Scenario 2: PowerShell Suspicious Web Request 9/19/2025

Explanation:

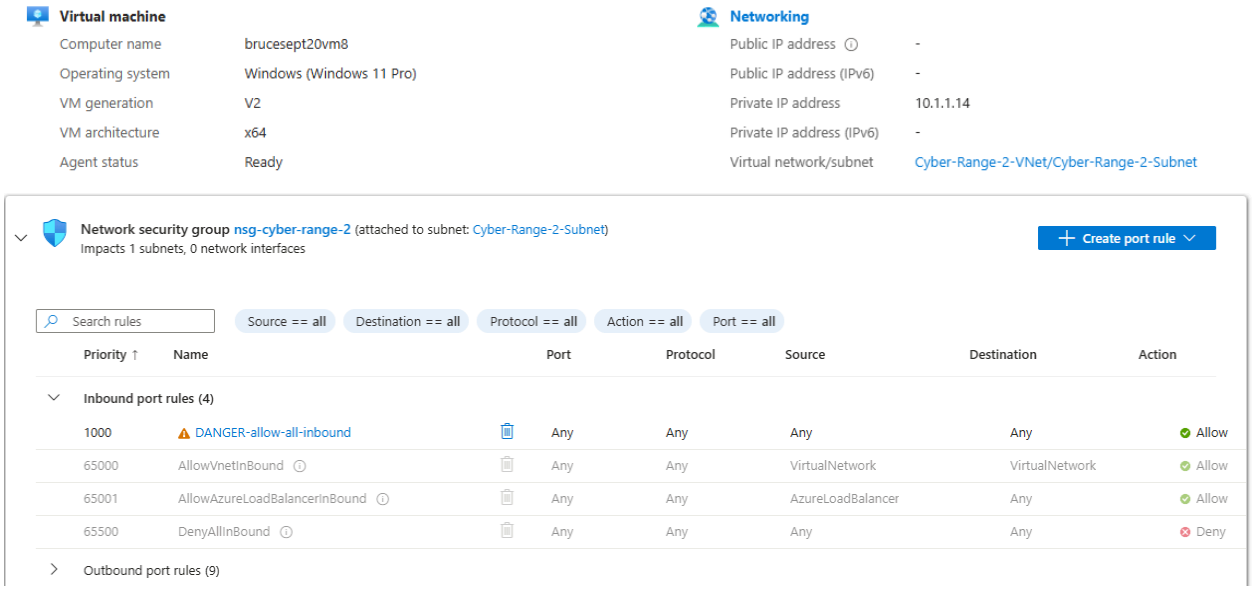
Sometimes when a bad actor has access to a system, they will attempt to download malicious payloads or tools directly from the internet to expand their control or establish persistence. This is often achieved using legitimate system utilities like PowerShell to blend in with normal activity. By leveraging 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.

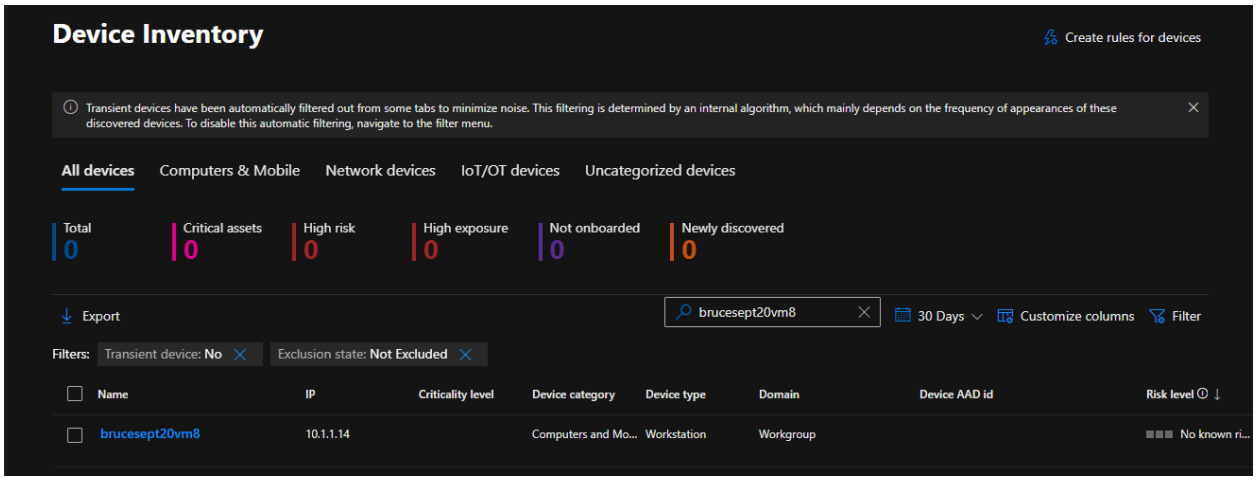
Preliminary Steps:

- 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: brucesept20vm8
Virtual Machine Operating System: Windows 11
Screenshots of Virtual Machine and Network Security Group settings:



Onboarded to Microsoft Defender for Endpoint (MDE) Screenshot:



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:
BruceSuspicious PowerShell Download via Invoke-WebRequest

Alert Rule Created: 9/20/2025 12:03:34 PM

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 == "brucesept20vm8"
| 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 “brucesept20vm8” 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:

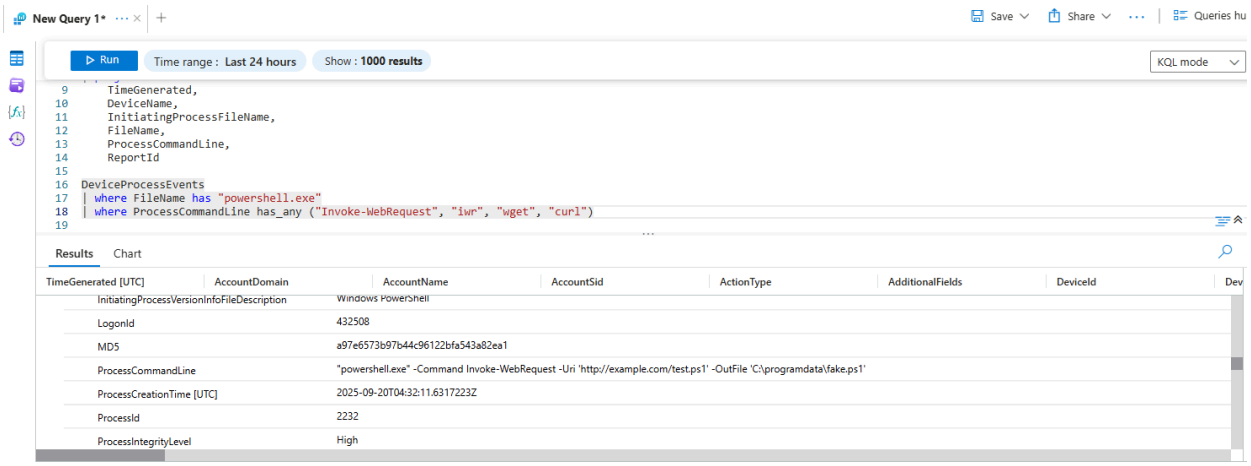
- This will generate 5 log entries in a few seconds, giving us multiple alerts to demonstrate our Sentinel rule in action.

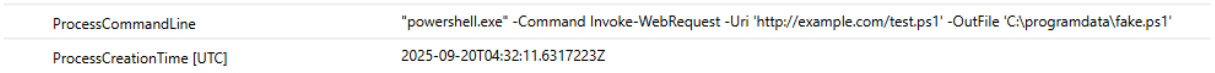
Alert Rule Triggered: 9/20/2025 12:38:43 PM

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:

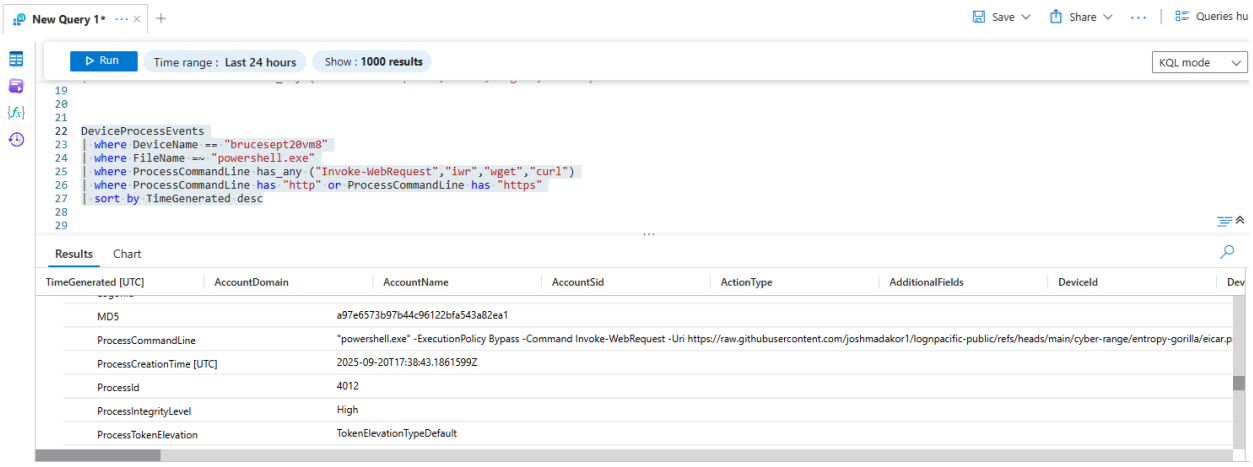




These screenshots and information was returned after running this query in Sentinel:

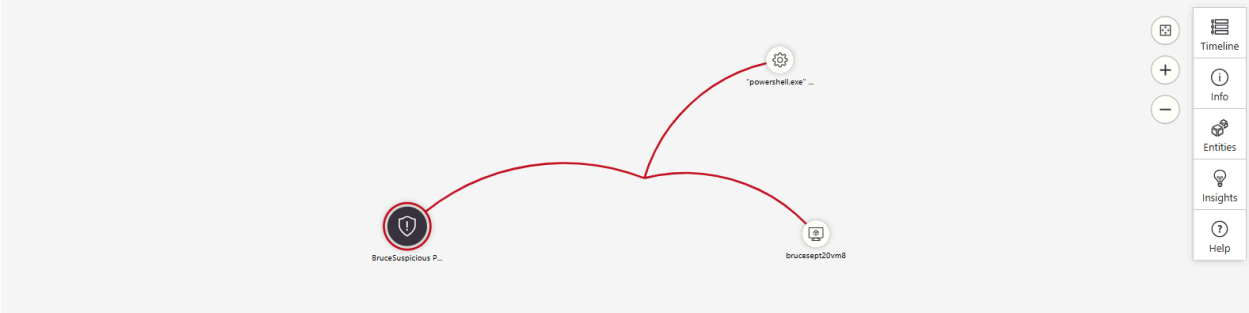
DeviceProcessEvents
| where FileName has "powershell.exe"
| where ProcessCommandLine has_any ("Invoke-WebRequest", "iwr", "wget", "curl")

Investigation and Analysis:
The Alert has been triggered, here is evidence of the findings from the script that was used within the Sentinal Alert Rule:



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

Entity Mapping screenshot:



The "BruceSuspicious PowerShell Download via Invoke-WebRequest" incident was triggered on one Device by one user. Device: "brucesept20vm8", User: ThreatHunt.
There was a downloaded script, and there could have been more had the Alert not been triggered.

Upon investigation the triggered incident "BruceSuspicious PowerShell Download via Invoke-WebRequest" it was discovered that the following PowerShell commands were run on Virtual Machine "brucesept20vm8"

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"
}
```

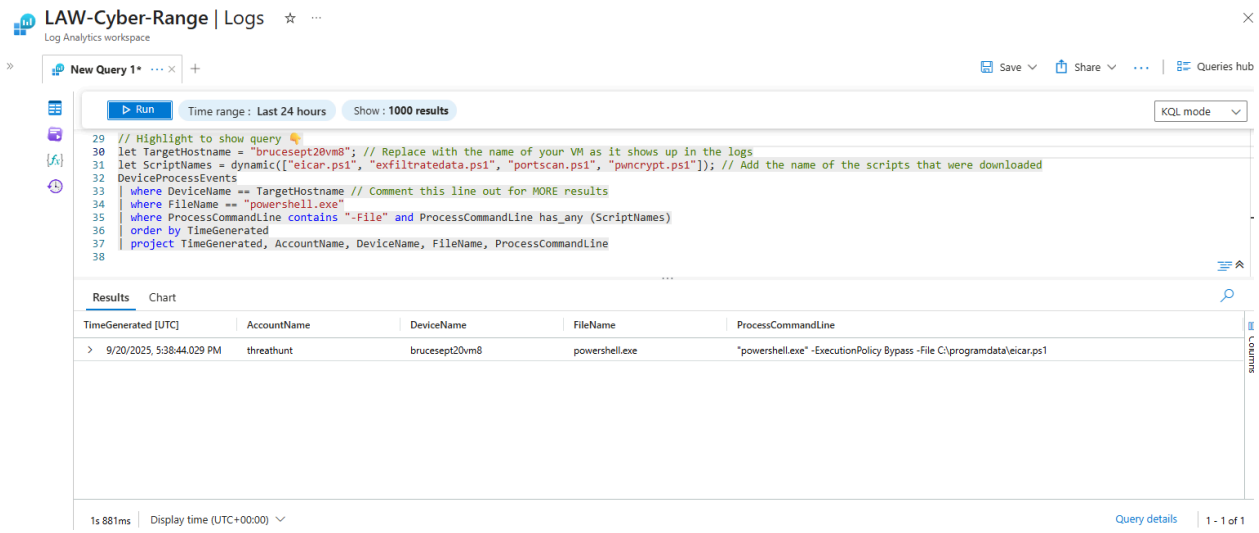
End logging
Log-Message "EICAR test file creation completed."

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 = "brucesept20vm8"; // 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:



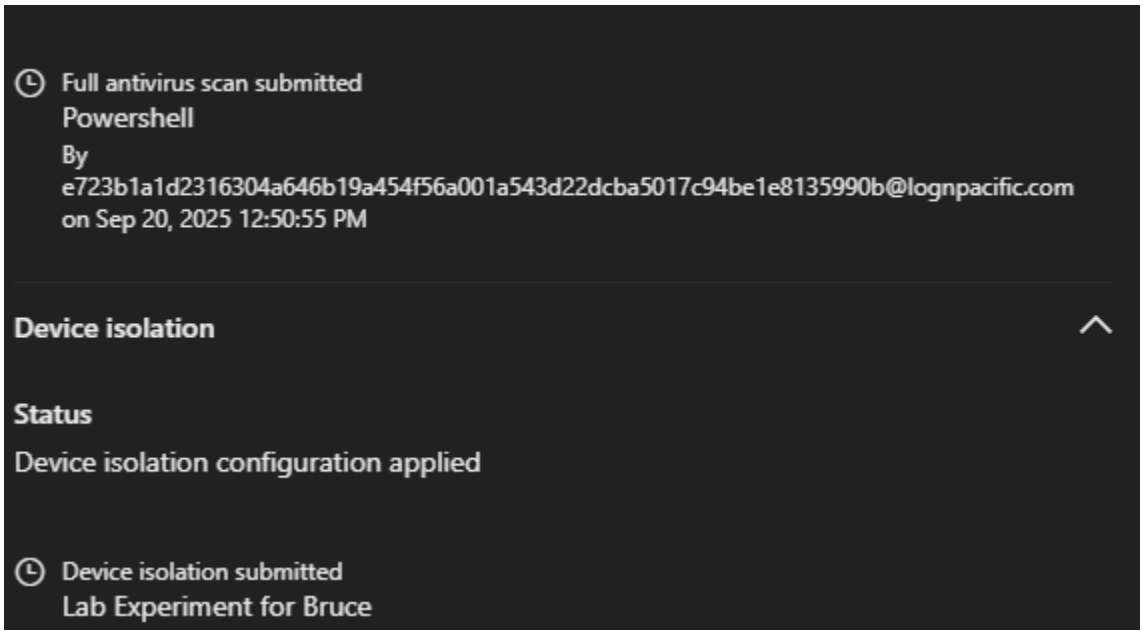
We passed the script off to the Malware reverse engineering team (A.K.A.: Me and ChatGPT), and these were the one liners that they came up with for this script:

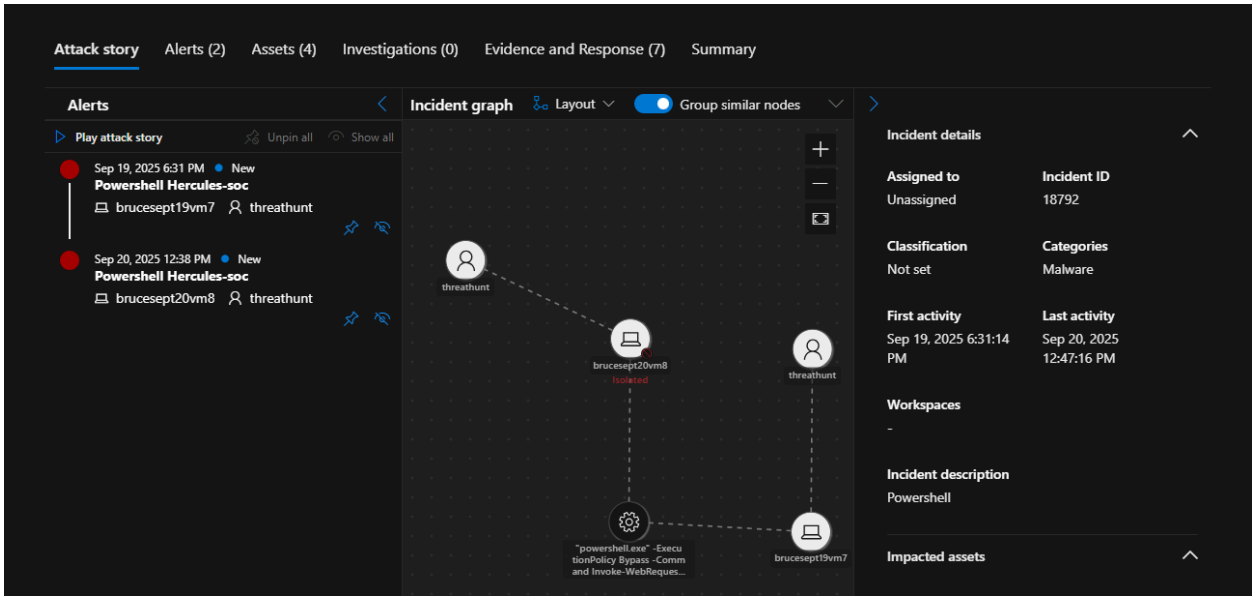
"powershell.exe" -ExecutionPolicy Bypass -Command Invoke-WebRequest -Uri https://raw.githubusercontent.com/joshmadakor1/lognpacific-public/refs/heads/main/cyber-range/entropy-gorilla/eicar.ps1 -OutFile C:\programdata\eicar.ps1

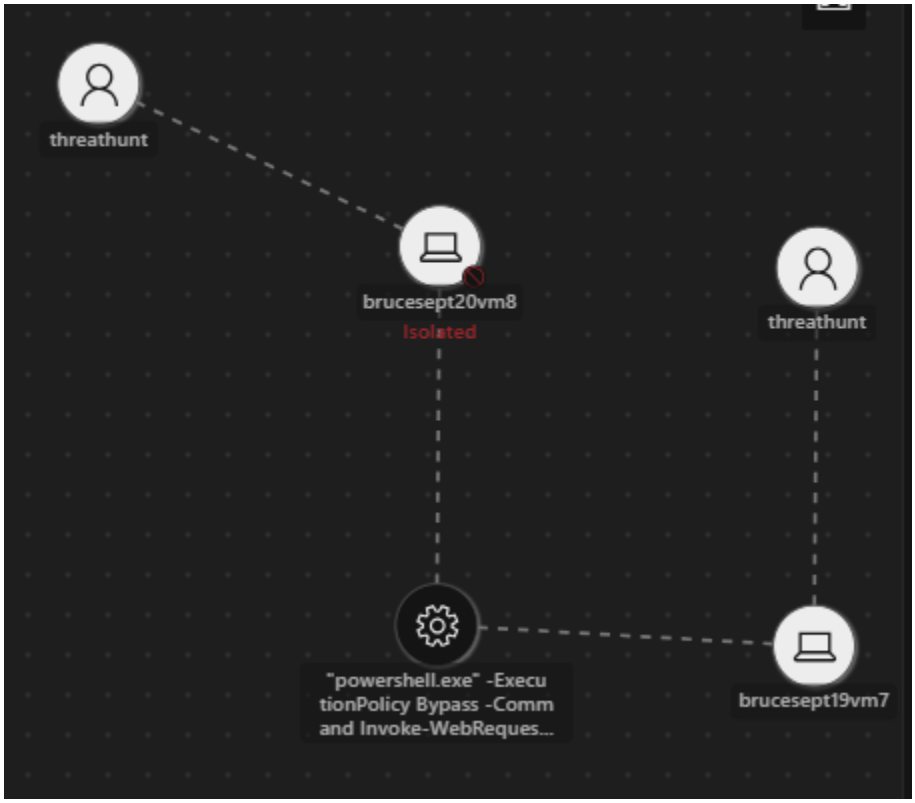
- Creates an eicar test file, a standard for testing Anti-Virus solutions, and logs the process.
- This also triggers a process of becoming persistent if left alone, and enabling the Malicious Attacker to exfiltrate data.

Containment, Eradication, and Recovery

The Virtual Machine has been Isolated in MDE.







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 Positive