# 

# **Detecting PowerShell Exploitation Using Wazuh**

## **Introduction to PowerShell Monitoring and Detection Using Wazuh**

**What Is Achieved?**

PowerShell is a built-in command-line shell and scripting language in Windows. It’s commonly used by system administrators for automating tasks and managing systems. However, it’s also often misused by attackers because of its powerful capabilities and deep access to system components.

Here are some common ways attackers exploit PowerShell:

Running malware directly in memory, allowing them to avoid detection by traditional antivirus software.  
Downloading malicious files or scripts from the internet using commands like Invoke-WebRequest or Invoke-WebClient.  
Hiding malicious actions using Base64-encoded scripts, which makes it difficult for security tools to analyze the commands.  
Stealing passwords and credentials from memory using tools like Mimikatz.  
Performing internal network reconnaissance and privilege escalation using tools like SharpHound.

### **Mimikatz**

Mimikatz is a well-known post-exploitation tool used to extract plaintext passwords, password hashes, PIN codes, and Kerberos tickets directly from system memory. Attackers commonly use it for credential dumping, allowing them to gain unauthorized access to accounts and services.

**Key Features of Mimikatz:**

Extracts plaintext passwords from memory  
Dumps NTLM and Kerberos hashes  
Performs Pass-the-Hash and Pass-the-Ticket attacks  
Exploits weaknesses in Windows authentication mechanisms  
Often used through PowerShell to stay memory-resident and undetected

### **SharpHound**

SharpHound is the data collection component of the BloodHound toolset. It is used to map and analyze relationships and permissions within Active Directory (AD). Attackers use it to find the most efficient paths to escalate privileges or compromise high-value accounts.

**Key Features of SharpHound:**

Collects data on user privileges, group memberships, sessions, and trust relationships  
Helps attackers identify attack paths to escalate privileges  
Can be executed via PowerShell to remain stealthy and avoid detection  
Useful for planning lateral movement within the network  
Helps answer the question: “If I compromise this low-level user, how can I reach a Domain Admin?”

### **Why Monitoring PowerShell with Wazuh Matters**

By enabling advanced PowerShell logging and monitoring it through Wazuh, you gain:

Visibility into all script block executions, even if the commands are encoded or obfuscated  
Awareness of PowerShell module and command usage, helping identify unusual or unauthorized activities  
Detection of known attack tools like Mimikatz and SharpHound based on behavior or command patterns  
Real-time alerts integrated with your SIEM, enabling quick investigation and response  
Stronger security posture by identifying suspicious activity early in the attack chain

## **Environment Overview**

Windows Server 2022 (running Wazuh agent) — the target endpoint to be monitored.

Kali Linux (running Wazuh manager) — the central analysis point where logs are gathered, rules are applied, and alerts are generated.

Network connectivity is required between the agent and manager for log forwarding.

Wazuh agent installed on Windows Server (usually in C:\Program Files (x86)\ossec-agent\).

Wazuh manager installed and operational on Kali Linux.

## **Steps to Enable PowerShell Monitoring and Configure Wazuh**

## **Enable PowerShell Advanced Logging on Windows Server 2022**

PowerShell does not log detailed script contents by default for performance reasons. To detect exploitation, the following logging must be enabled:

Script Block Logging (Event ID 4104): Captures executed script content, including encoded scripts.

Module Logging (Event ID 4103): Logs pipeline executions and module function calls.

Operational PowerShell Event Channel: Collects these event logs for monitoring.

## **Enable via PowerShell Admin:**

Run this script to enable both Script Block and Module logging:

# Enable Script Block Logging

New-Item -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging" -Force | Out-Null

Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging" -Name "EnableScriptBlockLogging" -Value 1

# Enable Module Logging

New-Item -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Force | Out-Null

Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Name "EnableModuleLogging" -Value 1

Set-ItemProperty -Path "HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Name "ModuleNames" -Value @("\*") -Type MultiString

Write-Output "PowerShell Script Block and Module Logging Enabled"

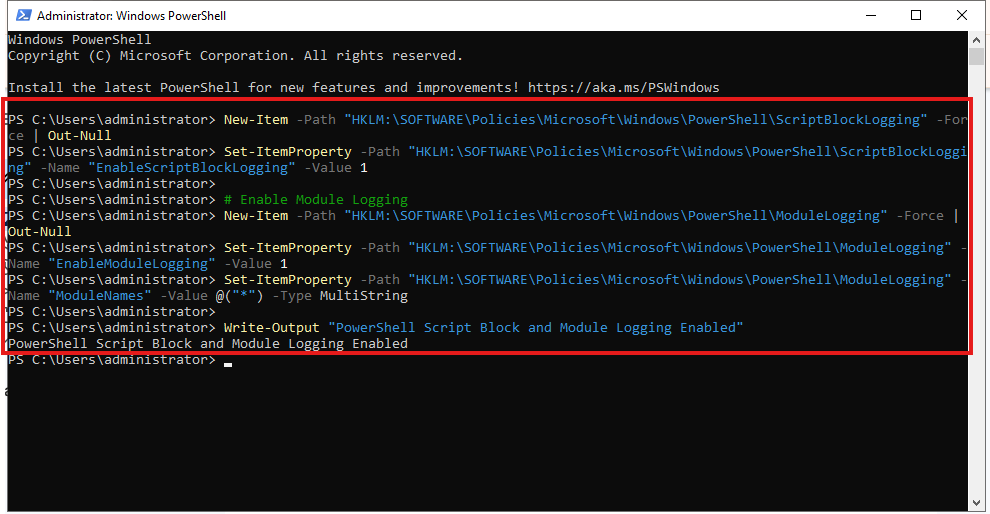
## **Explanation:**

These registry keys are the equivalent of Group Policy settings that turn on deep PowerShell logging.

EnableScriptBlockLogging captures the entire script blocks executed.

EnableModuleLogging tracks command execution by module, giving visibility into used cmdlets.

Setting "ModuleNames" to "\*" captures all modules.



**Configure Wazuh Agent to Collect PowerShell Event Logs**

Open the Wazuh agent configuration file on Windows (default path):

notepad "C:\Program Files (x86)\ossec-agent\ossec.conf"

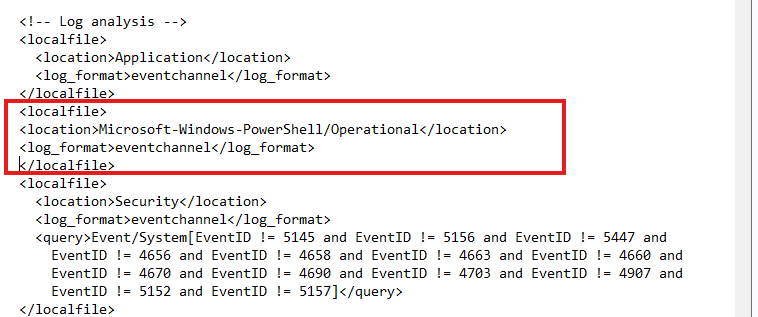
Add the following inside <ossec\_config> to instruct the agent to read the PowerShell Operational event channel:

<localfile>

<location>Microsoft-Windows-PowerShell/Operational</location>

<log\_format>eventchannel</log\_format>

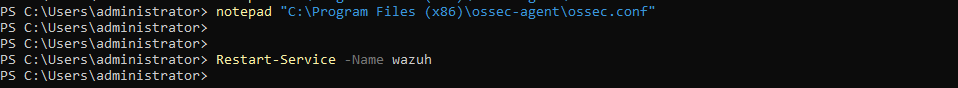
</localfile>



Save and close the file.

Restart the Wazuh agent service for changes to take effect:

Restart-Service -Name wazuh



## **Configure Wazuh Manager to Detect PowerShell Exploitation Patterns**

The manager uses rules to identify suspicious or malicious activities from the collected logs.

Edit the local rules file on your Kali Linux Wazuh manager:

sudo nano /var/ossec/etc/rules/local\_rules.xml

Insert the following comprehensive rules for PowerShell exploitation detection:

<group name="windows,powershell">

<!-- 100201: Encoded PowerShell command detected -->

<rule id="100201" level="8">

<if\_sid>60009</if\_sid>

<field name="win.eventdata.payload" type="pcre2">(?i)CommandInvocation</field>

<field name="win.system.message" type="pcre2">(?i)EncodedCommand|FromBase64String|EncodedArguments|-e\b|-enco\b|-en\b</field>

<description>Encoded command executed via PowerShell. 360 ForTress Multi Layer Cyber Protection System</description>

<mitre>

<id>T1059.001</id>

<id>T1562.001</id>

</mitre>

</rule>

<!-- 100202: PowerShell blocked by antivirus -->

<rule id="100202" level="4">

<if\_sid>60009</if\_sid>

<field name="win.system.message" type="pcre2">(?i)blocked by your antivirus software</field>

<description>Windows Security blocked malicious command executed via PowerShell.</description>

<mitre>

<id>T1059.001</id>

</mitre>

</rule>

<!-- 100203: Malicious cmdlet like Invoke-Mimikatz detected -->

<rule id="100203" level="10">

<if\_sid>60009</if\_sid>

<field name="win.eventdata.payload" type="pcre2">(?i)CommandInvocation</field>

<field name="win.system.message" type="pcre2">(?i)Add-Persistence|Find-AVSignature|Invoke-Mimikatz|Invoke-Shellcode|Set-MasterBootRecord</field>

<description>Risky CMDLet executed. Possible malicious activity detected. 360 ForTress Multi Layer Cyber Protection System</description>

<mitre>

<id>T1059.001</id>

</mitre>

</rule>

<!-- 100204: mshta used for suspicious download -->

<rule id="100204" level="8">

<if\_sid>91802</if\_sid>

<field name="win.eventdata.scriptBlockText" type="pcre2">(?i)mshta.\*GetObject|mshta.\*new ActiveXObject</field>

<description>Mshta used to download a file. Possible malicious activity detected. 360 ForTress Multi Layer Cyber Protection System</description>

<mitre>

<id>T1059.001</id>

</mitre>

</rule>

<!-- 100205: Execution policy set to bypass -->

<rule id="100205" level="5">

<if\_sid>60009</if\_sid>

<field name="win.eventdata.contextInfo" type="pcre2">(?i)ExecutionPolicy\s\*bypass|exec\s\*bypass</field>

<description>PowerShell execution policy set to bypass.</description>

<mitre>

<id>T1059.001</id>

</mitre>

</rule>

<!-- 100206: Invoke-WebRequest or IWR used -->

<rule id="100206" level="5">

<if\_sid>60009</if\_sid>

<field name="win.eventdata.contextInfo" type="pcre2">(?i)Invoke-WebRequest|IWR.\*-url|IWR.\*-InFile</field>

<description>Invoke-WebRequest executed, possible download cradle detected.</description>

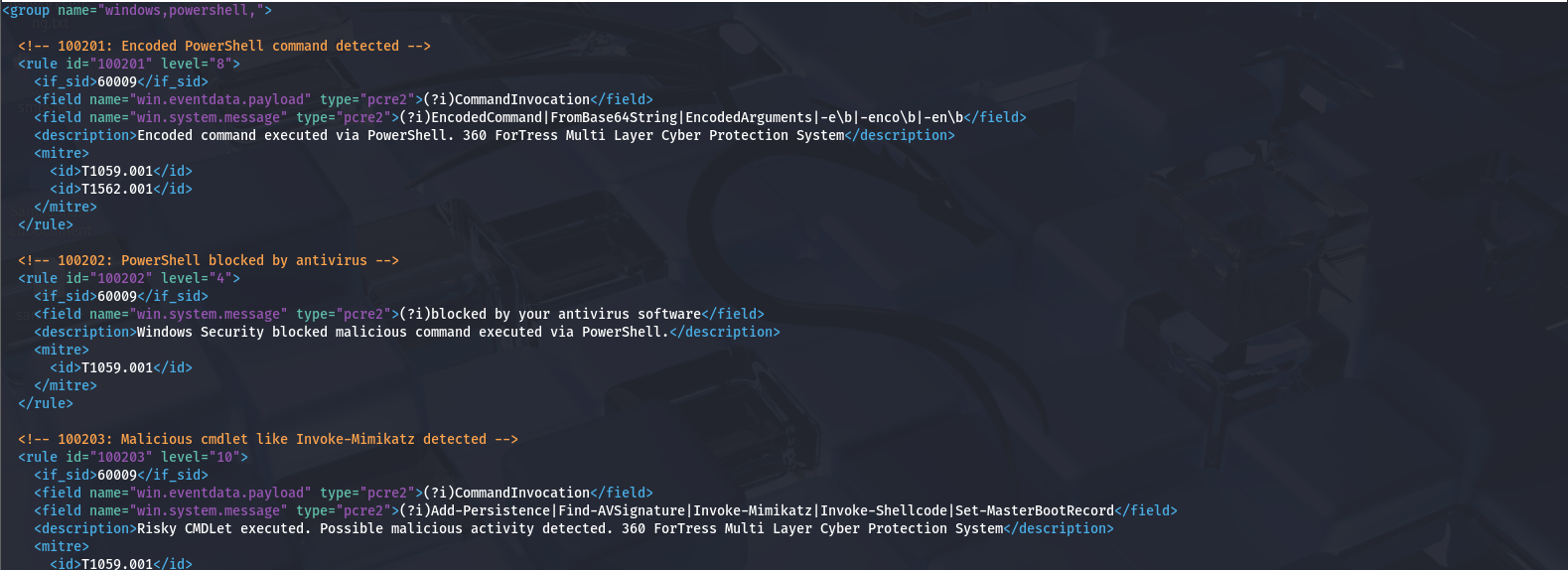
<mitre>

<id>T1059.001</id>

</mitre>

</rule>

</group>



Save and exit.

Restart Wazuh manager to apply rules:

sudo systemctl restart wazuh-manager

# **Testing PowerShell Exploitation Detection Rules in Wazuh**

## **Test 1: Trigger Detection for Invoke-WebRequest Usage (Rule ID 100206)**

Invoke-WebRequest -Uri <http://example.com>

**What This Command Does:**

This PowerShell command sends an HTTP GET request to the URL http://example.com and downloads the webpage content.

It is a simple, safe command often used by administrators for legitimate purposes such as downloading files or checking web resources.

However, attackers commonly misuse Invoke-WebRequest or its alias IWR to download malicious payloads or scripts from external sources during an attack — a technique often called a "download cradle."

**Why This Test is Important:**

The Wazuh detection rule 100206 is designed to match any PowerShell log entries where Invoke-WebRequest or IWR is used.

Detecting such commands helps security teams identify potentially malicious download activity early.

Even though this test uses a safe URL, it verifies that your monitoring setup will correctly capture and alert on this class of activity.

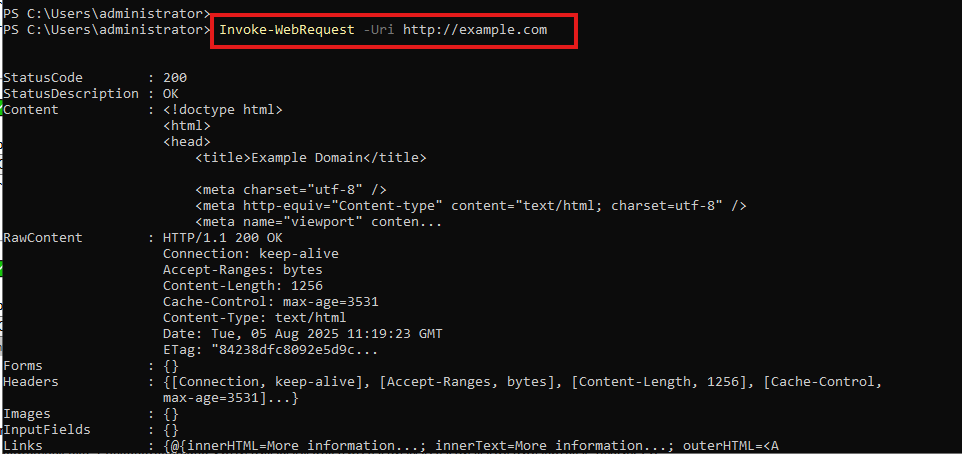
This test ensures your detection rule behaves as expected, generating alerts based on PowerShell’s downloaded web requests, enabling you to distinguish between normal and suspicious behavior with further analysis.

**Expected Outcome:**

After running this command on your monitored Windows Server, Wazuh agent collects the PowerShell Operational logs related to Invoke-WebRequest.

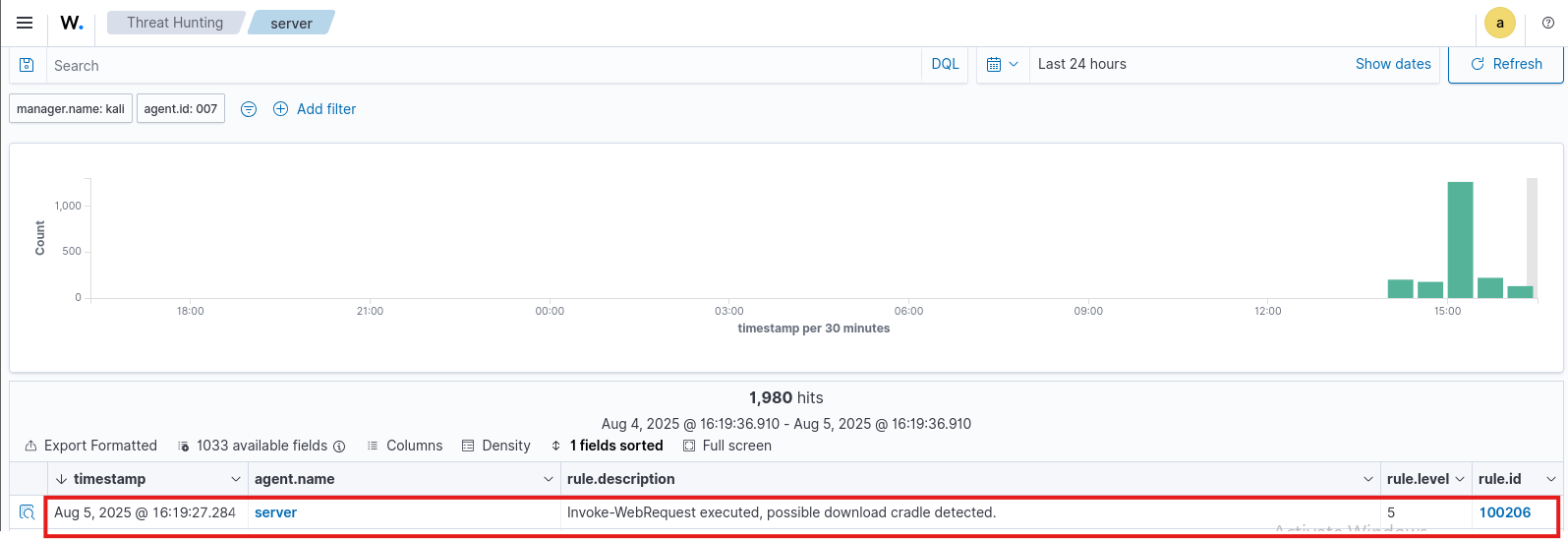
The Wazuh manager applies rule 100206, triggering an alert such as:

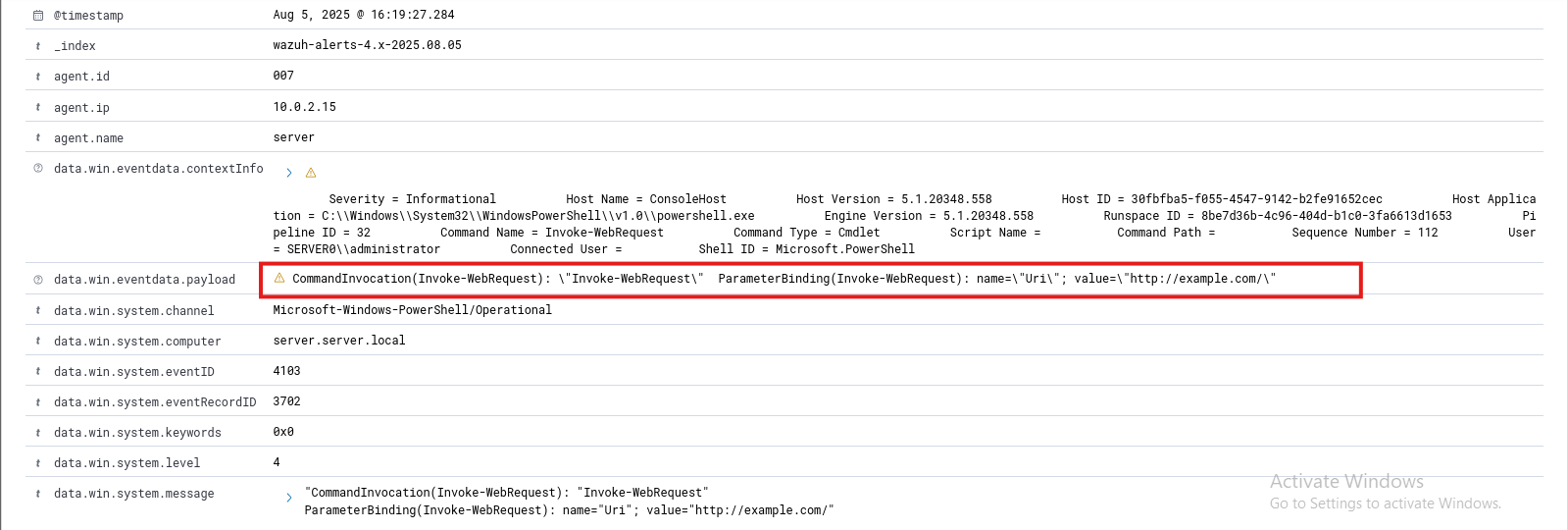
“ Invoke-WebRequest executed, possible download cradle detected. “



Now, Open the Wazuh Dashboard and navigate to Security Events or Threat Detection → Command Monitoring (depending on your configuration).

Confirm that the alert for the PowerShell encoded command (Invoke-WebRequest download cradle) is visible in the logs.







**Test 2: Trigger Detection for EICAR File Download and Execution (Rule ID 100206 & related)**

Invoke-WebRequest https://secure.eicar.org/eicar.com.txt -OutFile eicar; .\eicar

## Explanation:

Invoke-WebRequest downloads the file from the specified URL (https://secure.eicar.org/eicar.com.txt) and saves it locally as eicar.

After downloading, .\eicar attempts to execute the downloaded file.

The EICAR file is a harmless test file designed to trigger antivirus and security detection systems.

Executing this command tests whether your antivirus and Wazuh setup can detect and alert on suspicious download and execution activity triggered by PowerShell.

Your Wazuh rules that detect PowerShell download cradles (Invoke-WebRequest) and execution of suspicious files should generate alerts.

This simulates how attackers use PowerShell to download and execute malicious payloads from the internet.

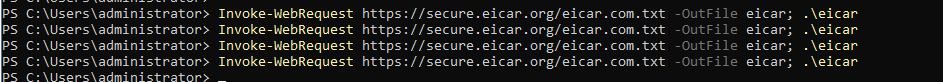
**Expected Behavior:**

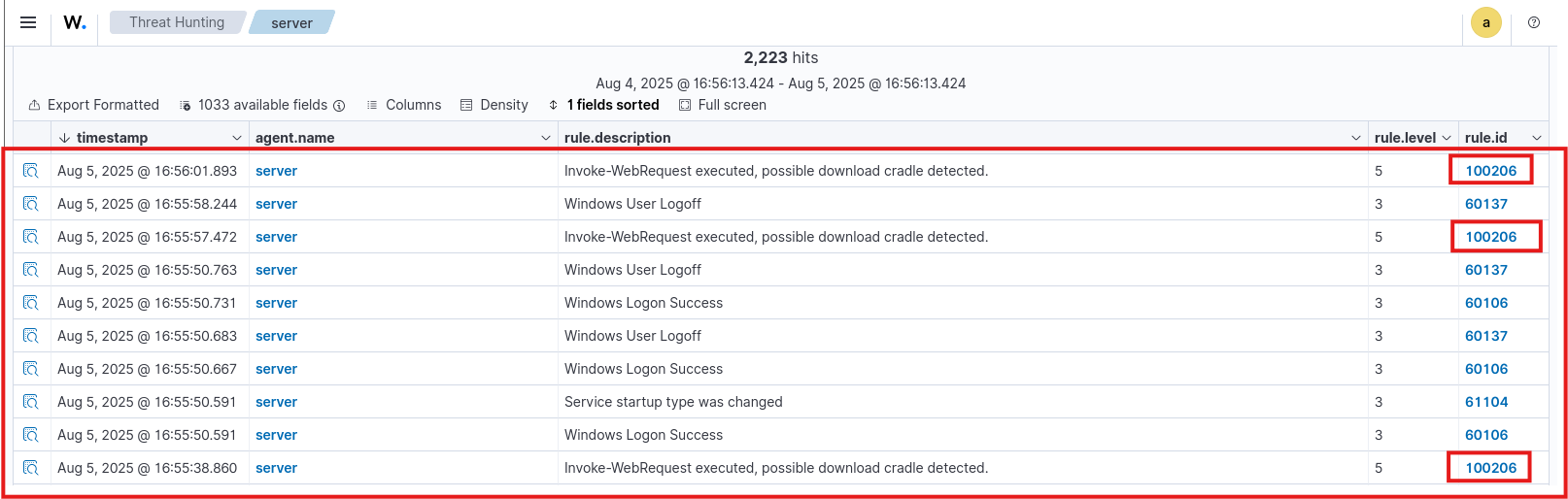
Antivirus should detect and block the EICAR file download or execution.

Wazuh captures the PowerShell event logs generated.

Alerts fire in the Wazuh dashboard indicating detection of suspicious PowerShell download and execution, verifying your monitoring setup.

This test is safe, effective, and widely used for validating endpoint protection and detection capabilities.





## **Test 3: Verify PowerShell Execution Policy Bypass Detection (Rule ID 100205)**

powershell.exe -ExecutionPolicy Bypass -NoProfile -Command "Get-Process"

**Explanation:**

This command runs PowerShell with the -ExecutionPolicy Bypass option, temporarily disabling any script execution restrictions.

Bypassing the execution policy is a common attacker technique used to run unsigned or unauthorized scripts that would otherwise be blocked.

The -NoProfile parameter avoids loading user profiles, making the execution faster and less detectable.

The actual PowerShell command run here is simple (Get-Process) and harmless; the focus is on triggering the policy bypass detection.

Your Wazuh detection rules looking for ExecutionPolicy bypass in PowerShell commands will generate alerts when this is run.

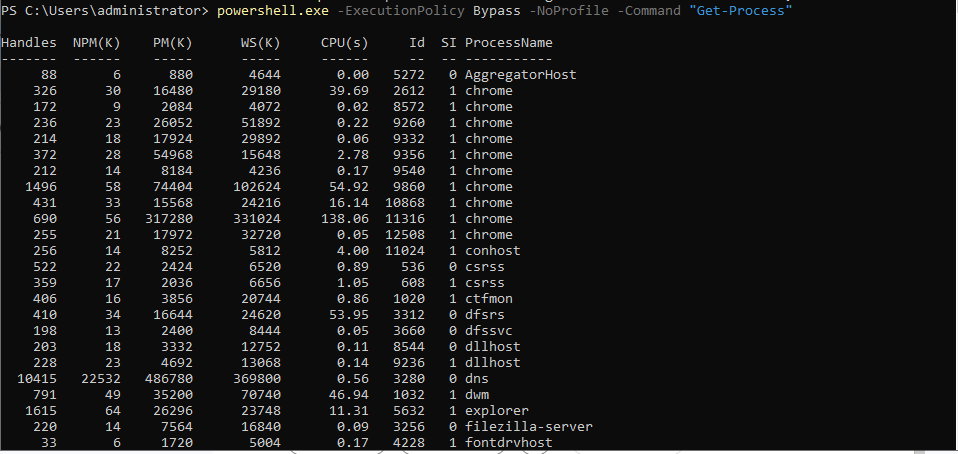
This test confirms that your monitoring system detects attempts to circumvent execution restrictions, which is critical for spotting potential malicious script activity.

**Expected Outcome:**

Wazuh agent collects the PowerShell operational logs containing the policy bypass usage.

An alert from rule ID 100205 appears in your Wazuh dashboard stating PowerShell execution policy set to bypass.

This validates that your detection mechanisms effectively catch this risky behavior.



To monitor PowerShell execution policy bypass alerts (Rule ID 100205) in the Wazuh dashboard:

Log into Wazuh and select your Windows Server 2022 agent from the "Agents" list.

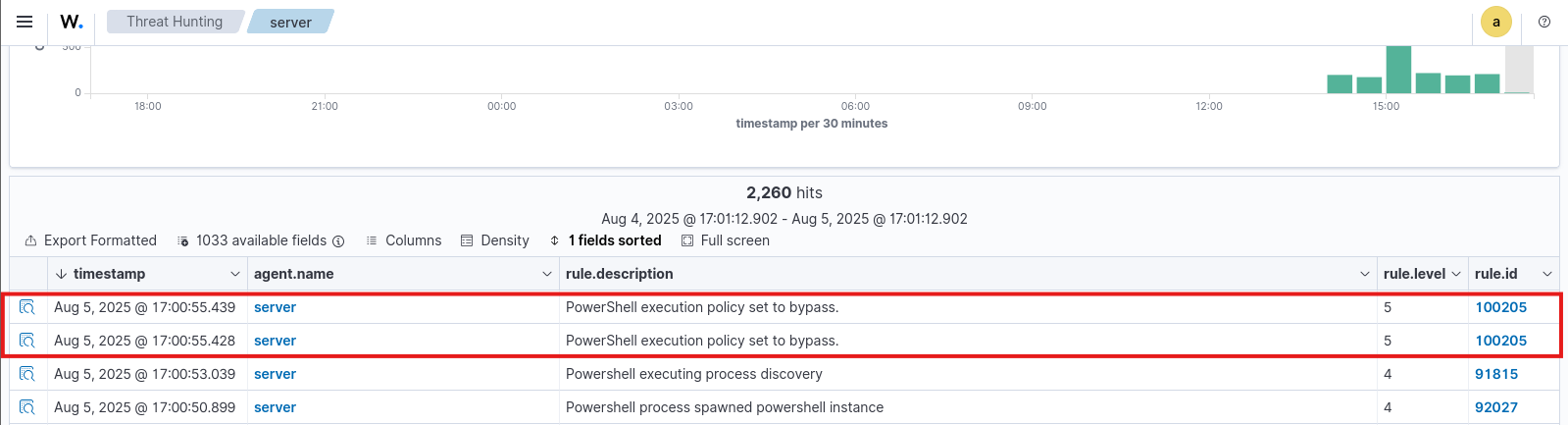
Go to the "Threat Hunting" or "Events" tab to view security events from the agent.

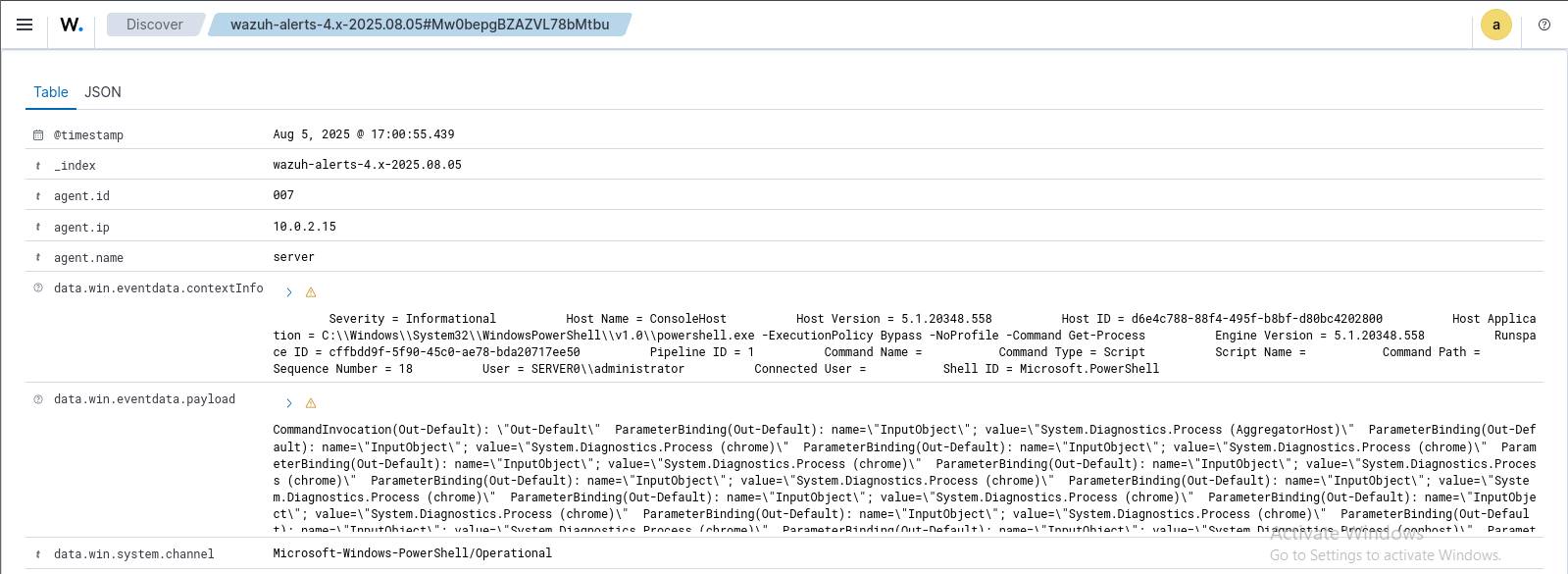
Use the search box to filter by rule ID 100205 or keywords like "execution policy bypass."

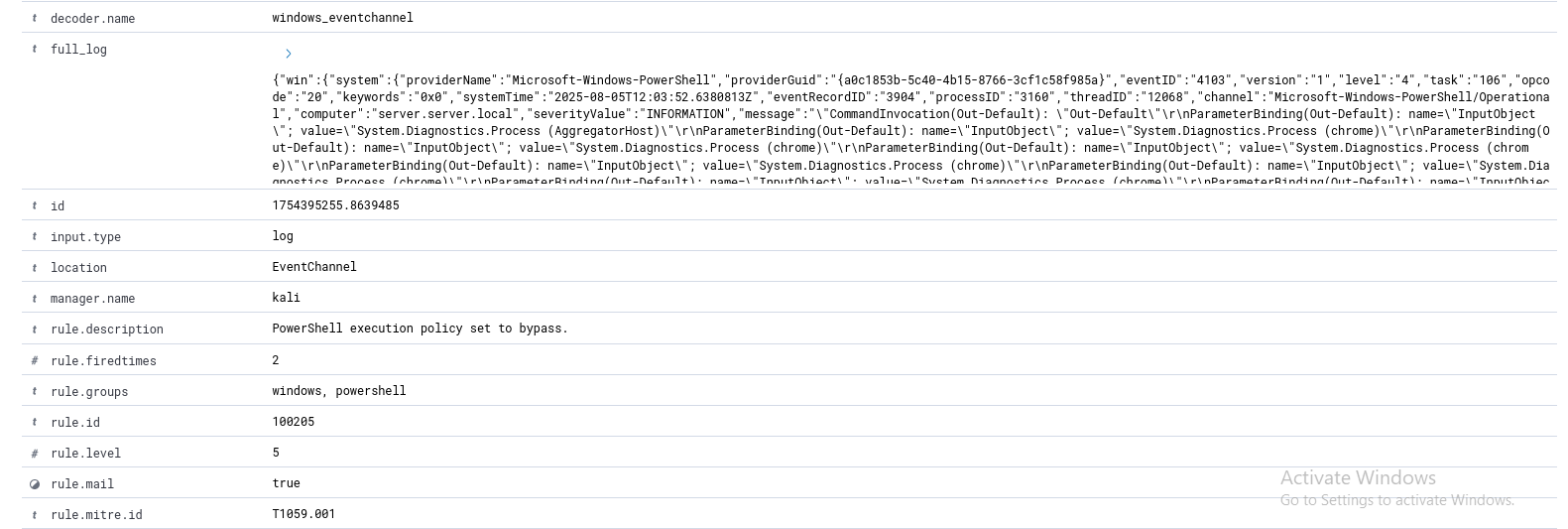
Click an alert to see details—time, command executed, severity, and related MITRE technique.

These alerts indicate PowerShell was run with policy bypass enabled, signaling potentially risky or malicious activity.

Regular review helps you quickly detect and respond to threats utilizing PowerShell bypass techniques.







**Summary:**

In this project, we set up Wazuh to watch and detect harmful PowerShell activities on a Windows Server 2022. PowerShell is a strong tool used by administrators, but attackers also misuse it to run bad commands, download harmful files, or steal passwords using tools like Mimikatz and SharpHound. We turned on detailed logging on the Windows server so every PowerShell command and script, even secret or encoded ones, get recorded. The Wazuh agent on the server collects these logs and safely sends them to the Wazuh manager on Kali Linux. The manager uses special rules to check the logs and find suspicious actions like encoded commands, downloads, or bypassing security rules. When such activity is detected, alerts are created and shown on the Wazuh dashboard, making it easy for security teams to find and investigate threats. We tested the system with real-like attacks, such as downloading a harmless test file (EICAR) and bypassing PowerShell restrictions, and Wazuh successfully caught and alerted on them. This setup helps organizations quickly spot and respond to PowerShell attacks, turning raw data into clear alerts to keep systems safe from advanced threats.