

## Zumanek: novo malware tenta roubar credenciais de serviços das vítimas

By Cassius Puodzius

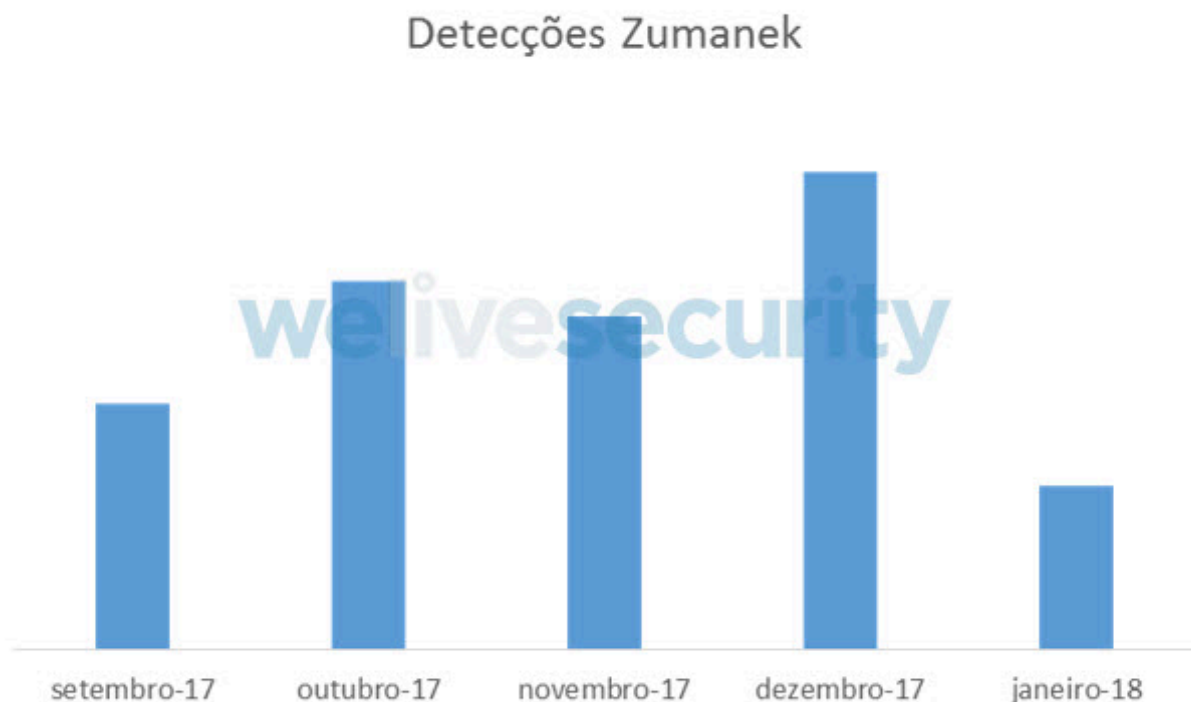
Archived: 2026-04-05 18:19:07 UTC

Há cerca de três meses detectamos **uma nova família de banker com foco no Brasil**: Zumanek.

A história da escolha do nome desta família é um tanto quanto peculiar. Na verdade, é uma homenagem ao nosso colega Jakub Tomanek, da República Tcheca, que teve um acidente enquanto andava de bicicleta e quebrou um de seus dentes ("Zub" em tcheco).

Após esse acidente, exatamente no dia em que Jakub voltou a trabalhar, essa família de banker foi detectada. Essa relação entre os dois fatos, foi a fonte de inspiração do nome Zumanek (Zub + Tomanek => Zumanek), de modo que Jakub até hoje guarda o dente quebrado em sua carteira.

Apesar do nome tcheco, **a ameaça é detectada (quase que) exclusivamente no Brasil**. Seu nível de detecção ainda não a coloca dentre do top 10 de spywares/bankers mais detectados no país, mas essa família dá mostras dos planos futuros do cibercrime brasileiro que está focado em **bancos e criptomoeadas**.



Neste post, vamos analisar uma amostra dessa família (versão 3.0) a fim de entender seu funcionamento, verificar as precauções tomadas por seus desenvolvedores para evitar a detecção e entender quais medidas podem ser tomadas para estar protegido.

## Zumanek: Downloader

Assim como a grande maioria dos malwares em atividade, sua cadeia de ataque é subdividida em diferentes estágios. O objetivo disso é fazer com que seja possível manter estágios do ataque desconhecidos o máximo possível, evitando efetua-los em máquinas que não cumprem algum perfil desejado.

Para chegar às vítimas, os cibercriminosos se valem da [Engenharia Social](#) a fim de convencer a vítima a baixar e executar esse primeiro estágio de infecção.

No **primeiro estágio** do Zumanek (i.e.: downloader), o intuito é fazer uma triagem inicial da máquina onde está sendo executado, realizar o download do payload final e, por fim, executá-lo na máquina comprometida.

A fim de proteger o downloader (e como veremos, também o banker final), os desenvolvedores do Zumanek, nas versões analisadas, utilizam o packer PECompact.

Para a análise estática e dinâmica dessa amostra, é necessário desempacotá-la. Na versão utilizada, esse procedimento pode ser realizado até mesmo [manualmente](#).

Apesar da preocupação para dificultar a engenharia reversa de seu malware, a mesma precaução não parece ter sido tomada na indicação do timestamp de compilação, que data de 28 de dezembro de 2017, mesmo período das primeiras detecções desta amostra, que não parece ter sido alterada.

| pFile    | Data     | Description             | Value                          |
|----------|----------|-------------------------|--------------------------------|
| 00000104 | 014C     | Machine                 | IMAGE_FILE_MACHINE_I386        |
| 00000106 | 0002     | Number of Sections      |                                |
| 00000108 | 5A2EADEC | Time Date Stamp         | 2017/12/11 Mon 16:10:20 UTC    |
| 0000010C | 00000000 | Pointer to Symbol Table |                                |
| 00000110 | 00000000 | Number of Symbols       |                                |
| 00000114 | 00E0     | Size of Optional Header |                                |
| 00000116 | 818E     | Characteristics         |                                |
|          | 0002     |                         | IMAGE_FILE_EXECUTABLE_IMAGE    |
|          | 0004     |                         | IMAGE_FILE_LINE_NUMS_STRIPPED  |
|          | 0008     |                         | IMAGE_FILE_LOCAL_SYMS_STRIPPED |
|          | 0080     |                         | IMAGE_FILE_BYTES_REVERSED_LO   |
|          | 0100     |                         | IMAGE_FILE_32BIT_MACHINE       |
|          | 8000     |                         | IMAGE_FILE_BYTES_REVERSED_HI   |

Figura 1: Timestamp de compilação conforme indicado no header PE (provavelmente não alterado)

Um dos motivos que explicam o porquê das detecções serem (quase que) exclusivamente brasileiras, está no fato do downloader verificar a língua do sistema escolhida pelo usuário.

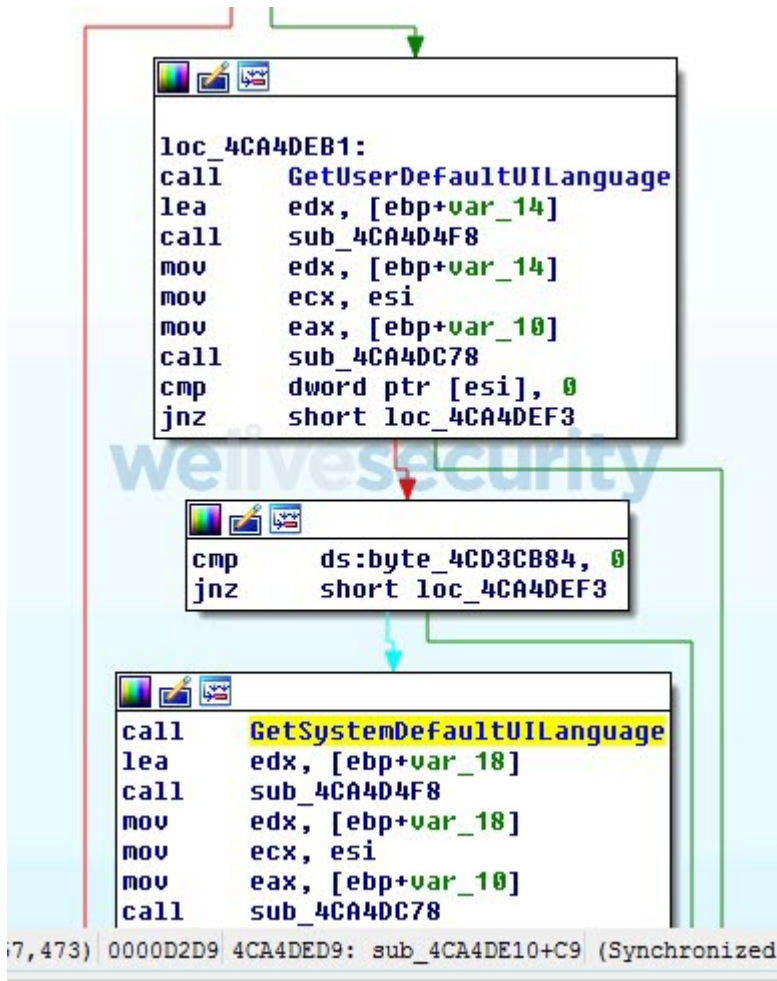


Figura 2: Obtenção da linguagem do sistema durante execução do

O retorno da chamada para GetSystemDefaultUILangafe é tratada e verifica-se se corresponde a 'pt-br'.

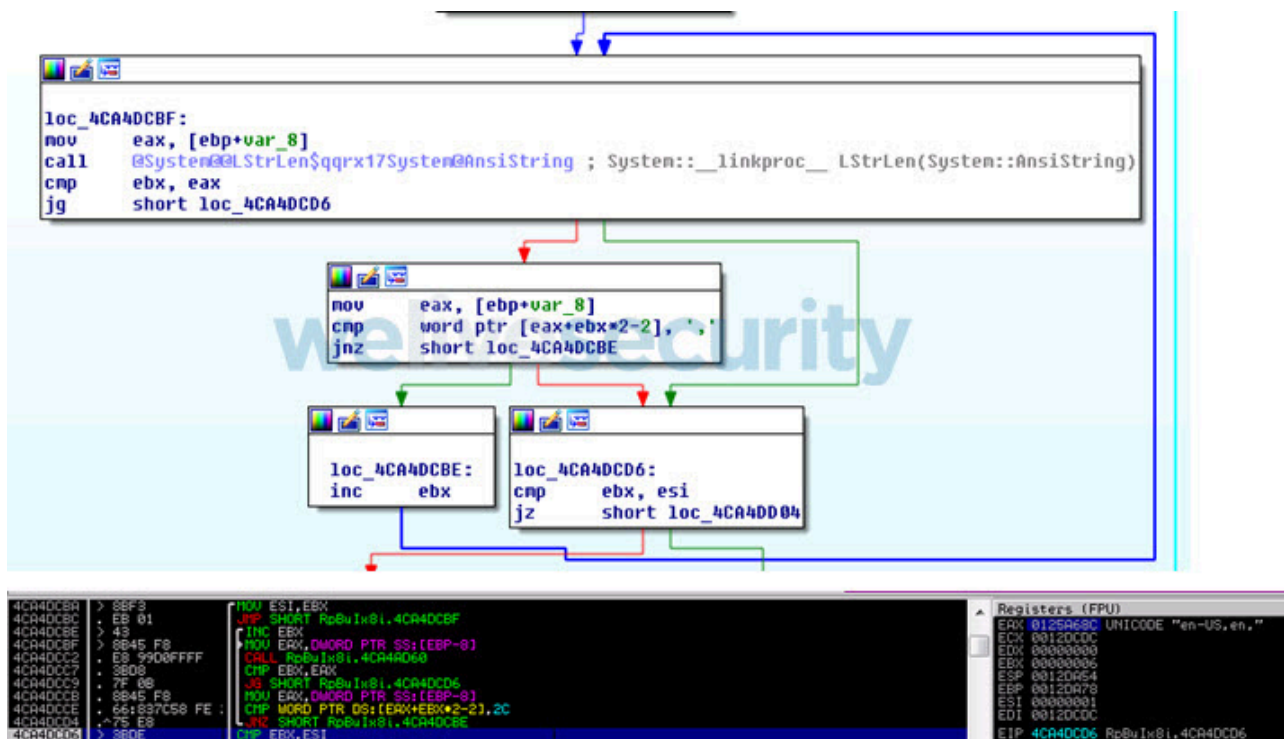


Figura 3: Verificação da linguagem do sistema (patch do JMP em tempo de execução)

Além disso, antes de tentar fazer o download de qualquer arquivo, o Downloader verifica a presença de diferentes antivírus. Caso algum deles seja detectado, o processo é imediatamente encerrado.

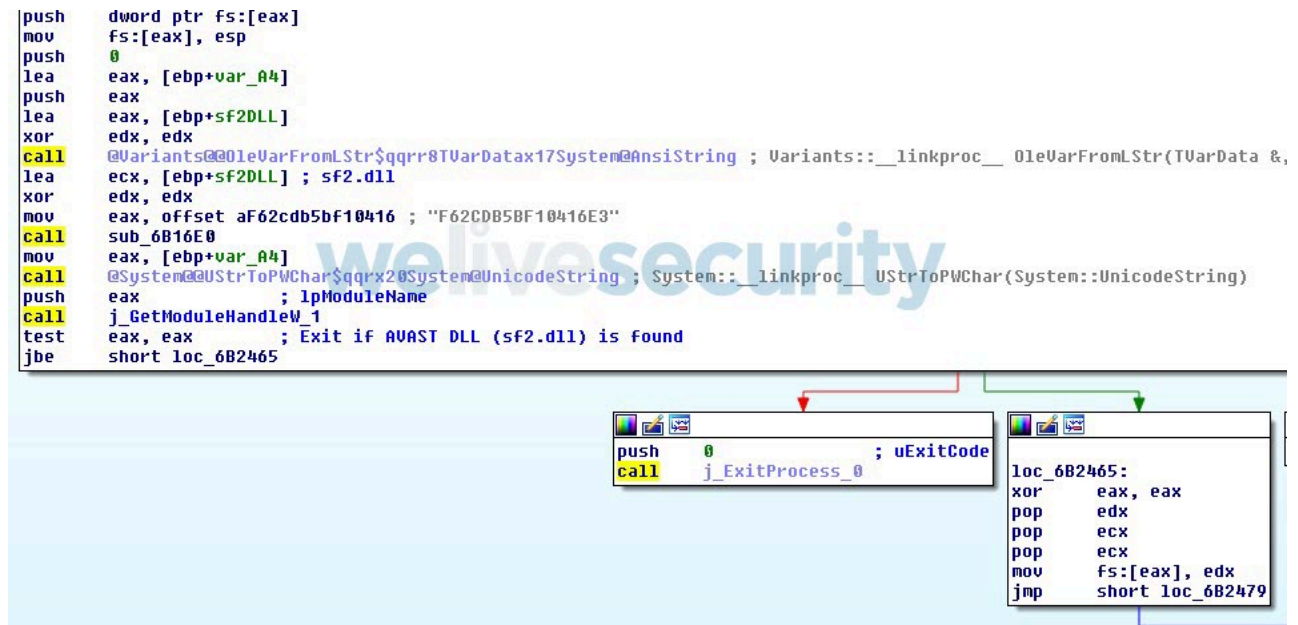


Figura 4: Verificação do módulo sf2.DLL no processo para detecção do AVAST

Listagem completa de AVs/proteções verificados:

- sf2.dll (AVAST)
- snxhk.dll (AVAST)
- cmdvrt32.dll (COMODO)
- SxIn.dll (360 Total Security)
- SbieDll.dll (Sandboxie)

A malware prossegue com o download do payload final (i.e.: banker) e sua execução na máquina da vítima.

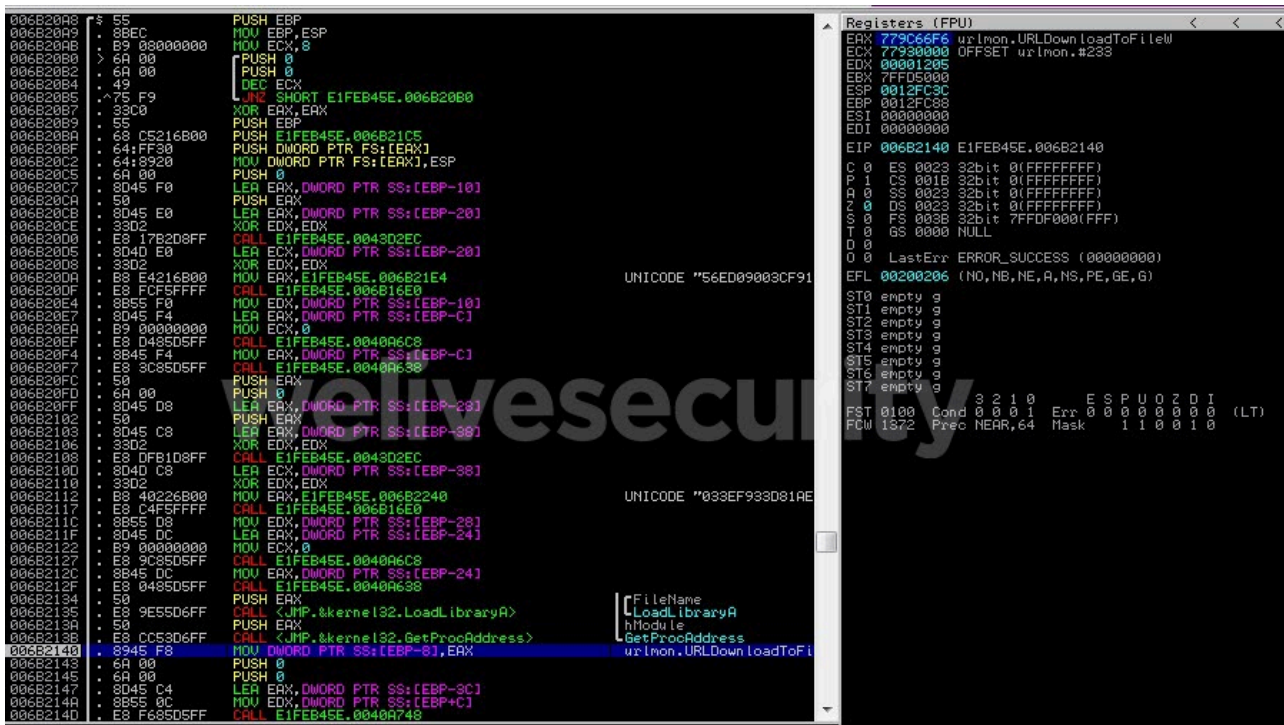


Figura 5: Execução de urlmon.URLDownloadToFileW para download do payload final

Nota-se que logo antes da execução de urlmon.URLDownloadToFileW, há strings hexadecimais sendo passadas como argumento da chamada em 006B16E0 (posição da memória virtual que pode mudar dependendo de onde o módulo for carregado). É importante perceber que o Delphi segue a [convenção de registradores Borland fastcall](#), que em x86 passa parâmetros através de EAX, EDX, ECX e, somente depois, através da pilha.

```
xor     eax, eax
push   ebp
push   offset aSR          ; "Tr+"
push   dword ptr fs:[eax]
mov    fs:[eax], esp
push   0
lea   eax, [ebp+URLDownloadToFileW]
push  eax
lea   eax, [ebp+URLDownloadToFileW_]
xor   edx, edx
call  @Variants@@@01eVarFromLStr$qqrr8TVarDataX17System@AnsiString ; Variants::_
lea   ecx, [ebp+URLDownloadToFileW_]
xor   edx, edx
mov   eax, offset a56ed09003cf917 ; "56ED09003CF917E429FF24E966B4B376AE73AB"
call  sub_6B16E0
mov   edx, [ebp+URLDownloadToFileW]
lea   eax, [ebp+URLDownloadToFileW_ASCII]
mov   ecx, 0
call  @System@@LStrFromUStr$qqrr27System@_AnsiStringT$us$i0$_x20System@Unicode!
mov   eax, [ebp+URLDownloadToFileW_ASCII]
call  @System@@@LStrToPChar$qqrx27System@_AnsiStringT$us$i0$_ ; System::__linkpi
push  eax
push  0
lea   eax, [ebp+ur1nonDLL]
push  eax
lea   eax, [ebp+ur1nonDLL_]
xor   edx, edx
call  @Variants@@@01eVarFromLStr$qqrr8TVarDataX17System@AnsiString ; Variants::_
lea   ecx, [ebp+ur1nonDLL_]
xor   edx, edx
mov   eax, offset a033ef933d81aee ; "033EF933D81AEE7C98628B"
call  sub_6B16E0
mov   edx, [ebp+ur1nonDLL]
lea   eax, [ebp+ur1nonDLL_ASCII]
mov   ecx, 0
call  @System@@@LStrFromUStr$qqrr27System@_AnsiStringT$us$i0$_x20System@Unicode!
```

Figura 6: Strings encriptadas na função de download

Como haveria de se esperar, essas strings estão encriptadas e guardam as informações necessárias tanto para a execução da chamada, quanto para obter as informações necessárias para o download do payload final que está hospedado no C&C.

```
mov     eax, offset a033ef933d81aee ; "033EF933D81AEE7C98628B"  
call   sub_6B16E0  
mov     edx, [ebp+urlmonDLL]  
lea     eax, [ebp+urlmonDLL_ASCII]  
mov     ecx, 0  
call   @System@@LStrFromUStr$qqrr27System@_AnsiStringT$us$i0$_x20  
mov     eax, [ebp+urlmonDLL_ASCII]  
call   @System@@LStrToPChar$qqrx27System@_AnsiStringT$us$i0$_ ; S  
push   eax ; lpLibFileName  
call   j_LoadLibraryA_0  
push   eax ; hModule  
call   j_GetProcAddress_1  
mov     [ebp+var_8], eax  
push   0  
push   0  
lea     eax, [ebp+var_3C]  
mov     edx, [ebp+PayloadZIPPath]  
call   @System@@LStrFromPWChar$qqrr17System@AnsiStringpb ; Borlan  
mov     eax, [ebp+var_3C]  
call   @System@@WStrToPWChar$qqrx17System@WideString ; System::__  
push   eax  
lea     eax, [ebp+var_40]  
mov     edx, [ebp+PayloadZIPURL]  
call   @System@@LStrFromPWChar$qqrr17System@AnsiStringpb ; Borlan  
mov     eax, [ebp+var_40]  
call   @System@@WStrToPWChar$qqrx17System@WideString ; System::__  
push   eax  
push   0  
call   [ebp+var_8]
```

Figura 7: Chamada de urlmon.URLDownloadToFileW resolvida em tempo de execução (call [ebp+var\_8])

O algoritmo de decriptação tem como entrada a string encriptada e outra string (unicode) que atua como uma chave (e varia de amostra para amostra). Byte a byte, o algoritmo vai obtendo o valor da string decriptada sempre fazendo uso do valor do último byte decriptado para o cálculo do próximo byte.

O método de decriptação das strings não segue nenhum padrão seguro de criptografia (e.g.: AES) e foi implementado exclusivamente para dificultar a análise estática do código. No entanto, é interessante notar que uma vez entendido como esse algoritmo funciona, é possível escrever um script para a obtenção das strings encriptadas não apenas dessa amostra, mas de outras da família Zumanek.

```
static_str =  
"FIUxRfaxgEaXalkLgaonACAZhAbnQOylEhHOqXYETApwxrpqkqWuWRYbnbglopKPSzdBI" #  
  
Exemplo  
enc_strs = ["F67E8782BE52C063"] # Exemplo  
def decrypt_str(enc_str):  
prev_byte = int(enc_str[:2], 16)  
dec_str = ""  
i = 2  
while i < len(enc_str): current_byte = int(enc_str[i:i+2], 16) static_char = static_str[i/2 - 1] dec_char =  
current_byte ^ ord(static_char) if dec_char >= prev_byte:  
dec_char -= prev_byte  
else:  
dec_char += 0xFF - prev_byte  
dec_str += chr(dec_char)  
prev_byte = current_byte  
i += 2  
return dec_str
```

Ao final de todos os downloads e da descompressão dos arquivos (já que o payload vem em forma de ZIP), é verificado se os arquivos foram baixados e modificados corretamente.

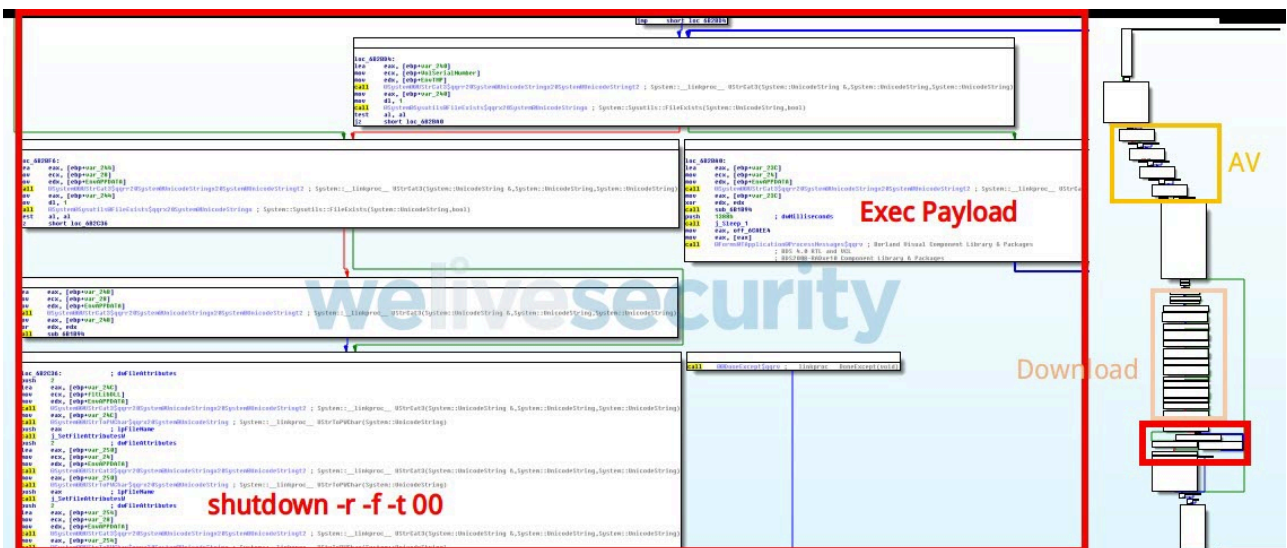


Figura 8: Fluxo de execução do downloader

Se a verificação for positiva, o payload final é executado, caso contrário os arquivos são escondidos (FILE\_ATTRIBUTE\_HIDDEN) e o sistema é reiniciado.

```

loc_6B2C36:                ; dwFileAttributes
push     2
lea     eax, [ebp+var_24C]
mov     ecx, [ebp+fltLibDLL]
mov     edx, [ebp+EnvAPPDATA]
call    @System@@UStrCat3$qqrr20System@Unicode!
mov     eax, [ebp+var_24C]
call    @System@@UStrToPWChar$qqrx20System@Uni
push    eax                ; lpFileName
call    j_SetFileAttributesW
push    2                  ; dwFileAttributes
lea     eax, [ebp+var_250]
mov     ecx, [ebp+var_24]
mov     edx, [ebp+EnvAPPDATA]
call    @System@@UStrCat3$qqrr20System@Unicode!
mov     eax, [ebp+var_250]
call    @System@@UStrToPWChar$qqrx20System@Uni
push    eax                ; lpFileName
call    j_SetFileAttributesW
push    2                  ; dwFileAttributes
lea     eax, [ebp+var_254]
mov     ecx, [ebp+var_28]

```

Figura 9: Atribuição de FILE\_ATTRIBUTE\_HIDDEN aos arquivos baixados

## Zumanek: Banker

O **segundo estágio** trata-se de um banker/RAT, cuja finalidade é prover ao atacante o controle remoto à máquina da vítima, enfocando no **roubo de credencias de acesso a serviços online banking e a casas de câmbio de criptomoeda**.

A cadeia de ataque é realizado de maneira muito simples: o downloader se encarrega de baixar um ZIP contendo dois arquivos, um executável legítimo (e assinado) e uma DLL maliciosa que é carregada pelo executável, executando, na sequência, o arquivo legítimo.

| Name   | Type | Compressed size |
|--------|------|-----------------|
| drive0 | File | 6,645 KB        |
| drive1 | File | 599 KB          |

Figura 10: Arquivos (comprimidos) baixados pelo downloader

Semelhantemente à proteção do downloader, o payload final (i.e.: drive0, uma DLL maliciosa) também utiliza o packer PECompact.

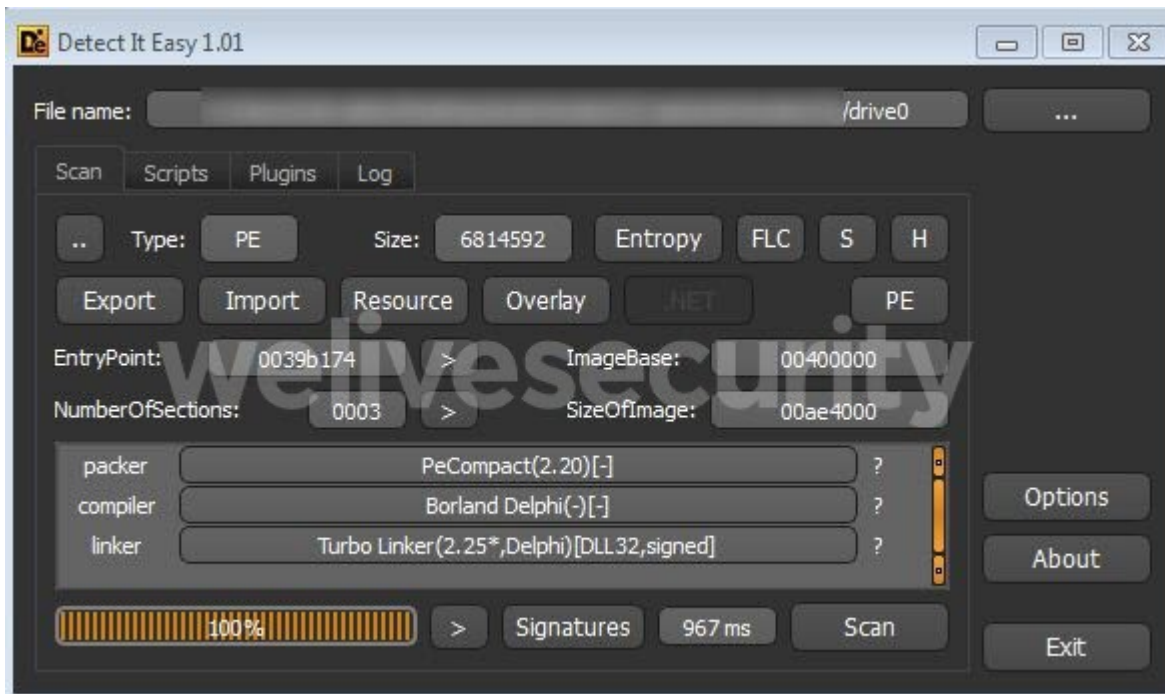


Figura 11: DLL maliciosa (drive0) protegida com PeCompact(2.20)

O outro arquivo (drive1) é uma cópia legítima (e assinada) do GbpSv.EXE. Esse arquivo é baixado junto à DLL maliciosa, que é renomeada para fltLib.DLL pelo downloader. Quando, então, o GbpSv.EXE é executado, ao invés da DLL legítima ser carregada, devido à ordem de busca de DLLs no sistema, a fltLib.DLL (maliciosa) será carregada e executada no lugar (i.e.: DLL Hijacking).

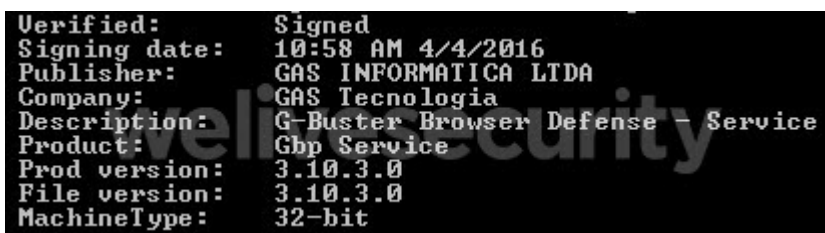


Figura 12: Informações do executável drive1 segundo o SigCheck

O procedimento de encriptação de strings é idêntico ao empregado no downloader (modificando-se somente a string unicode que funciona como chave). Com isso, de maneira estática, é possível entender a finalidade deste malware com base no conteúdo de algumas strings.

| String encriptada  | Conteúdo (i.e.: alvos) |
|--------------------|------------------------|
| 0018010C101B38C040 | BANRISUL               |
| 07001117283521     | SICOOB                 |
| 19011B1D222D39C6   | BBCOMBR                |
| 223FD066E166F10D15 | CITIBANK               |

| <b>String encriptada</b>         | <b>Conteúdo (i.e.: alvos)</b> |
|----------------------------------|-------------------------------|
| 342C35C048D978F009               | BANESTES                      |
| 47D37EFC0F12                     | BRADA (i.e.: BRADESCO)        |
| 4DD54FDE6CF91F3CC823589EEA00     | BANCOORIGINAL                 |
| 5BC74CD852DF75F07FFB7E           | BLOCKCHAIN                    |
| 68E17B8A9093                     | SANTA (i.e.: SANTANDER)       |
| 75ED7786989BBB                   | BANPAR                        |
| 94BD5FE17AFD1410                 | SICREDI                       |
| 9B87889FACB751D2                 | BITCION                       |
| B2A1ADAD                         | ITA (i.e.: ITAU)              |
| B3A3AA40CC54FE                   | FOXBIT                        |
| B656FC05061A0900                 | UNICRED                       |
| B958F375F6                       | HSBC                          |
| BB53C050DD5DF70A0179F9057BFD75   | MERCADOBITCION                |
| D543DC65F377                     | CAIXA                         |
| FB64FD0F1C292D3FC0364BA0D02845DD | BANCORENDIMENTO               |

Em sua execução, o módulo ftLib.DLL altera a chave de registro Software\Microsoft\CurrentVersion\Run para executar o binário (legítimo) sempre que o sistema é inicializado. Além disso, o módulo cria um novo processo de notepad.EXE e injeta-se na memória do mesmo.

```

loc_78A199:
xor     eax, eax
push   ebp
push   offset loc_78A3D3
push   dword ptr fs:[eax]
mov    fs:[eax], esp
lea    eax, [ebp+NumberOfBytesWritten]
push   eax                ; lpNumberOfBytesWritten
mov    eax, [ebp+dwSize]
push   eax                ; nSize
mov    eax, [ebp+lpBuffer]
push   eax                ; lpBuffer
mov    eax, [ebp+lpBaseAddress]
push   eax                ; lpBaseAddress
mov    eax, [ebp+hProcess]
push   eax                ; hProcess
call   j_WriteProcessMemory
test   eax, eax
jnz    short loc_78A1E0
    
```

Figura 13: Injeção de fltLib.DLL em notepad.EXE

Dessa forma, o processo do notepad.EXE carrega também a fltLib.DLL na memória. As ações maliciosas são realizadas apenas quando a fltLib.DLL está sendo executada em notepad.EXE.

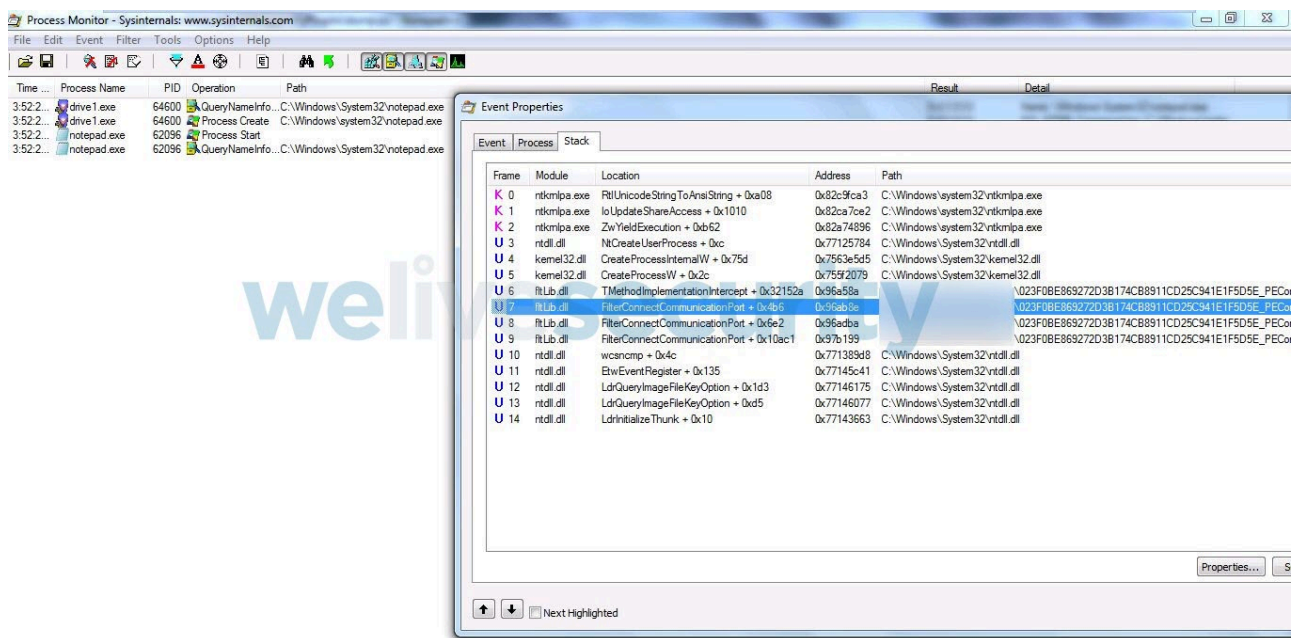


Figura 14: fltLib.DLL injetado no processo notepad.EXE executando a função (exportada) FilterConnectCommunicationPort.

O payload redireciona os acessos web das páginas alvo realizados no firefox.EXE ou chrome.EXE para ser executado via Internet Explorer, injetando formulários (form grabbing) para roubar as senhas de acesso das vítimas e enviá-las ao C&C.

```
call    sub_76A6A4
xor     eax, eax
push   ebp
push   offset loc_7693E9
push   dword ptr fs:[eax]
mov    fs:[eax], esp
lea    edx, [ebp+var_10]
mov    eax, offset a262945ea2ed81e ; "262945EA2ED81E739A3367EF57F15F8D20539B8"...
call   sub_74C500
mov    edx, [ebp+var_10]
lea    eax, [ebp+var_C]
call   sub_40B290
mov    edx, [ebp+var_C]
mov    eax, [ebp+var_4]
mov    eax, [eax+424h]
call   sub_53C7E0
lea    edx, [ebp+var_10]
mov    eax, offset aA8a24df119c57f ; "A8A24DF119C57FAA4152F551449F38D510D4AD5"...
call   sub_74C500
mov    edx, [ebp+var_18]
lea    eax, [ebp+var_14]
call   sub_40B290
```

Preencha os dados para prosseguir

Figura 15: Criação de formulário falso para o roubo de credencial (form grabbing)

A comunicação inicial com o C&C dá-se através de requisições HTTP POST, com os seguintes parâmetros:

- User agent: NULL
- Headers: Content-Type:application/x-www-form-urlencoded
- POST: "op={8 chars aleatorios}+{BASE64 de dados encriptados}"

Dados enviados via POST:

- %VERSION% - versão do banker
- %BANKPROTECTIONSW% - proteções instaladas (valores possíveis: ["SCAPD" | "WARSAW" | "GB" ])
- %COMPUTERID% - {Nome do Computador}+{Número Serial do Volume}
- %OSVERSION% - OS version: ["Desconhecido " | "Windows XP " | "Vista " | "Windows 7 " | "Windows 8 " | "Windows 10 "] + [" Intel" | " Intel Itanium-based X64" | " Arquitetura desconhecida" | " x64 (AMD ou Intel)"] + [" 64Bits" | " 32Bits"]
- %AVs% - Antivírus instalados
- %DATETIME% - data e hora

Exemplo: [4.5.0]#[SCAPD]#[MAQUINA]#[Windows 7 Intel 64Bits]#[01/10/2018 4:07:00 PM]#

```
mov     fs:[eax], esp
lea     edi, [ebp+var_4C]
mov     eax, offset a70bd7Fae52fb3f ; "70BD7FAE52FB3FFE3F4A92D676D20F39F00E24D"...
call    sub_74C500
mov     edx, [ebp+var_4C]
lea     eax, [ebp+var_48]
call    sub_40B290
mov     eax, [ebp+var_48]
lea     edx, [ebp+var_44]
call    sub_74FA34
mov     edx, [ebp+var_44]
lea     eax, [ebp+var_20]
call    sub_43DA24
push    0
lea     edi, [ebp+var_60]
mov     eax, offset aC446cf5f ; "C446CF5F"
call    sub_74C500
mov     eax, [ebp+var_60]
push    eax
lea     edi, [ebp+var_64]
mov     eax, offset aD977e472f97f85 ; "D977E472F97F85E85E4DA7C124465FE90DD6192"...
call    sub_74C500
mov     eax, [ebp+var_64]
push    eax
```

winmgmts:\\localhost\root\SecurityCenter2

WQL

SELECT \* FROM AntiVirusProduct

Figura 16: Obtenção do AV instalado na máquina fornecido pelo WMI provider SecurityCenter2

Além da comunicação via HTTP POST, três sockets são criados: Comando, Foto e Texto. As portas e os endereços aos quais os sockets se conectam estão hardcoded de forma encriptada no código do banker.

Quando os sockets estabelecem com sucesso uma conexão com o C&C, uma mensagem inicial é enviada:

- Socket FOTO: "SOQUETEFOTOS;VAZIO;VAZIO;VAZIO;VAZIO"
- Sockets COMANDO e TEXTO:  
"SOQUETETEXTOS;%COMPUTERID%;%OSVERSION%;%BROWSER%;%VERSION%;%BANK%;"

Os valores de %COMPUTERID%, %OSVERSION% e %VERSION% são iguais aos enviados via HTTP POST. Os valores possíveis para %BANK% seguem a lista da tabela acima, enquanto %BROWSER% pode assumir os valores ["Explorer" | "Firefox" | "Chrome" | "Opera" | "Microsoft Edge" | "Avast SafeZone" | "UC Browser" | "App Brada"].

Através do socket Comando, o operador do Zumanek pode enviar diversos comandos à máquina da vítima. A lista autoexplicativa de comandos está apresentada na tabela abaixo:

**LISTARMODULOS**  
**ATUALIZARMODULO**  
**ENIVARDUPLOCLIQUE**  
**ENIVARCLIQUE**  
**ENIVARCLIQUEINVERSO**  
**ENIVARMOUSEMOVE**  
**ENIVARTEXTO**  
**ENIVARTECLAS**  
**COLARDATA**  
**MUDARQUALIDADEBMP**  
**TIPOFOTO**  
**TIPOPRINT**  
**ENVIARMESSAGEBOX**  
**ABRIRCHROMEABRIRCHROME**  
**ABRIRFIREFOX**  
**ABRIRIEXPLORER**  
**MAXIMIZARBROWSER**  
**MINIMIZARBROWSER**  
**ENIVARCLIQUEARRASTA0**  
**ENIVARCLIQUEARRASTA1**  
**ENIVARNOMECLIENTE**  
**ENIVARAJUSTEXY**  
**ENVIARMOVIMENTOMOUSE1**  
**ENVIARMOVIMENTOMOUSE0**  
**ENVIARAUTOGETHANDLES0**  
**ENVIARAUTOGETHANDLES1**  
**LISTARHANDLES**  
**LISTARDESKTOPS**  
**SETARHANDLEALVO**  
**DETALHESJANELAFILHA**  
**ENVIARINTERVALOMOVEMOUSE**  
**CRIARDESKTOP**  
**REINICIARPC**  
**RESTARTKL**  
**BLOQUEARKL**  
**DELETARKL**  
**FECHARKL**  
**RESETARKL**  
**EXECUTARPCHUNTER**  
**FECHARINFO**  
**OCULTARBARRATAREFAS**  
**MOSTRARBARRATAREFAS**

**FECHARBROWSERS**  
**FINALIZARINFO**  
**RECONECTARREMOTO**  
**AJUSTARBROWSER**  
**ENVIARCONECTAPHOTO**  
**ENVIARDESCONECTAPHOTO**  
**MATAHOOK**  
**ATIVAKEYLOG**  
**DESATIVAKEYLOG**  
**RECEBERDADOSKEY**  
**DESATIVARAERO**  
**ATIVAAERO**  
**DESATIVATRUSTEER**  
**DETONARPC**  
**LIBERAATUALIZA**  
**XYRECORTE**  
**FECHABURACO**  
**ATUALIZEMAIL**  
**TRAVAUPD**  
**TRAVAGENERICA**  
**ATUALIZABB**  
**ATUALIZACEF**  
**ATUALIZASANTA**  
**ATUALIZAITA**  
**ATUALIZABRADA**  
**ATUALIZASICREDI**  
**ATUALIZAUNICRED**  
**ATUALIZASICOOB**  
**ATUALIZABANRISUL**  
**CANCELATELA**  
**BANRISULSENHA**  
**BBFISICASENHA8**  
**BBFISICASENHACONTA**  
**BBGFSENHACONTA**  
**BBGFSENHACERTIFICADO**  
**BRADAPOSICAOTABELA**  
**BRADACHAVE**  
**BRADATOKEN**  
**CEFASSINATURA**  
**ITAFISICASENHA**  
**ITATABELA**  
**ITAFISICATOKEN**

ITADATANASCIMENTO  
ITAFISICASMSTOKENITAFISICASMSTOKEN  
SANTATABELA  
SANTAASSTOKEN  
SANTASMSTOKEN  
SANTAASSINATURA  
SANTASOTOKEN  
SANTATOKEN  
SICOOBASSINATURA  
SICOOBSENHA4  
SICOOBSENHA6  
SICOOBTOKEN  
SICREDIASSINATURA  
SICREDITOKEN  
UNICREDITOKEN  
UNICREDASSINATURA  
BLOQUEARSICREDI  
BLOQUEARBB  
BLOQUEARITA  
BLOQUEARCEF  
BLOQUEARBRADA  
BLOQUEARSANTA  
BLOQUEARHSBC  
BLOQUEARBANRISUL  
BLOQUEARBANESTES  
BLOQUEARUNICRED  
BLOQUEARSICOOB  
BLOQUEARCITIBANK

Através do socket Foto, o operador pode visualizar a tela da vítima a partir dos dados enviados:

| Comando      | Ação   |
|--------------|--|
| PRIMEIRAFOTO | Tira screenshot da máquina da vítima e envia o tamanho da screenshot comprimida (zlib) |
| SEGUNDAFOTO  | Cria diff da screenshot atual com a anterior e envia tamanho do diff                   |
| MANDASTREAM  | Envia screenshot comprimida e diff   |

Dessa forma, percebemos que a família de malware Zumanek **trata-se de um RAT com características de Banker**, focado no mercado financeiro nacional: seja tradicional (ou seja, Bancos) ou seja no novo mercado das criptomoedas.

Interessantemente, o C&C não fica ativo a todo o tempo, o que sugere que o Zumanek segue a arquitetura clássica de “Cliente-Servidor”, onde o servidor é a máquina da vítima e o cliente é a aplicação do operador. Nesse caso, é bastante possível que o malware seja desenvolvido e comercializado por pessoas diferentes das quais operam os ataques.

Em especial, essa família emprega técnicas que vemos extensivamente utilizadas no Brasil, mas aponta também para o enfoque que o cibercrime vem tomando em torno das criptomoeadas.

## DLL Hijacking

Ao longo do ano passado, vimos uma enorme quantidade de bankers no Brasil **utilizando DLL Hijacking para executar suas ações maliciosas**. Notariamente, essa é a técnica utilizada pelo Client Maximus para se executar na máquina das vítimas.

O ataque funciona devido ao fato de que sempre que uma DLL é carregada através de LoadLibrary ou LoadLibraryEx, o sistema busca pela DLL desejada em uma certa ordem:

1. Diretório onde a aplicação foi carregada
2. Diretório System
3. Diretório System (16 bits)
4. Diretório Windows
5. Diretório de trabalho (CWD)
6. Diretórios listados em PATH

Como o Downloader se encarrega de colocar o executável (legítimo) e a DLL no mesmo diretório, quando a aplicação é carregada, a DLL terá a maior prioridade na ordem de carregamento.

Esse ataque, no entanto, [pode ser mitigado](#) tanto pelos lado dos desenvolvedores quanto dos usuários.

## Como estar seguro?

A Microsoft possui algumas recomendações que podem ser seguidas pelos desenvolvedores a fim de evitar, ou ao menos dificultar, terem suas aplicações exploradas nesse tipo de ataque (que pode acabar tendo algum tipo de impacto para a imagem da marca). Alguns exemplos simples de implementação (veja este [artigo](#) para outras recomendações):

- Validação das DLLs carregadas. Exemplos:
  - Uso de SearchPath para identificar o caminho da DLL
  - Uso de LoadLibrary para identificar a versão do sistema operacional
- Uso de caminhos absolutos para as chamadas LoadLibrary, CreateProcess e ShellExecute

Em especial, aplicações assinadas digitalmente deveriam buscar carregar apenas a DLL que também fosse assinada digitalmente. O fato de um executável digitalmente assinado poder carregar uma DLL sem assinatura abre brecha para que códigos maliciosos sejam executados em contextos “autenticados”.

Para os usuários, é possível controlar a ordem de busca de DLLs através do registro CWDIllegalInDllSearch. Já para as versões de Windows a partir do Windows Server 2012 (servidores) e Windows 8.1 (PCs), esse registro já está disponível sem necessidade da instalação do [KB2264107](https://www.welivesecurity.com/br/2018/01/17/zumanek-malware-tenta-roubar-credenciais-de-servicos/).

Scenario 1: The application is started from a local folder, such as **C:\Program Files**

| CWDIllegalInDllSearch Value | Behavior of the DLL search path in LoadLibrary and in LoadLibraryEx  |
|-----------------------------|--|
| 0xFFFFFFFF                  | Removes the current working directory from the default DLL search order  |
| 0                           | Uses the default DLL search path that was mentioned earlier  |
| 1                           | Blocks a DLL Load from the current working directory if the current working directory is set to a WebDAV folder                                    |
| 2                           | Blocks a DLL Load from the current working directory if the current working directory is set to a remote folder (such as a WebDAV or UNC location) |
| No key or other values      | Uses the default DLL search path that was mentioned earlier  |

Figura 17: Chave de registro CWDIllegalInDllSearch com valor 0xFFFFFFFF remove o diretório local de trabalho (CWD) da busca de DLL

Como vimos, o cibercrime brasileiro é bastante inovador e ativo. Portanto, é sempre importante estar atento, principalmente quando utilizamos as facilidades trazidas pelo online banking e, agora, pelas criptomoedas.

Source: <https://www.welivesecurity.com/br/2018/01/17/zumanek-malware-tenta-roubar-credenciais-de-servicos/>