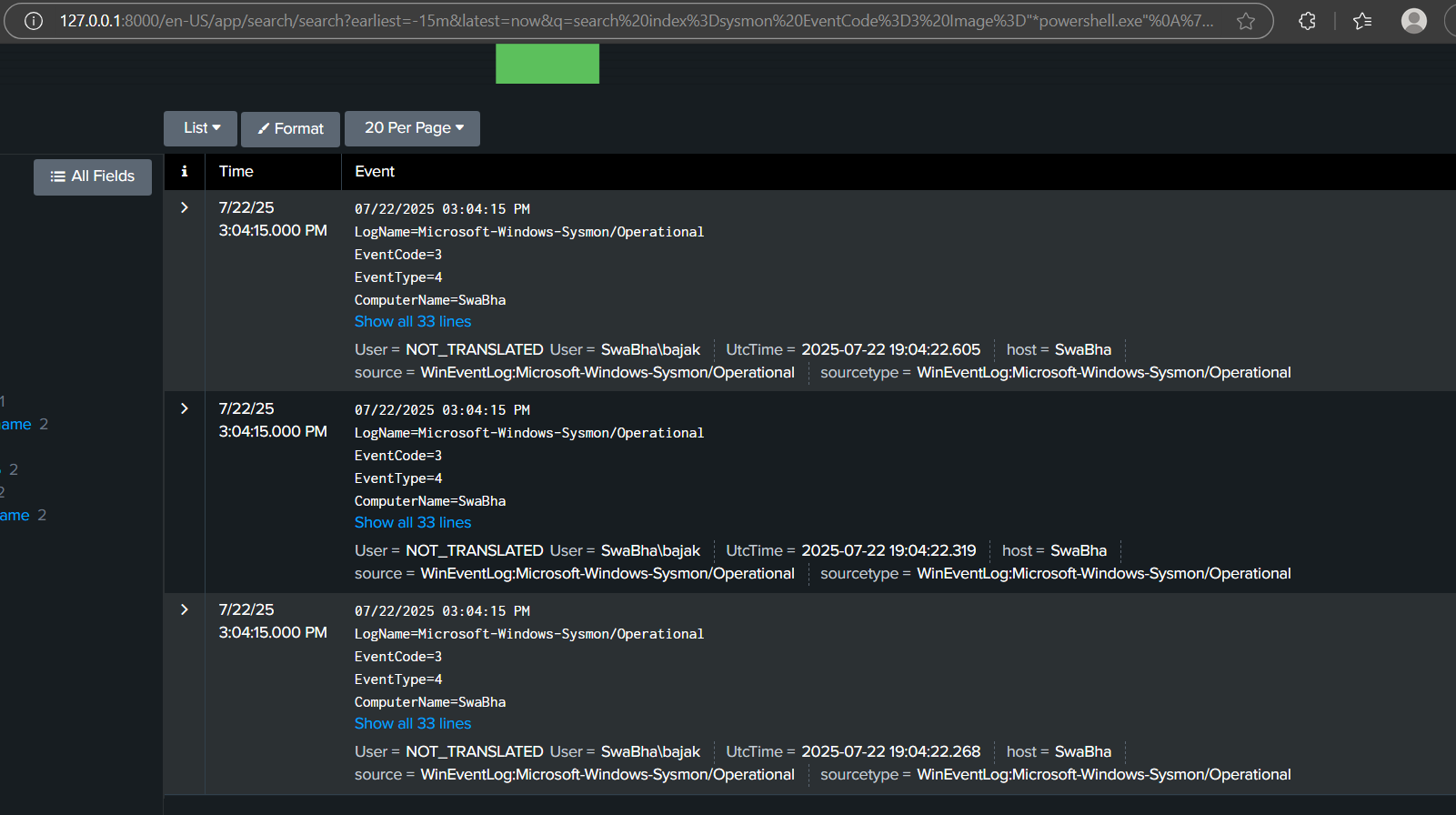
| table \_time, Image, DestinationIp, DestinationPort, CommandLine

**What to Look For:**

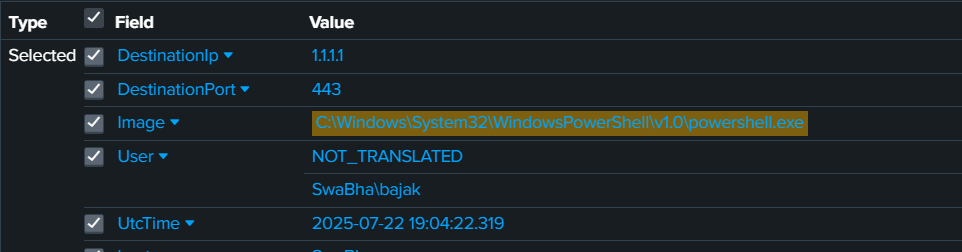
* Non-browser processes connecting to the internet
* Unknown or unusual IP addresses

you are looking for a log entry that clearly shows powershell.exe (in the Image field) connecting to 1.1.1.1 (in DestinationIp) on port 443 (in DestinationPort) at the correct time.

when you ran Invoke-WebRequest -Uri http://1.1.1.1, the PowerShell command ultimately made a connection over **HTTPS (port 443)**, not HTTP (port 80).







## Sysmon Event ID 3 (NetworkConnect) Verification for PowerShell Outbound Connection.

I **verified:**

1. **Simulation:** You used curl http://1.1.1.1 in PowerShell to simulate an outbound network connection, mimicking malicious C2 or data exfiltration.
2. **Endpoint Visibility:** You confirmed Sysmon correctly detected this outbound connection and logged it as an **Event ID 3 (Network Connection)** in the Windows Event Viewer, showing powershell.exe as the source and 1.1.1.1 as the destination.
3. **Splunk Detection:** You successfully used a Splunk search to find these specific network connection events, proving your ability to detect suspicious outbound communications initiated by PowerShell.

**The Threat/Technique (Malicious Outbound Network Connections):**

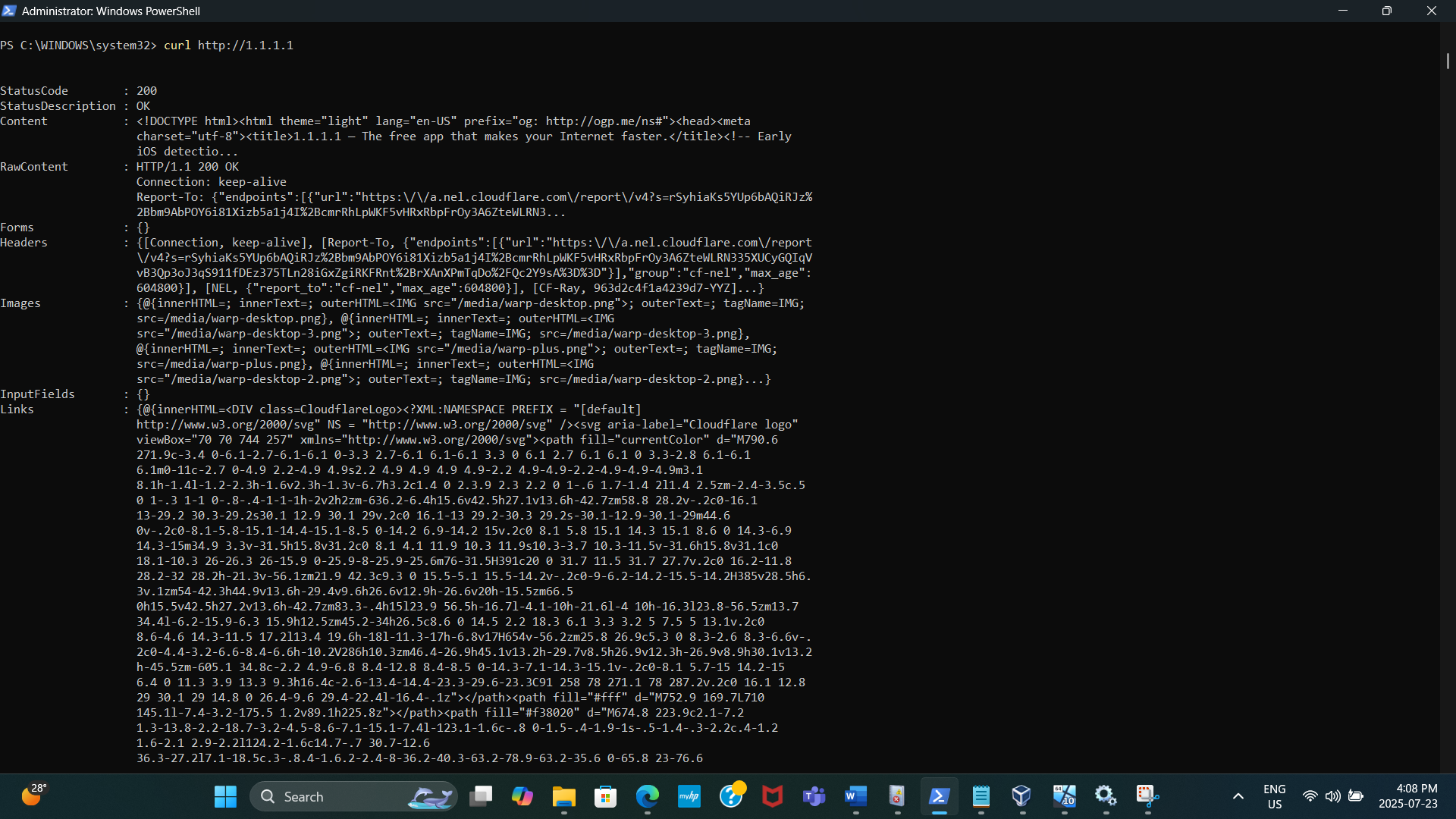
* Attackers, after gaining initial access, nearly always need to communicate with external servers for various reasons:
  + **Command and Control (C2):** To receive further instructions from the attacker.
  + **Data Exfiltration:** To send stolen data out of your network.
  + **Payload Delivery:** To download additional malware or tools.
* **PowerShell** is a favored tool for this. It's built into Windows, often trusted by security tools, and highly capable of making network requests (like curl which is an alias for Invoke-WebRequest or Invoke-RestMethod in PowerShell). This makes it a powerful and stealthy way for attackers to establish connections.

Simulate:

curl <http://1.1.1.1>

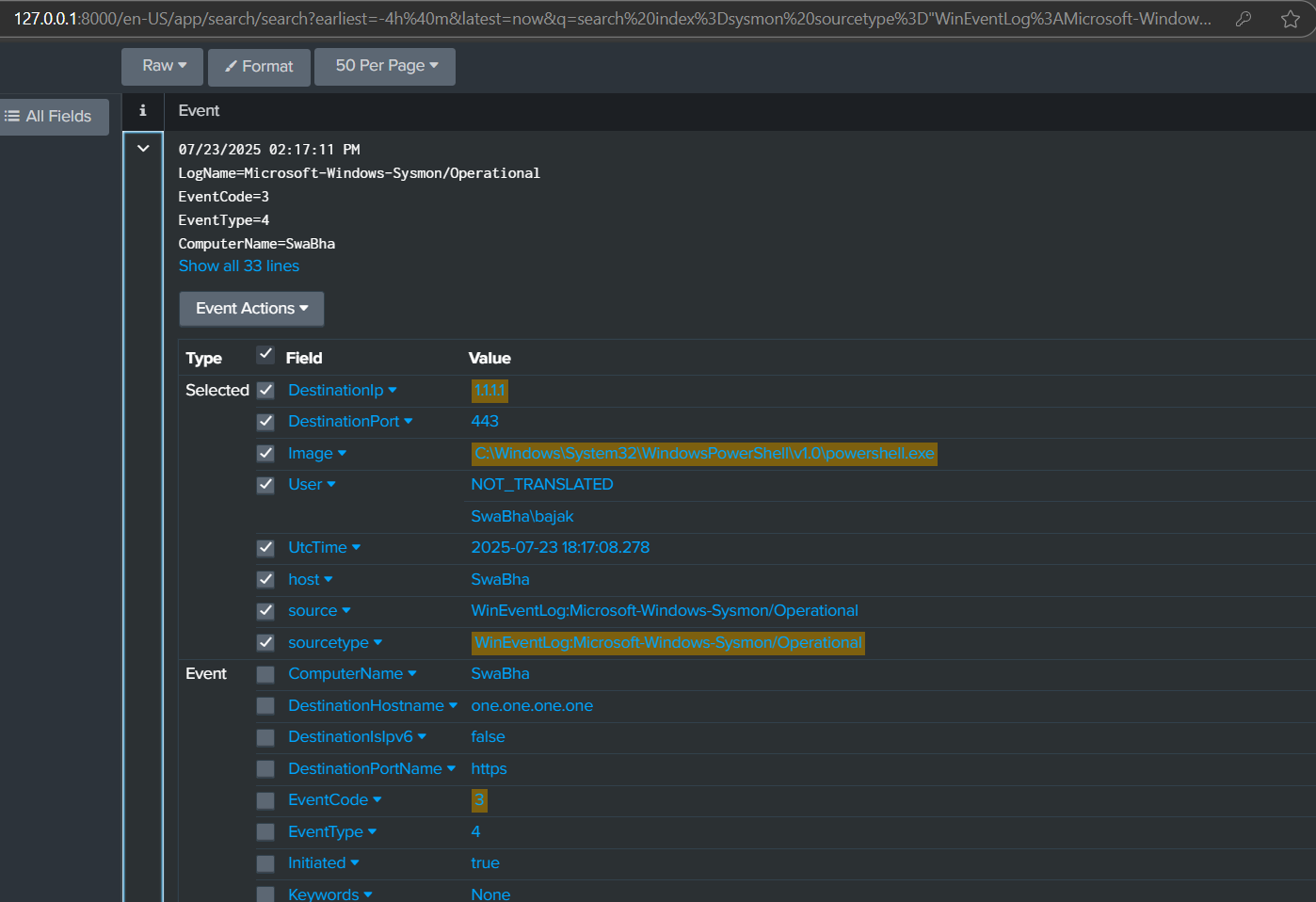
**1.1.1.1 (Cloudflare Public DNS):**

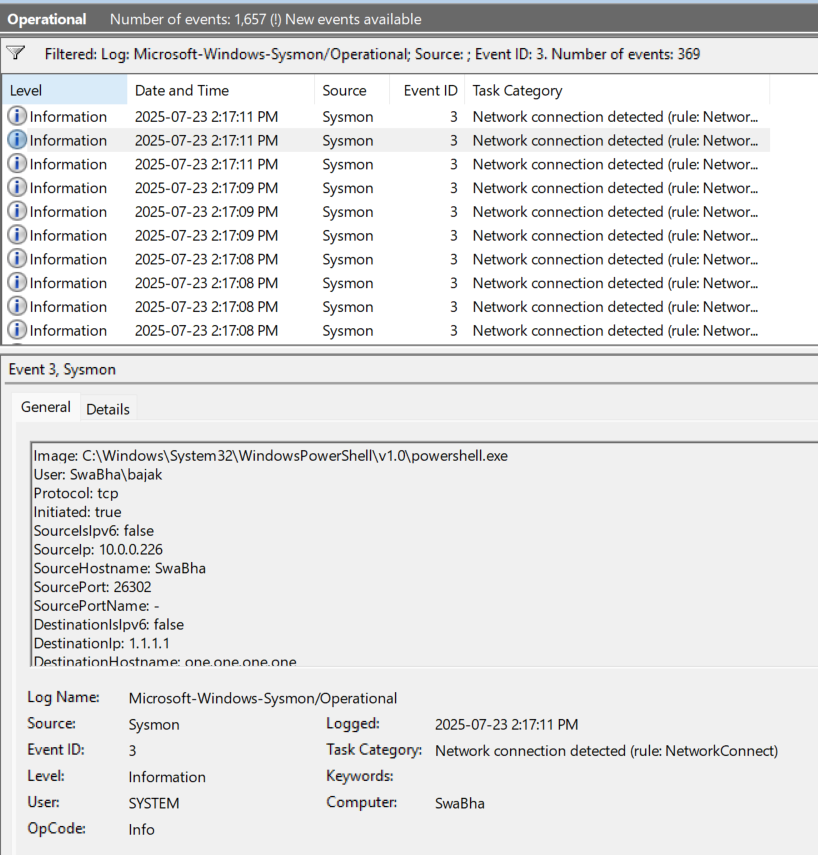
* This is a **public, globally routable, live IP address**.
* It is specifically used by Cloudflare for their public DNS resolver service.
* When you try to connect to 1.1.1.1 (e.g., via ping or Invoke-WebRequest), you are attempting to connect to an **active, responsive server** on the internet. Your connection attempt will typically succeed (for ping if ICMP is allowed, or for HTTP/HTTPS if the service is running).



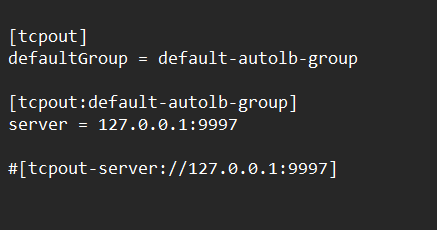
**Detect in Splunk:**

index=sysmon sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=3 Image="\*powershell.exe" DestinationIp="1.1.1.1"

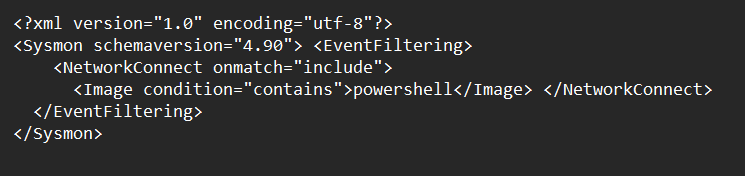




Output.conf file configuration



Sysmon.xml file configuration



# Category: Persistence & System Changes

## New Service Created on the System

**I verified:**

1. **Simulation:** You used sc.exe to simulate an attacker creating a new service (TestBackdoor pointing to calc.exe).
2. **Sysmon Logging:** You confirmed that Sysmon correctly generated a **Process Creation event (Event ID 1)** whenever sc.exe was run.
3. **Splunk Detection:** You then successfully used a Splunk search to find these EventCode=1 events where sc.exe was the Image, thus proving your ability to detect the execution of this service creation command.

**The Threat/Technique (Persistence via New Service):**

* Attackers often achieve **persistence** by creating new Windows services. Services are programs that run in the background, often starting automatically when the system boots, even before any user logs in.
* This is a highly effective way for malware or backdoors to ensure they execute reliably.
* The sc.exe utility is a legitimate Windows command-line tool used to create, start, stop, or query services. Attackers frequently abuse sc.exe to install their malicious services.

Simulate

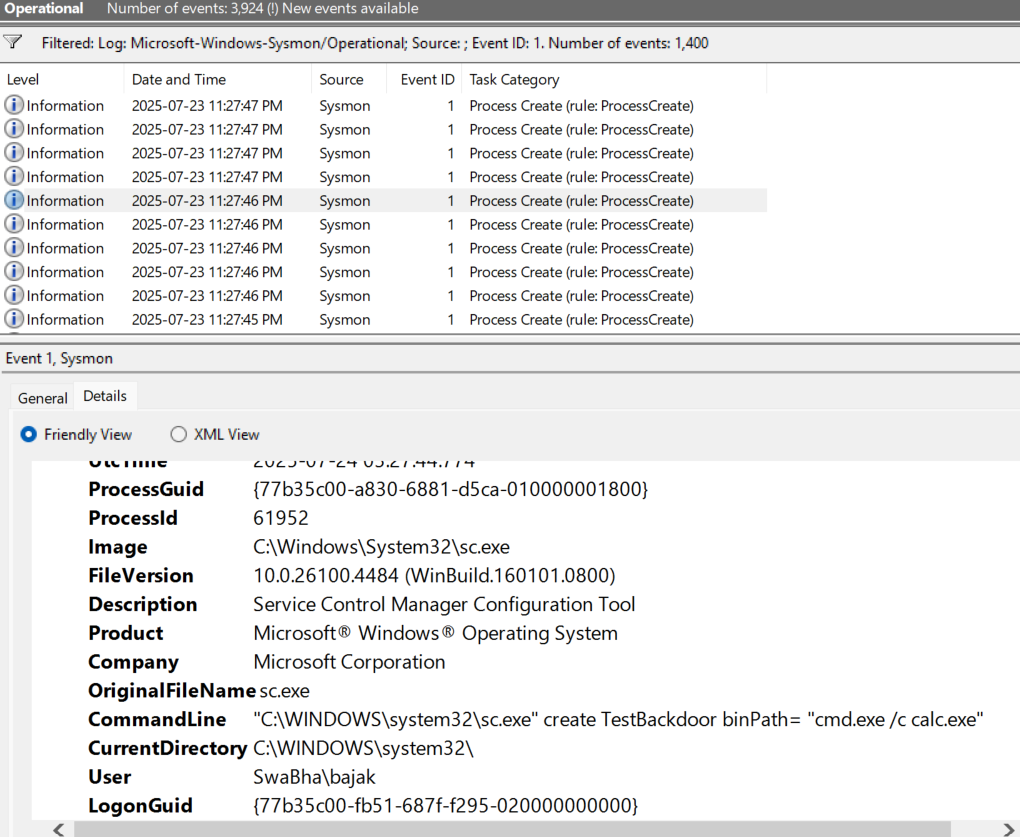
sc.exe create TestBackdoor binPath= "cmd.exe /c calc.exe"

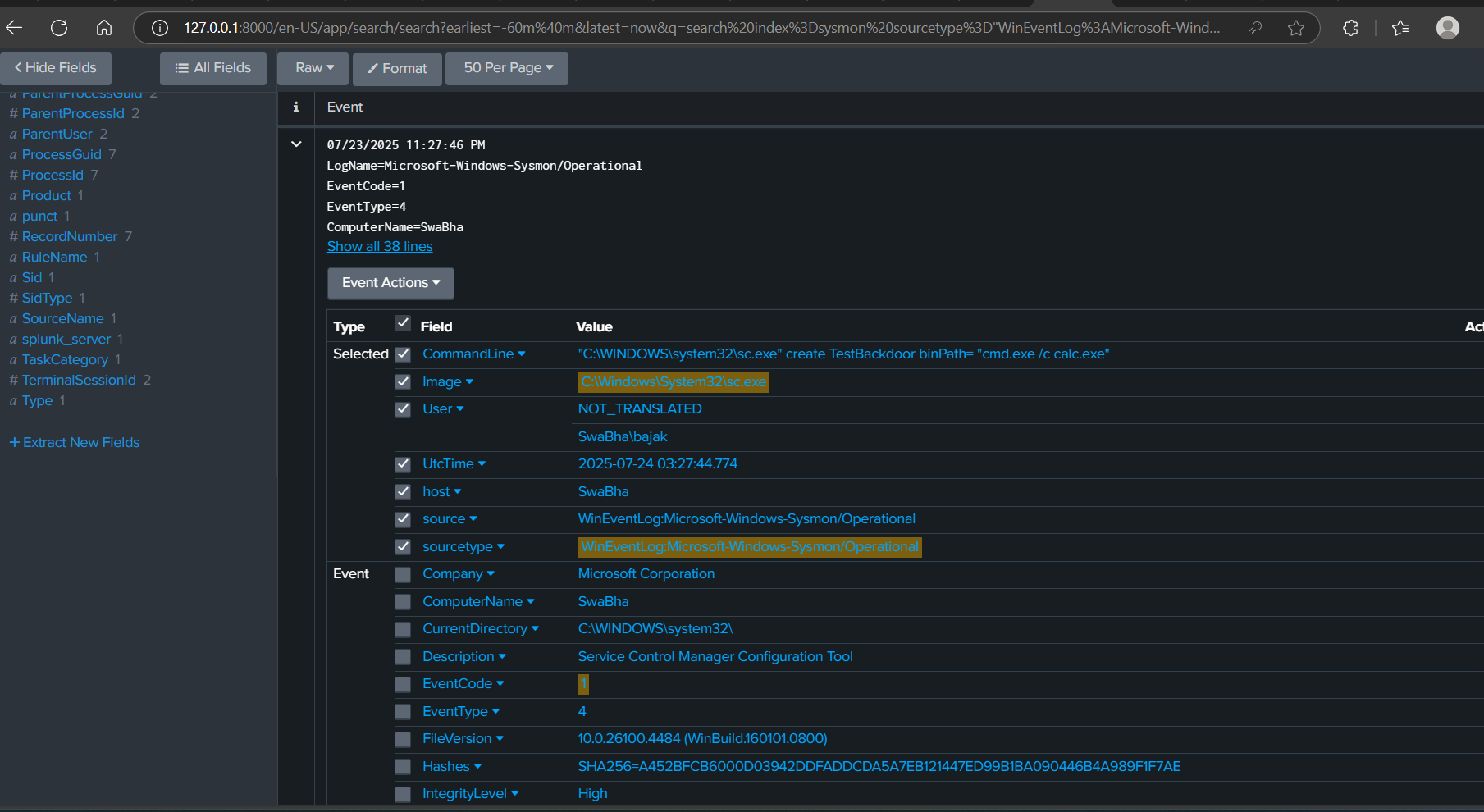
detected in Splunk:

index=sysmon sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 Image="\*sc.exe"

**What to Look For:**

* New service with suspicious path (e.g., running cmd.exe, powershell.exe)





## **Registry Run Key Modification (Persistence)**

I verified the entire chain of detection for this persistence technique:

1. **Simulated Attack:** You manually triggered a registry run key modification (like an attacker would).
2. **Endpoint Visibility:** You confirmed Sysmon saw this activity and logged it correctly into the Windows Event Viewer.
3. **Log Ingestion:** You ensured Splunk Enterprise was properly configured (by placing the inputs.conf in the correct location and restarting) to pull these live Sysmon events from the Windows Event Log.
4. **Search & Detection:** You then successfully searched in Splunk for those specific Sysmon events (Event IDs 12 or 13, targeting the Run key) to prove that Splunk could detect this activity.

**The Threat/Technique (Persistence via Run Keys):**

* You were verifying a **"persistence"** technique. Persistence is when an attacker establishes a way to re-access a system even after a reboot or a user logs off/on.
* One very common way to achieve this is by modifying **"Run" registry keys** (like HKCU\Software\Microsoft\Windows\CurrentVersion\Run). Programs listed in these keys automatically launch when a user logs in. Attackers use this to ensure their malware or backdoors run every time the user starts their session.

**Simulate:**

New-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run" -Name "Malware" -Value "calc.exe"

Detect in Splunk

index=sysmon sourcetype="WinEventLog:Microsoft-Windows-Sysmon/Operational" (EventCode=12 OR EventCode=13)

| table \_time, host, EventCode, EventType, Image, CommandLine, TargetObject, NewValue, User

| sort -\_time

**What to Look For:**

* New entries in Run keys — often used for startup persistence

