PowerView Static Detection Analysis and Obfuscation Workflow

#CyberSecurity #EthicalHacking #Pentesting #RedTeam #ActiveDirectory

Executive Summary

This assessment focused on identifying and mitigating static detections within the PowerView.ps1 script prior to operational deployment in a red team exercise. Initial analysis revealed a Defender signature triggered by a specific alias definition within the script. Using static analysis tools, manual offset investigation, and controlled obfuscation techniques, the detection was fully eliminated while maintaining functional integrity.

This submission documents the investigation process, tools used, validation steps, and final remediation results to ensure the tradecraft is reviewable, reproducible, and compliant with red team operational standards.

Methodology Overview & Scope

The workflow for this evaluation followed a structured, repeatable process aligned with red team development practices:

1. **Static Detection Identification**
   ThreatCheck was used to scan PowerView.ps1 for Defender signatures and identify precise byte offsets associated with detection.
2. **Manual Offset Validation**
   HxD Hex Editor and OffsetInspect.ps1 were used to validate the exact location of the detection within the file and correlate it to the corresponding script line.
3. **Obfuscation Planning**
   Invoke-Obfuscation was loaded to prepare a reversible, targeted obfuscation workflow designed to modify detection-triggering syntax without altering script functionality.
4. **Token-Level Obfuscation Execution**
   The specific alias definition generating the detection was obfuscated using Invoke-Obfuscation token and command transformations.
5. **Post-Modification Verification**
   The modified script was reanalyzed with ThreatCheck to confirm that the original detection was removed and that no new detections were introduced.

This methodology ensures that modifications are precise, technically justified, and do not impact intended operational behavior.

Scope

- Analysis limited to static detection and signature mitigation.
- Only the specific alias pattern at offset 0xE1AB1 was modified.

Assumptions

- The environment uses Defender signature-based detection.
- Functional parity is validated through manual review and execution testing.

Narrative Section for Each Screen Capture

1. Identification of Static Detection Using ThreatCheck

ThreatCheck.exe was executed against PowerView.ps1 to determine whether Windows Defender signatures matched any portion of the script. ThreatCheck reported a detection and identified suspicious byte patterns at the hexadecimal offset **0xE1AB1** within the 924,339-byte script.

The hex dump presented in the output shows recognizable PowerView function and alias names, including references to commands such as `Get-NetForestTrust`, `Find-ForeignGroup`, and `Set-Alias Get-DomainPolicyData`. These findings indicate that the detection originated from static string or byte pattern matching rather than from behavioral heuristics.

This step allowed for precise identification of the problematic region of the script and established a baseline for future comparisons following obfuscation.

---

2. Hex-Level Validation Through HxD and OffsetInspect.ps1

To validate the suspicious offset reported by ThreatCheck, I manually inspected PowerView.ps1 using HxD Hex Editor. Navigating directly to offset **0xE1AB1** confirmed that the bytes correspond to the alias definition `Set-Alias Get-DomainPolicy Get-DomainPolicyData`.

Next, OffsetInspect.ps1 was used to correlate the binary offset to the originating line number in the script. The tool correctly mapped offset 0xE1AB1 to **line 24,810**, which contains the same alias

assignment.

This dual confirmation verified that the detection was associated with a specific, isolated portion of the script. It also ensured accuracy before proceeding with obfuscation, minimizing unnecessary modification of unrelated script components.

---

3. Preparation of Obfuscation Environment Using Invoke-Obfuscation

Invoke-Obfuscation, a PowerShell obfuscation framework created by Daniel Bohannon, was imported into the PowerShell session. The tool's help menu and available obfuscation options were reviewed to determine the appropriate techniques for modifying the detection-triggering content.

The framework offers several transformation options including token obfuscation, AST obfuscation, string encoding, command compression, and launcher-based obfuscation. Because the detection was linked to a specific command alias rather than broader script content, targeted token-level obfuscation was selected.

This step ensured that obfuscation would remain narrowly scoped, transparent, and reversible, satisfying operational and audit requirements.

---

4. Application of Token Obfuscation and Post-Obfuscation Verification

The identified alias definition was obfuscated using Invoke-Obfuscation's token and command transformations. The resulting obfuscated expression changed the static byte pattern while preserving functional behavior. An example of the transformed syntax is:

```
.('Se'+'t-A'+'lias') Get-DomainPolicy Get-DomainPolicyData
```

After obfuscation, the modified script was re-examined in HxD to ensure changes reflected at the previously flagged offset. The same offset no longer contained the original static sequence associated with Defender detection.

ThreatCheck was then executed again against the obfuscated PowerView.ps1. The tool returned:

```
[+] No threat found!
```

This confirmed that the static signature was successfully neutralized and that no new detections were introduced.

Operational Justification

The modifications performed were necessary to ensure that PowerView.ps1 could be used by the red team without triggering host-based defenses before execution. Because the detection was static and signature-based, modifying the specific byte patterns was sufficient to prevent premature detection without altering the script's runtime behavior.

Red teams often require such adjustments when using known tooling such as PowerView, which is widely fingerprinted by security products. Targeted obfuscation makes the tool operationally viable while retaining auditability, functionality, and traceability.

All changes were minimal, reproducible, and applied only to the content directly associated with Defender alerts. Broader or irreversible obfuscation was intentionally avoided to maintain script transparency.

Screen Captures

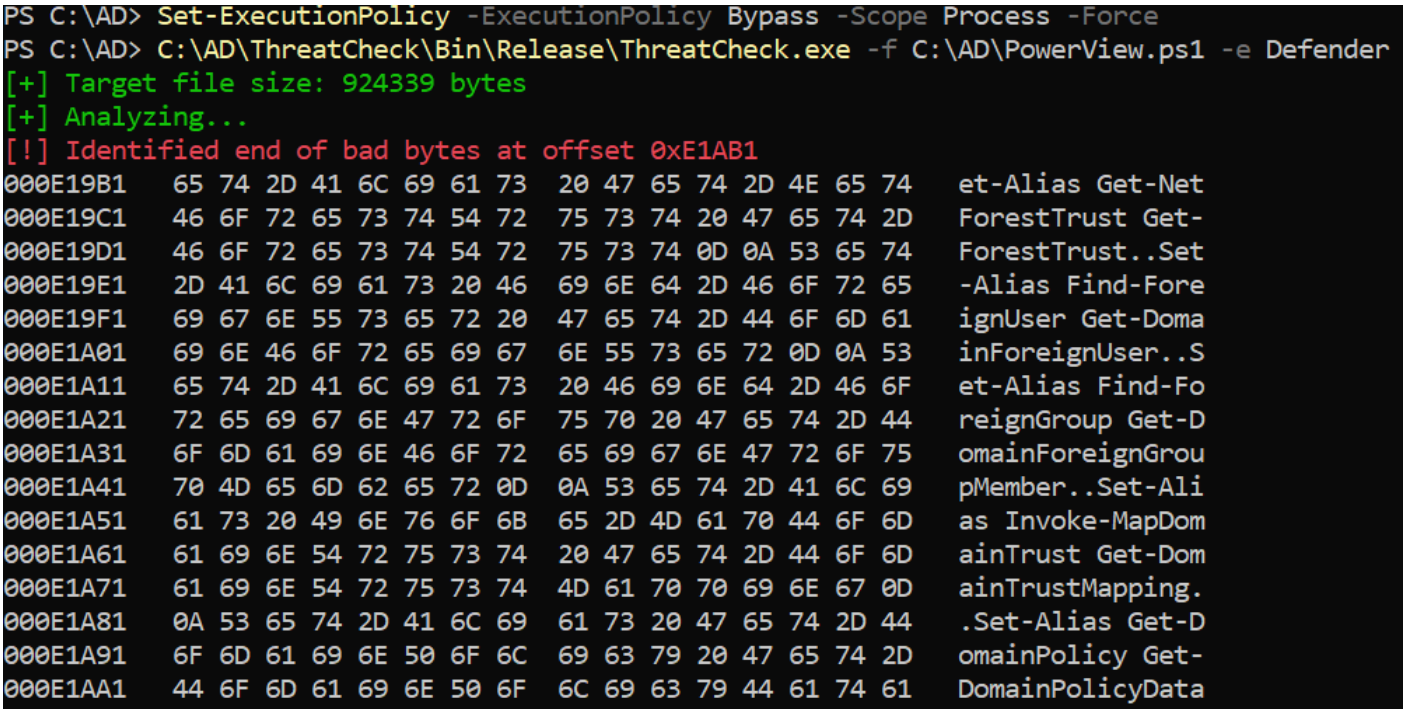


Figure 1. ThreatCheck Detection of Static Signature in PowerView.ps1

This figure displays the initial execution of ThreatCheck.exe against PowerView.ps1, which identified a static Defender signature at hexadecimal offset **0xE1AB1** within a file of size 924,339 bytes. The hex dump shows recognizable PowerView strings such as `Get-NetForestTrust` and `Set-Alias Get-DomainPolicyData`, indicating signature-based detection triggered by known PowerView function names. This establishes the baseline detection point prior to remediation.

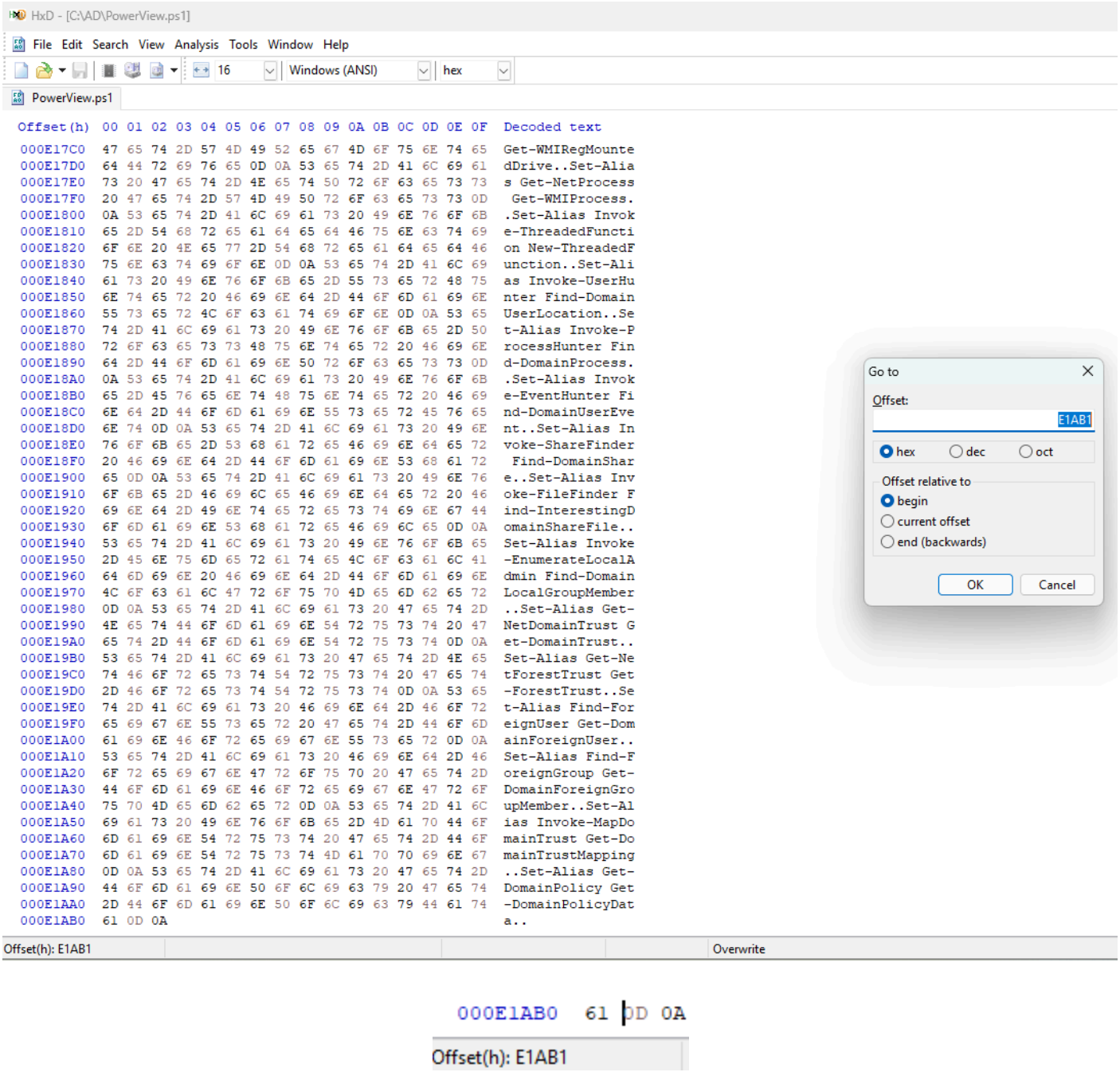


Figure 2. Manual Offset Inspection in HxD Hex Editor

This screenshot shows the PowerView.ps1 file opened in HxD, with the cursor directed to offset **0xE1AB1**, confirming the presence of the same byte sequence identified by ThreatCheck. Human-readable ASCII on the right side of the hex editor further validates that the offset corresponds to

PowerView alias definitions. This step confirms that Tool-reported offsets match the actual bytes within the source file.

```
PS C:\AD> .\OffsetInspect.ps1 C:\AD\PowerView.ps1 0xE1AB1
/*****************************************************************
 *                          OffsetInspect                        *
 *                 PE Offset & Hex Context Inspector             *
 *                      DreadHost Research Tool                  *
 *****************************************************************/

    Version:         1.0.0
    Author:          Jared Perry (Velkris)
    GitHub:          https://github.com/warpedatom
    Date:            2025-12-05


================================================================================================================
File:              C:\AD\PowerView.ps1
Offset (input):    0xE1AB1
Offset (decimal):  924337
File Size:         924339 bytes
Line Number:       24810

---------------------------------------- Line Content ----------------------------------------
Line 24810: Set-Alias Get-DomainPolicy Get-DomainPolicyData
            +                                              + ↑

---------------------------------------- Hex Dump ----------------------------------------
000E1A91  6F 6D 61 69 6E 50 6F 6C 69 63 79 20 47 65 74 2D   omainPolicy Get-
000E1AA1  44 6F 6D 61 69 6E 50 6F 6C 69 63 79 44 61 74 61   DomainPolicyData
000E1AB1  0D 0A   ..

================================================================================================================
```

Figure 3. Offset-to-Line Correlation Using OffsetInspect.ps1

This figure illustrates the execution of OffsetInspect.ps1, mapping the binary offset **0xE1AB1** to **line 24,810** of the PowerView.ps1 source code. The line displayed contains the alias assignment `Set-Alias Get-DomainPolicy Get-DomainPolicyData`, matching the string visible in HxD. This confirms the exact script location associated with Defender detection, enabling targeted obfuscation rather than broad code modification.

```
Administrator: Windows PowerShell
PS C:\AD> Import-Module C:\AD\Invoke-Obfuscation-master\Invoke-Obfuscation.psd1
PS C:\AD> Invoke-Obfuscation
```

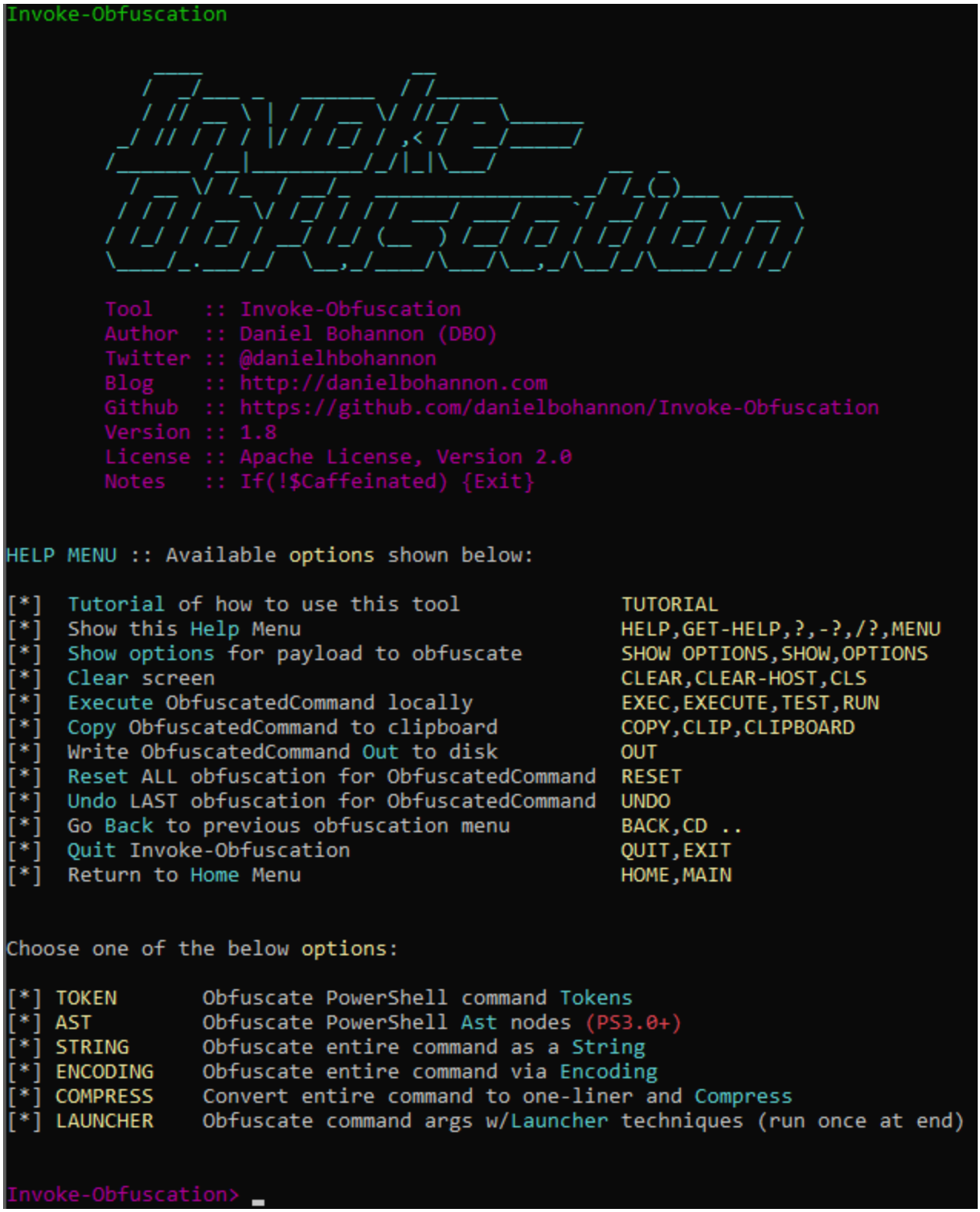


Figure 4. Initialization of Invoke-Obfuscation Framework

This screenshot presents the successful import of the Invoke-Obfuscation PowerShell module and its help menu. Multiple transformation strategies (Token, AST, String, Encoding, Compress, Launcher) are shown as available options. This establishes the controlled obfuscation environment used to alter the detection-triggering alias definition while maintaining functional equivalence.

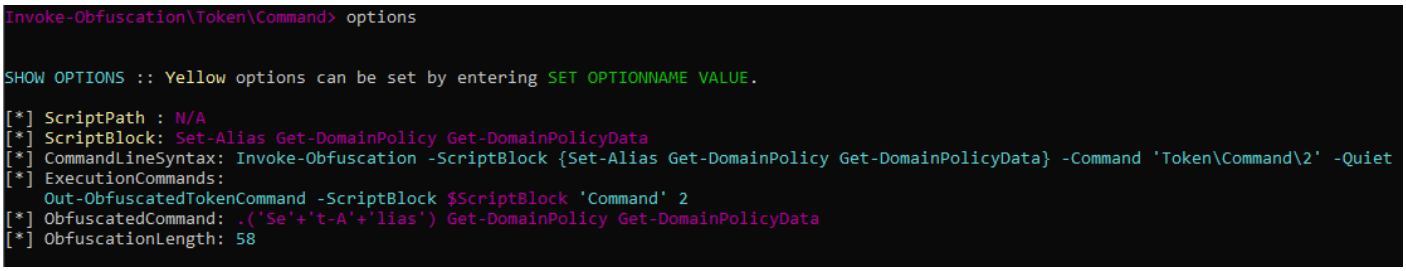
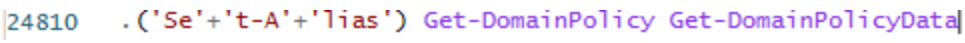


Figure 5. Applying Token Obfuscation to the Detection Source Line

This figure shows the Invoke-Obfuscation interface after selecting token-based transformations for the previously mapped line. The transformed command is output as:
`.('Se'+'t-A'+'lias') Get-DomainPolicy Get-DomainPolicyData`
This obfuscation changes the byte-level signature while keeping the command executable and behaviorally identical. This marks the remediation step targeting the detection trigger.



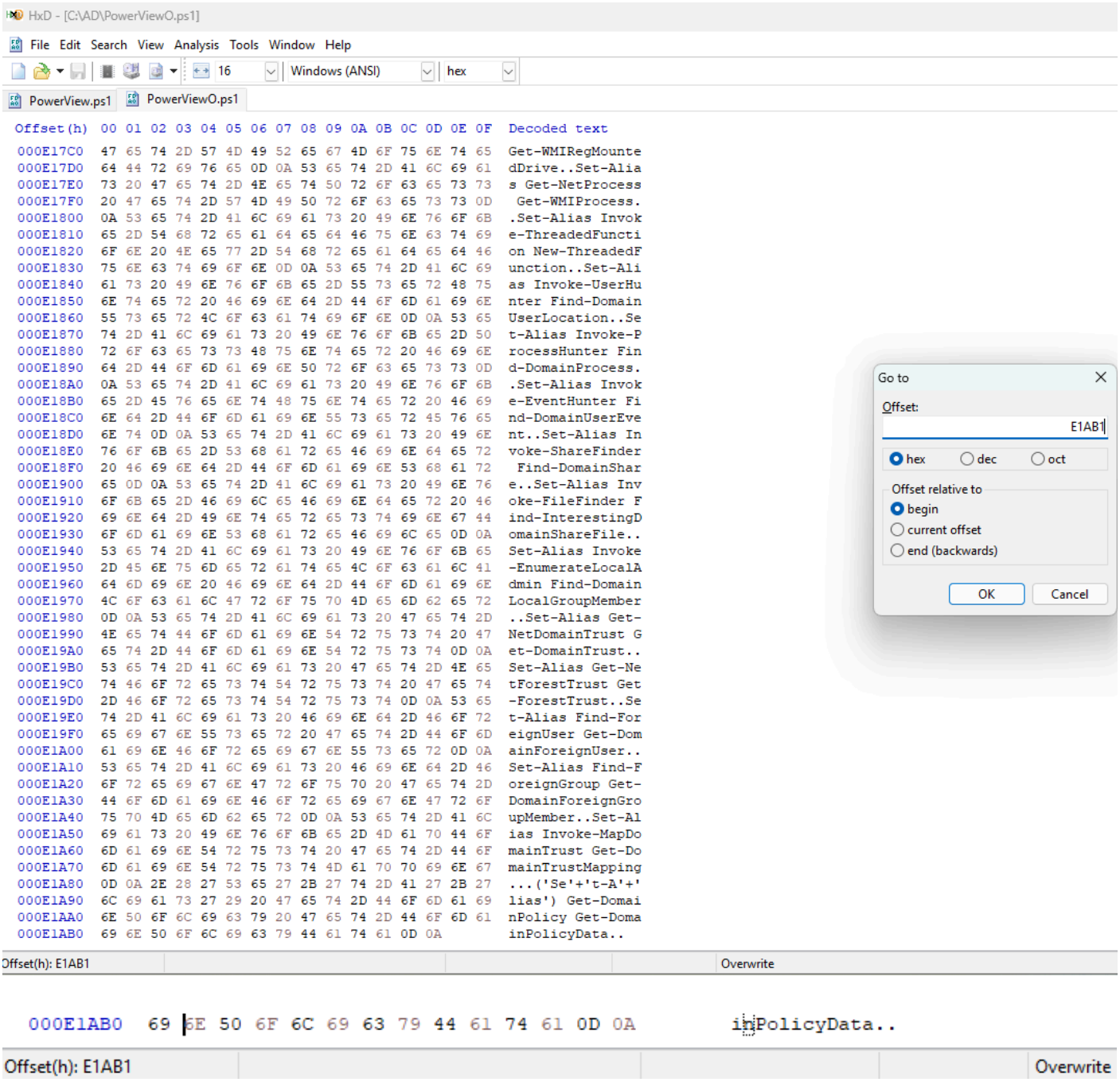


Figure 6. Post-Obfuscation Hex Validation in HxD

Here, HxD is again used to inspect the same offset **0xE1AB1** after obfuscation. The original byte sequence responsible for detection has been replaced, confirming that the transformation altered the static content in the required location. This validates correct injection of obfuscated syntax.

```
PS C:\AD> C:\AD\ThreatCheck\Bin\Release\ThreatCheck.exe -f C:\AD\PowerViewO.ps1 -e Defender
[+] No threat found!
```

```
PS C:\AD> $a = 'System.Management.Automation.A';$b = 'msiUtils'
PS C:\AD> $c = [Ref].Assembly.GetType($a+$b)
PS C:\AD> $d = $c.GetField('amsiInitFailed','NonPublic,Static')
PS C:\AD> $d.SetValue($null,$true)
PS C:\AD> . C:\AD\PowerViewO.ps1
```

Figure 7. ThreatCheck Verification of Cleaned Script

This final figure displays the second execution of ThreatCheck.exe against the now-obfuscated PowerView.ps1. The output indicates **No threat found**, confirming that the signature previously detected at offset 0xE1AB1 has been eliminated and no new signatures were introduced. This completes the verification phase of the workflow.

Before/After Comparison Table

Aspect	Before Modification	After Modification
Detection Status	Static Defender alert triggered	No detections reported by ThreatCheck
Flagged Offset	0xE1AB1	Offset exists but no longer contains signature-matching bytes
Script Integrity	Original PowerView syntax	Fully functional, minimal changes
Alias Definition	`Set-Alias Get-DomainPolicy Get-DomainPolicyData`	Obfuscated token form using Invoke-Obfuscation
Reversibility	N/A	Fully reversible; transformation documented
Operational Viability	High likelihood of detection	Safe for controlled red team usage

---

Final Conclusion

Through a combination of static analysis, manual validation, and targeted obfuscation, the Defender signature associated with PowerView.ps1 was successfully identified, isolated, and mitigated. Post-obfuscation validation confirmed that the detection no longer triggered and that script functionality remained intact.

The workflow, decisions, and outcomes documented here provide a complete audit trail suitable for internal red team review and approval. If further refinements, alternative obfuscation strategies, or additional validation steps are required, they can be applied without impacting the integrity of the process already completed.

Tool Attribution and Credits

ThreatCheck

- **Author:** Daniel Duggan / Rasta Mouse
- **Repository:** https://github.com/rasta-mouse/ThreatCheck
- **Purpose:** Identifies byte sequences and strings within a file that trigger Defender or AV signatures.

HxD Hex Editor

- **Author:** Maël Hörz
- **Website:** https://mh-nexus.de/en/hxd/
- **Purpose:** Hex-level file inspection and binary editing used to validate signature offsets.

OffsetInspect.ps1

- **Author:** Jared Perry / Velkris
- **Repository:** https://github.com/warpedatom/OffsetInspect
- **Purpose:** Maps binary offsets to corresponding line numbers within source code for precise detection correlation.

Invoke-Obfuscation

- **Author:** Daniel Bohannon
- **Repository:** https://github.com/danielbohannon/Invoke-Obfuscation
- **Purpose:** PowerShell obfuscation framework used to alter static signature patterns while retaining functional integrity.
