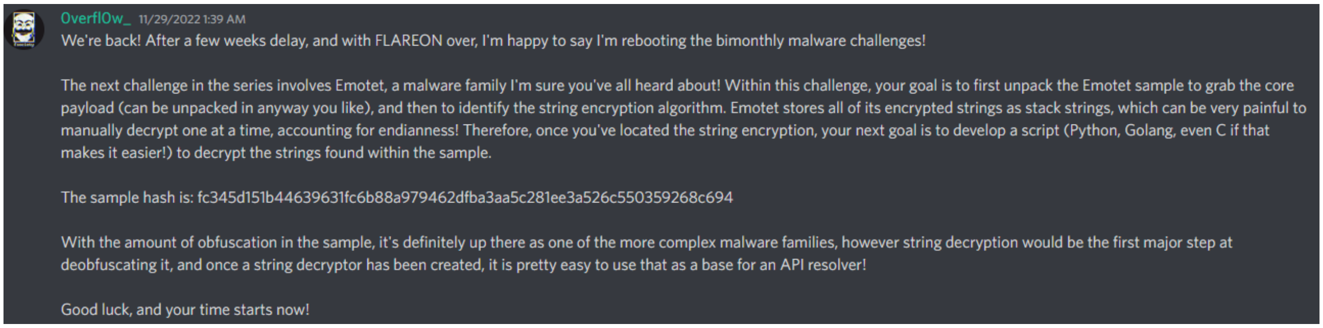
[Z2A]Bimonthly malware challege – Emotet (Back From the Dead)

kienmanowar.wordpress.com/2022/12/19/z2abimonthly-malware-challege-emotet-back-from-the-dead/

December 19, 2022

Summary



Overfl0w_ 11/29/2022 1:39 AM

We're back! After a few weeks delay, and with FLAREON over, I'm happy to say I'm rebooting the bimonthly malware challenges!

The next challenge in the series involves Emotet, a malware family I'm sure you've all heard about! Within this challenge, your goal is to first unpack the Emotet sample to grab the core payload (can be unpacked in anyway you like), and then to identify the string encryption algorithm. Emotet stores all of its encrypted strings as stack strings, which can be very painful to manually decrypt one at a time, accounting for endianness! Therefore, once you've located the string encryption, your next goal is to develop a script (Python, Golang, even C if that makes it easier!) to decrypt the strings found within the sample.

The sample hash is: fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694

With the amount of obfuscation in the sample, it's definitely up there as one of the more complex malware families, however string decryption would be the first major step at deobfuscating it, and once a string decryptor has been created, it is pretty easy to use that as a base for an API resolver!

Good luck, and your time starts now!

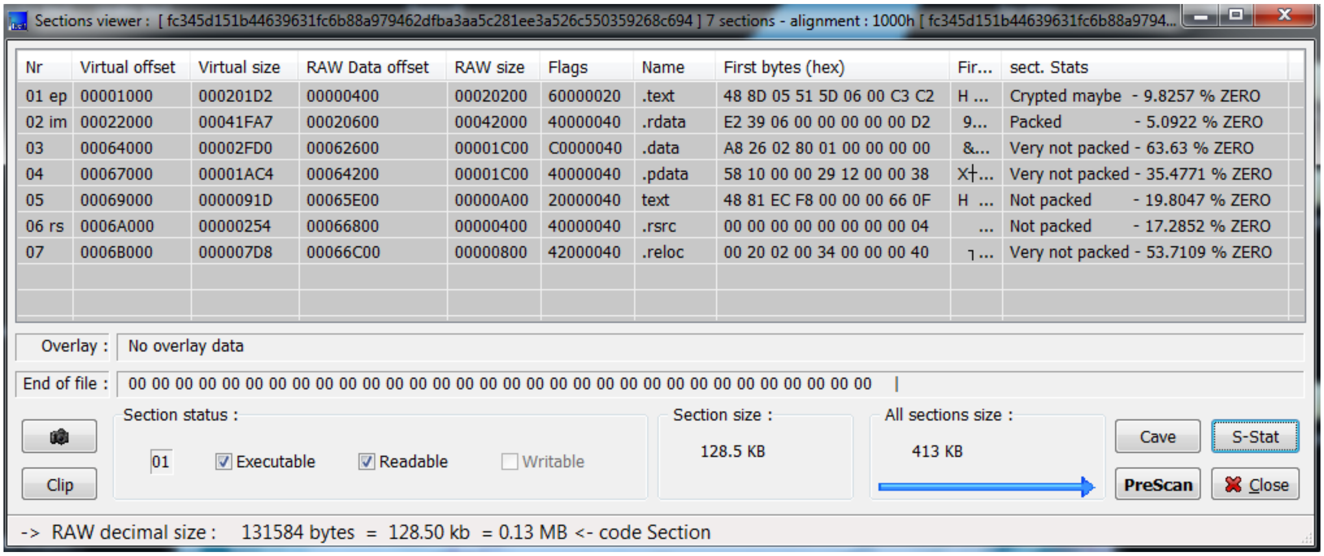
Sample hash is:
fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694

This write-up of mine will be divided into three parts:

- Grab core Emotet Dll payload.
- Recover API functions that used by core payload.
- Decrypt strings

1. Grab core payload

A quick check of information related to sections of this sample shows that it may be crypted/packed to conceal the real malware inside the original sample, besides there is an extra section with an unusual name: **text**



Sections viewer : [fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694] 7 sections - alignment : 1000h [fc345d151b44639631fc6b88a9794...

Nr	Virtual offset	Virtual size	RAW Data offset	RAW size	Flags	Name	First bytes (hex)	Fir...	sect. Stats
01 ep	00001000	000201D2	00000400	00020200	60000020	.text	48 8D 05 51 5D 06 00 C3 C2	H ...	Crypted maybe - 9.8257 % ZERO
02 im	00022000	00041FA7	00020600	00042000	40000040	.rdata	E2 39 06 00 00 00 00 00 D2	9...	Packed - 5.0922 % ZERO
03	00064000	00002FD0	00062600	00001C00	C0000040	.data	A8 26 02 80 01 00 00 00 00	&...	Very not packed - 63.63 % ZERO
04	00067000	00001AC4	00064200	00001C00	40000040	.pdata	58 10 00 00 29 12 00 00 38	X┼...	Very not packed - 35.4771 % ZERO
05	00069000	0000091D	00065E00	00000A00	20000040	text	48 81 EC F8 00 00 00 66 0F	H ...	Not packed - 19.8047 % ZERO
06 rs	0006A000	00000254	00066800	00000400	40000040	.rsrc	00 00 00 00 00 00 00 00 04	...	Not packed - 17.2852 % ZERO
07	0006B000	000007D8	00066C00	00000800	42000040	.reloc	00 20 02 00 34 00 00 00 40	┐...	Very not packed - 53.7109 % ZERO

Overlay : No overlay data

End of file : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

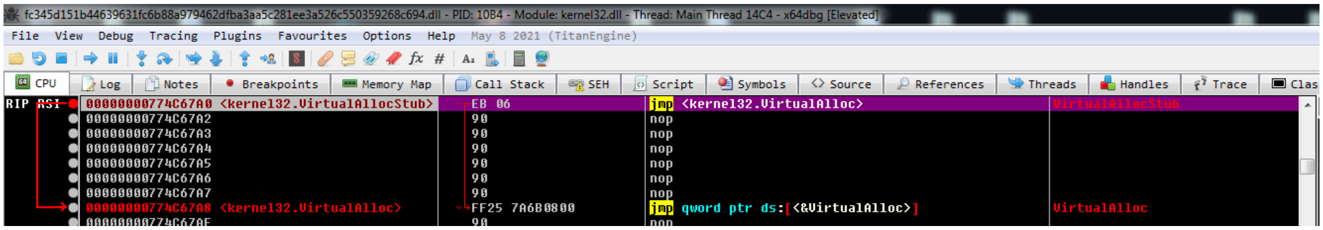
Section status : 01 ☑ Executable ☑ Readable ☐ Writable

Section size : 128.5 KB

All sections size : 413 KB

-> RAW decimal size : 131584 bytes = 128.50 kb = 0.13 MB <- code Section

Load the sample into x64dbg, set a breakpoint at the **VirtualAlloc** API function, run payload by press **F9**. It will break at the **VirtualAlloc** function:

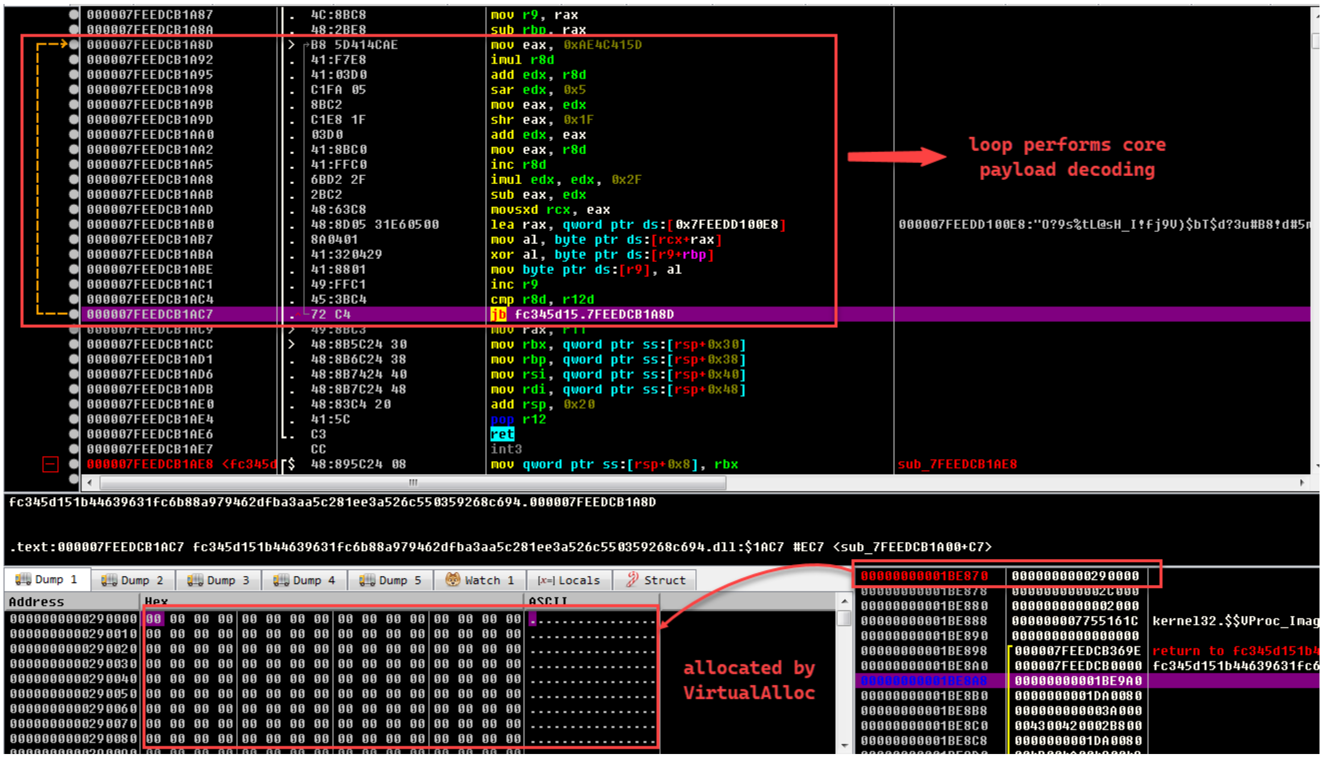


fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694.dll - PID: 10B4 - Module: kernel32.dll - Thread: Main Thread 14C4 - x64dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help May 8 2021 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace Clas

```
RIP RSI 00000000774C67A0 <kernel32.VirtualAllocStub>  EB 06          jmp <kernel32.VirtualAlloc>          VirtualAllocStub
        00000000774C67A2                               90             nop
        00000000774C67A3                               90             nop
        00000000774C67A4                               90             nop
        00000000774C67A5                               90             nop
        00000000774C67A6                               90             nop
        00000000774C67A7                               90             nop
        00000000774C67A8 <kernel32.VirtualAlloc>      FF25 7A6B0800  jmp qword ptr ds:[<&VirtualAlloc>]   VirtualAlloc
        00000000774C67AE                               90             nop
```

Execute till return (**Ctrl+F9**) and follow the allocated memory, trace over the `ret` instruction to return the Dll’s code will reach the code area like the following:

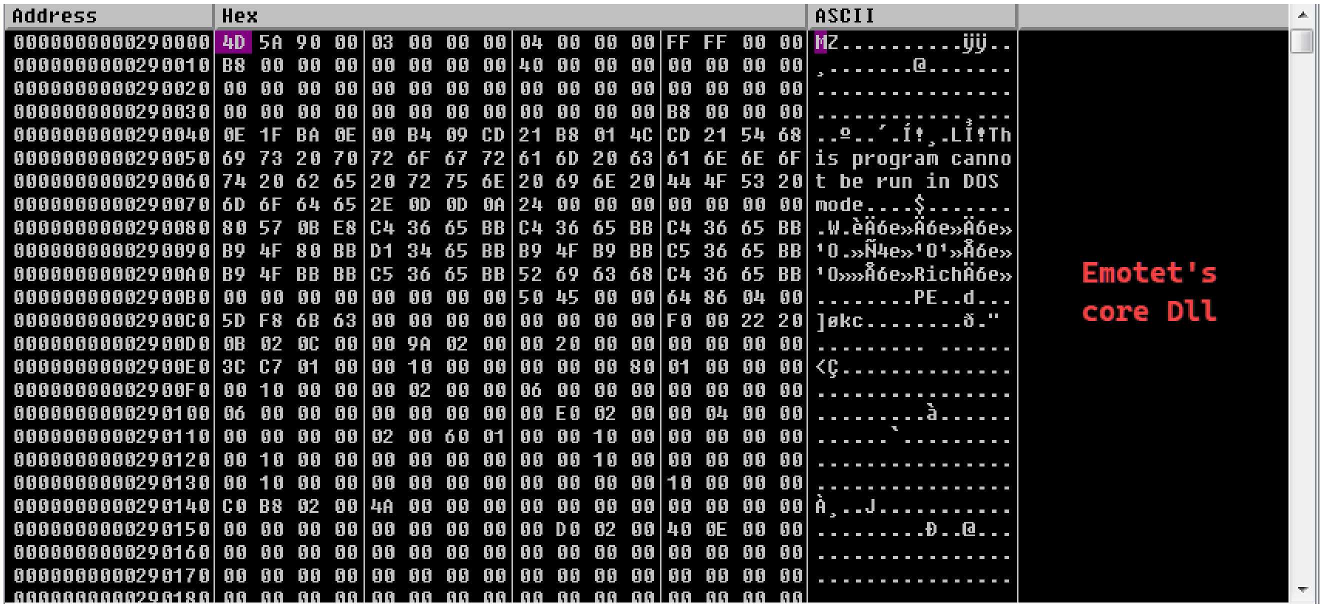


```
000007FEEDCB1A87      .  4C:8BC8             mov r9, rax
000007FEEDCB1A8A      .  48:2BE8             sub rbp, rax
000007FEEDCB1A8D      >  B8 5D414CAE         mov eax, 0xAE4C415D
000007FEEDCB1A92      .  41:F7E8             imul r8d
000007FEEDCB1A95      .  41:03D0             add edx, r8d
000007FEEDCB1A98      .  C1FA 05             sar edx, 0x5
000007FEEDCB1A9B      .  8BC2                mov eax, edx
000007FEEDCB1A9D      .  C1E8 1F             shr eax, 0x1F
000007FEEDCB1AA0      .  03D0                add edx, eax
000007FEEDCB1AA2      .  41:8BC0             mov eax, r8d
000007FEEDCB1AA5      .  41:FFC0             inc r8d
000007FEEDCB1AA8      .  6BD2 2F             imul edx, edx, 0x2F
000007FEEDCB1AAB      .  2BC2                sub eax, edx
000007FEEDCB1AAD      .  48:63C8             movsxd rcx, eax
000007FEEDCB1AB0      .  48:8D05 31E60500    lea rax, qword ptr ds:[0x7FEEDD100E8]     000007FEEDD100E8:"O?9s%tL@sH_I!fj9V)$bT$d?3u#B8!d#5m
000007FEEDCB1AB7      .  8A0401              mov al, byte ptr ds:[rcx+rax]
000007FEEDCB1ABA      .  41:320429           xor al, byte ptr ds:[r9+rbp]
000007FEEDCB1ABE      .  41:8801             mov byte ptr ds:[r9], al
000007FEEDCB1AC1      .  49:FFC1             inc r9
000007FEEDCB1AC4      .  45:3BC4             cmp r8d, r12d
000007FEEDCB1AC7      .  72 C4               jb fc345d15.7FEEDCB1A8D
000007FEEDCB1AC9      >  49:8BC3             mov rax, r11
000007FEEDCB1ACC      >  48:8B5C24 30        mov rbx, qword ptr ss:[rsp+0x30]
000007FEEDCB1AD1      .  48:8B6C24 38        mov rbp, qword ptr ss:[rsp+0x38]
000007FEEDCB1AD6      .  48:8B7424 40        mov rsi, qword ptr ss:[rsp+0x40]
000007FEEDCB1ADB      .  48:8B7C24 48        mov rdi, qword ptr ss:[rsp+0x48]
000007FEEDCB1AE0      .  48:83C4 20          add rsp, 0x20
000007FEEDCB1AE4      .  41:5C               pop r12
000007FEEDCB1AE6      .  C3                  ret
000007FEEDCB1AE7         CC                  int3
000007FEEDCB1AE8 <fc345d $ 48:895C24 08       mov qword ptr ss:[rsp+0x8], rbx           sub_7FEEDCB1AE8

fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694.000007FEEDCB1A8D

.text:000007FEEDCB1AC7 fc345d151b44639631fc6b88a979462dfba3aa5c281ee3a526c550359268c694.dll:$1AC7 #EC7 <sub_7FEEDCB1A00+C7>
```

```
Address           Hex                                                ASCII
0000000000290000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000000000290080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

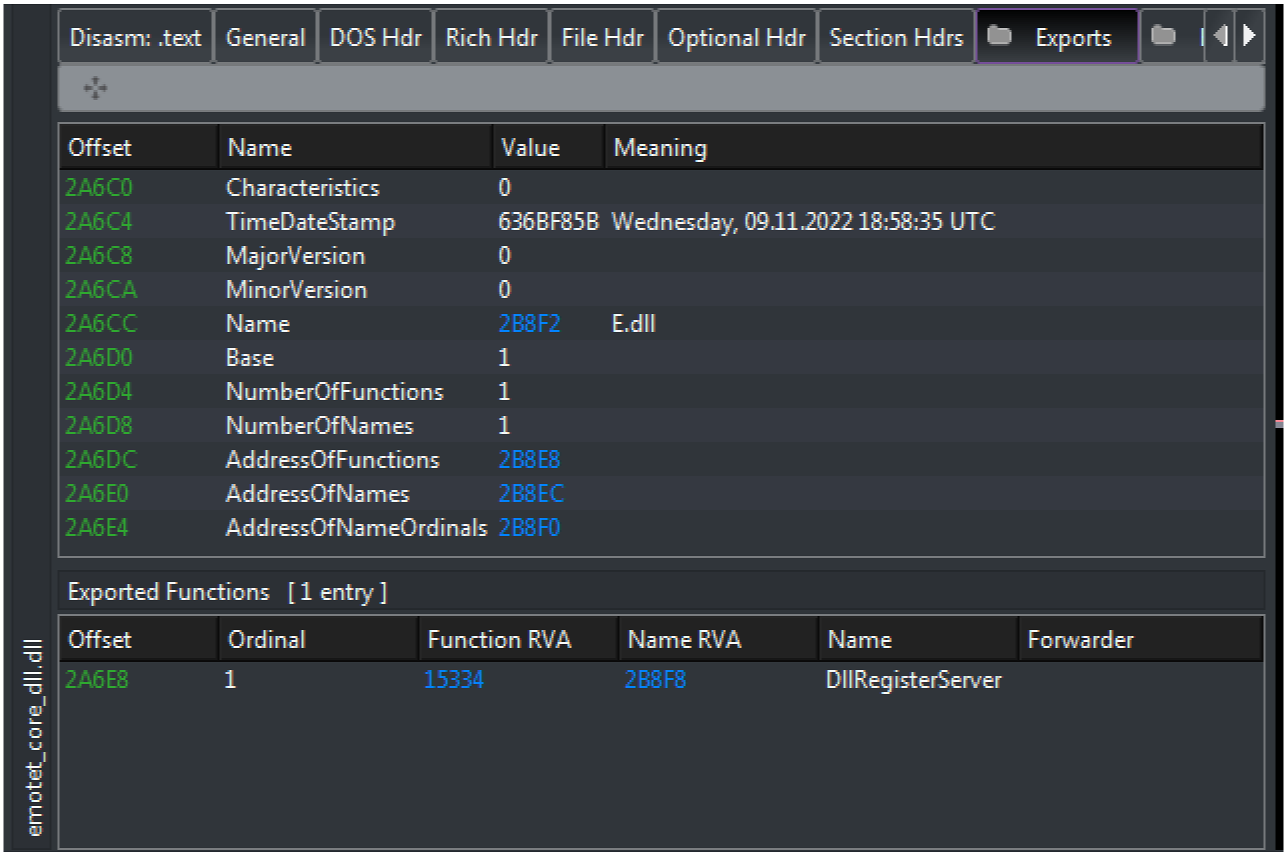
```
000000000001BE870  0000000000290000
000000000001BE878  000000000002C000
000000000001BE880  0000000000002000
000000000001BE888  000000007755161C  kernel32.$$VProc_Imag
000000000001BE890  0000000000000000
000000000001BE898  000007FEEDCB369E  return to fc345d151b4
000000000001BE8A0  000007FEEDCB0000  fc345d151b44639631fc6
000000000001BE8A8  000000000001BE9A0
000000000001BE8B0  000000001DA0080
000000000001BE8B8  00000000003A000
000000000001BE8C0  00430042002B800
000000000001BE8C8  000000001DA0080
```

To quickly get the Emotet core payload, set a bp at the `ret` command below the loop, then press **F9** to let the payload finish decrypting and fill core payload content to the allocated memory. The resulting core payload is decrypted as shown below:



```
Address          Hex                                              ASCII
0000000000290000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
0000000000290010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ¸.......@.......
0000000000290020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000000000290030 00 00 00 00 00 00 00 00 00 00 00 00 B8 00 00 00 ............¸...
0000000000290040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..º..´.Í!¸.LÍ!Th
0000000000290050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
0000000000290060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS 
0000000000290070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
0000000000290080 80 57 0B E8 C4 36 65 BB C4 36 65 BB C4 36 65 BB .W.èÄ6e»Ä6e»Ä6e»
0000000000290090 B9 4F 80 BB D1 34 65 BB B9 4F B9 BB C5 36 65 BB ¹O.»Ñ4e»¹O¹»Å6e»
00000000002900A0 B9 4F BB BB C5 36 65 BB 52 69 63 68 C4 36 65 BB ¹O»»Å6e»RichÄ6e»
00000000002900B0 00 00 00 00 00 00 00 00 50 45 00 00 64 86 04 00 ........PE..d...
00000000002900C0 5D F8 6B 63 00 00 00 00 00 00 00 00 F0 00 22 20 ]økc........ð." 
00000000002900D0 0B 02 0C 00 00 9A 02 00 00 20 00 00 00 00 00 00 .......... .....
00000000002900E0 3C C7 01 00 00 10 00 00 00 00 00 80 01 00 00 00 <Ç..............
00000000002900F0 00 10 00 00 00 02 00 00 06 00 00 00 00 00 00 00 ................
0000000000290100 06 00 00 00 00 00 00 00 00 E0 02 00 00 04 00 00 .........à......
0000000000290110 00 00 00 00 02 00 60 01 00 00 10 00 00 00 00 00 ......`.........
0000000000290120 00 10 00 00 00 00 00 00 00 00 10 00 00 00 00 00 ................
0000000000290130 00 10 00 00 00 00 00 00 00 00 00 00 10 00 00 00 ................
0000000000290140 C0 B8 02 00 4A 00 00 00 00 00 00 00 00 00 00 00 À¸..J...........
0000000000290150 00 00 00 00 00 00 00 00 00 D0 02 00 40 0E 00 00 .........Ð..@...
0000000000290160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000000000290170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000000000290180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Now, dump the above memory to disk, then fix total size of the payload to `0x2B800`, we get the final Emotet core Dll (Md5: *577118e39051f0678a52f871f74cd675*):



Offset	Name	Value	Meaning
2A6C0	Characteristics	0	
2A6C4	TimeDateStamp	636BF85B	Wednesday, 09.11.2022 18:58:35 UTC
2A6C8	MajorVersion	0	
2A6CA	MinorVersion	0	
2A6CC	Name	2B8F2	E.dll
2A6D0	Base	1	
2A6D4	NumberOfFunctions	1	
2A6D8	NumberOfNames	1	
2A6DC	AddressOfFunctions	2B8E8	
2A6E0	AddressOfNames	2B8EC	
2A6E4	AddressOfNameOrdinals	2B8F0	

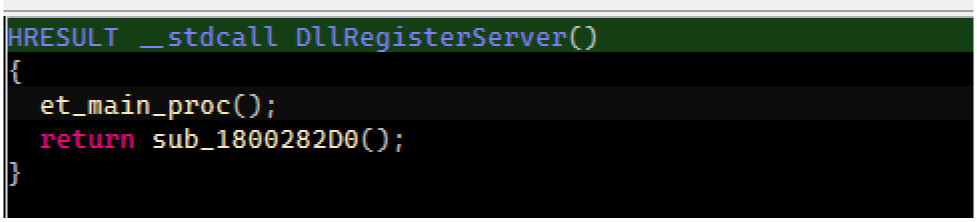
Exported Functions [1 entry]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
2A6E8	1	15334	2B8F8	DllRegisterServer	

2. API resolver

2.1. Recover Dll name from pre-calculated hash

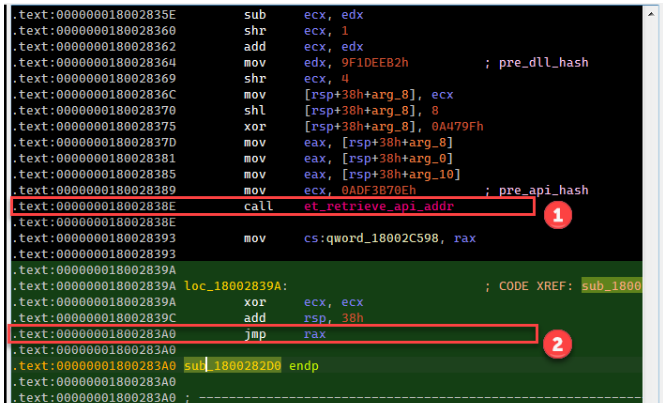
Load fixed core Dll above into IDA, go to the export function `DllRegisterServer` we see there are 2 sub routines as follows:



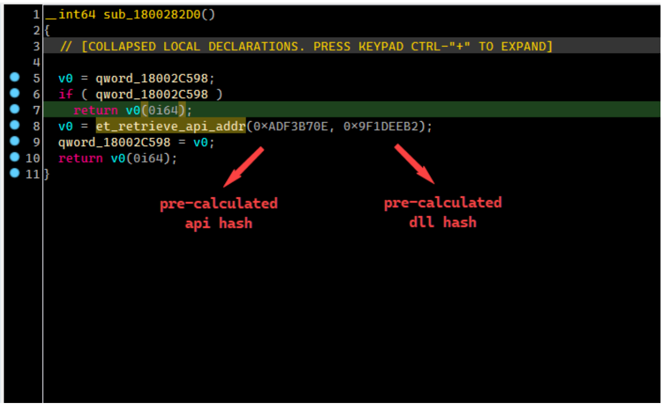
```
HRESULT __stdcall DllRegisterServer()
{
  et_main_proc();
  return sub_1800282D0();
}
```

At `sub_1800282D0`, Emotet will perform:

1. Get the address of the API function based on the pre-computed hash value.
2. Jump to the API function to execute.



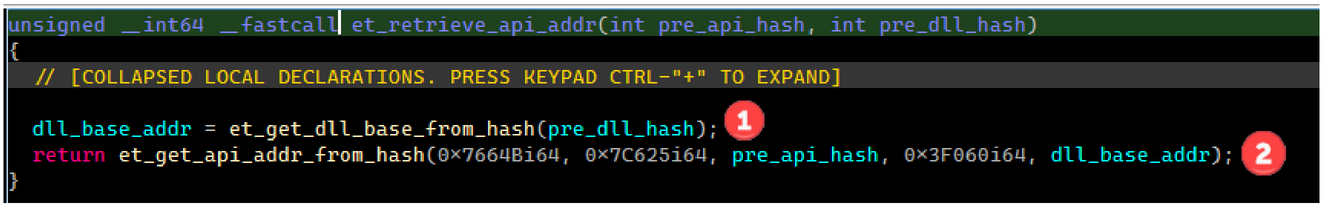
```
.text:000000018002835E                 sub     ecx, edx
.text:0000000180028360                 shr     ecx, 1
.text:0000000180028362                 add     ecx, edx
.text:0000000180028364                 mov     edx, 9F1DEEB2h          ; pre_dll_hash
.text:0000000180028369                 shr     ecx, 4
.text:000000018002836C                 mov     [rsp+38h+arg_8], ecx
.text:0000000180028370                 shl     [rsp+38h+arg_8], 8
.text:0000000180028375                 xor     [rsp+38h+arg_8], 0A479Fh
.text:000000018002837D                 mov     eax, [rsp+38h+arg_8]
.text:0000000180028381                 mov     eax, [rsp+38h+arg_0]
.text:0000000180028385                 mov     eax, [rsp+38h+arg_10]
.text:0000000180028389                 mov     ecx, 0ADF3B70Eh         ; pre_api_hash
.text:000000018002838E                 call    et_retrieve_api_addr
.text:000000018002838E
.text:0000000180028393                 mov     cs:qword_18002C598, rax
.text:0000000180028393
.text:000000018002839A
.text:000000018002839A loc_18002839A:                          ; CODE XREF: sub_1800
.text:000000018002839A                 xor     ecx, ecx
.text:000000018002839C                 add     rsp, 38h
.text:00000001800283A0                 jmp     rax
.text:00000001800283A0
.text:00000001800283A0 sub_1800282D0 endp
.text:00000001800283A0
.text:00000001800283A0 ; ---------------------------------------------------------
```



```
__int64 sub_1800282D0()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = qword_18002C598;
  if ( qword_18002C598 )
    return v0(0i64);
  v0 = et_retrieve_api_addr(0xADF3B70E, 0x9F1DEEB2);
  qword_18002C598 = v0;
  return v0(0i64);
}
```

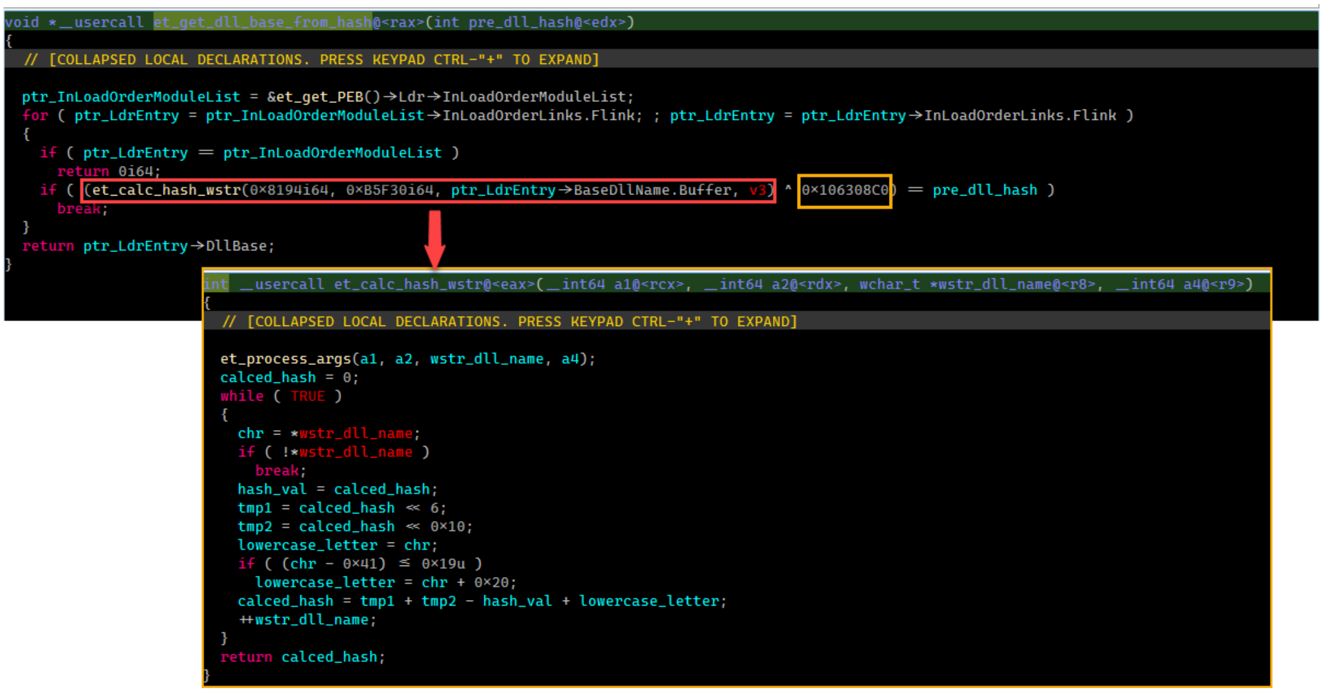
At `et_retrieve_api_addr (0x18000F174)` function, the code snippet does the following:

1. Retrieve the base address of the Dll based on the pre-computed hash value.
2. Retrieve the address of the API function belong to the Dll above.



```
unsigned __int64 __fastcall et_retrieve_api_addr(int pre_api_hash, int pre_dll_hash)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  dll_base_addr = et_get_dll_base_from_hash(pre_dll_hash);
  return et_get_api_addr_from_hash(0x7664Bi64, 0x7C625i64, pre_api_hash, 0x3F060i64, dll_base_addr);
}
```

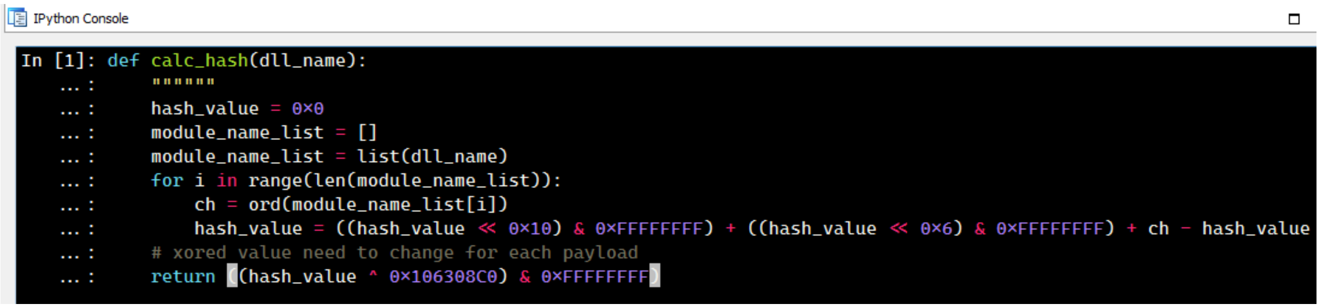
Continuing to dive into the `et_get_dll_base_from_hash (0x0180002960)` function, the process of getting the base address of the Dll will be as follows:



```
void *__usercall et_get_dll_base_from_hash@<rax>(int pre_dll_hash@<edx>)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  ptr_InLoadOrderModuleList = &et_get_PEB()→Ldr→InLoadOrderModuleList;
  for ( ptr_LdrEntry = ptr_InLoadOrderModuleList→InLoadOrderLinks.Flink; ; ptr_LdrEntry = ptr_LdrEntry→InLoadOrderLinks.Flink )
  {
    if ( ptr_LdrEntry == ptr_InLoadOrderModuleList )
      return 0i64;
    if ( (et_calc_hash_wstr(0×8194i64, 0×B5F30i64, ptr_LdrEntry→BaseDllName.Buffer, v3) ^ 0×106308C0) == pre_dll_hash )
      break;
  }
  return ptr_LdrEntry→DllBase;
}
```

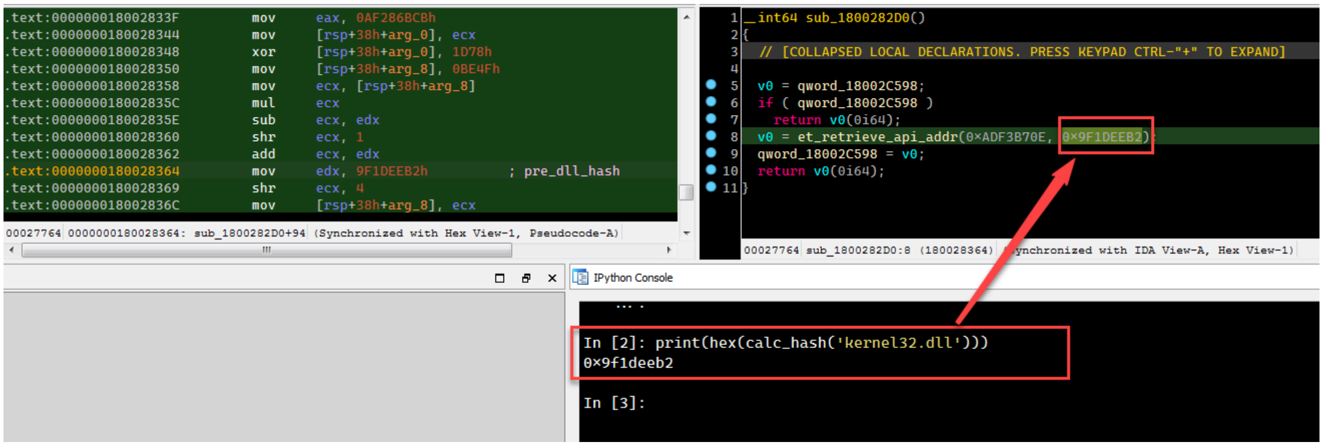
```
int __usercall et_calc_hash_wstr@<eax>(__int64 a1@<rcx>, __int64 a2@<rdx>, wchar_t *wstr_dll_name@<r8>, __int64 a4@<r9>)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  et_process_args(a1, a2, wstr_dll_name, a4);
  calced_hash = 0;
  while ( TRUE )
  {
    chr = *wstr_dll_name;
    if ( !*wstr_dll_name )
      break;
    hash_val = calced_hash;
    tmp1 = calced_hash ≪ 6;
    tmp2 = calced_hash ≪ 0×10;
    lowercase_letter = chr;
    if ( (chr - 0×41) ≤ 0×19u )
      lowercase_letter = chr + 0×20;
    calced_hash = tmp1 + tmp2 - hash_val + lowercase_letter;
    ++wstr_dll_name;
  }
  return calced_hash;
}
```

Based on the above pseudocode, rewrite the hash function in Python for the name of the Dll as follows:



```
In [1]: def calc_hash(dll_name):
   ...:     """"""
   ...:     hash_value = 0x0
   ...:     module_name_list = []
   ...:     module_name_list = list(dll_name)
   ...:     for i in range(len(module_name_list)):
   ...:         ch = ord(module_name_list[i])
   ...:         hash_value = ((hash_value << 0x10) & 0xFFFFFFFF) + ((hash_value << 0x6) & 0xFFFFFFFF) + ch - hash_value
   ...:     # xored value need to change for each payload
   ...:     return ((hash_value ^ 0x106308C0) & 0xFFFFFFFF)
```

Let's check again with the name of the Dll is `kernel32.dll`:



```
.text:000000018002833F                 mov     eax, 0AF286BCBh
.text:0000000180028344                 mov     [rsp+38h+arg_0], ecx
.text:0000000180028348                 xor     [rsp+38h+arg_0], 1D78h
.text:0000000180028350                 mov     [rsp+38h+arg_8], 0BE4Fh
.text:0000000180028358                 mov     ecx, [rsp+38h+arg_8]
.text:000000018002835C                 mul     ecx
.text:000000018002835E                 sub     ecx, edx
.text:0000000180028360                 shr     ecx, 1
.text:0000000180028362                 add     ecx, edx
.text:0000000180028364                 mov     edx, 9F1DEEB2h          ; pre_dll_hash
.text:0000000180028369                 shr     ecx, 4
.text:000000018002836C                 mov     [rsp+38h+arg_8], ecx
00027764 0000000180028364: sub_1800282D0+94 (Synchronized with Hex View-1, Pseudocode-A)
```

```
__int64 sub_1800282D0()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = qword_18002C598;
  if ( qword_18002C598 )
    return v0(0i64);
  v0 = et_retrieve_api_addr(0×ADF3B70E, 0×9F1DEEB2);
  qword_18002C598 = v0;
  return v0(0i64);
}
00027764 sub_1800282D0:8 (180028364) (Synchronized with IDA View-A, Hex View-1)
```

```
   ...:
In [2]: print(hex(calc_hash('kernel32.dll')))
0x9f1deeb2

In [3]:
```

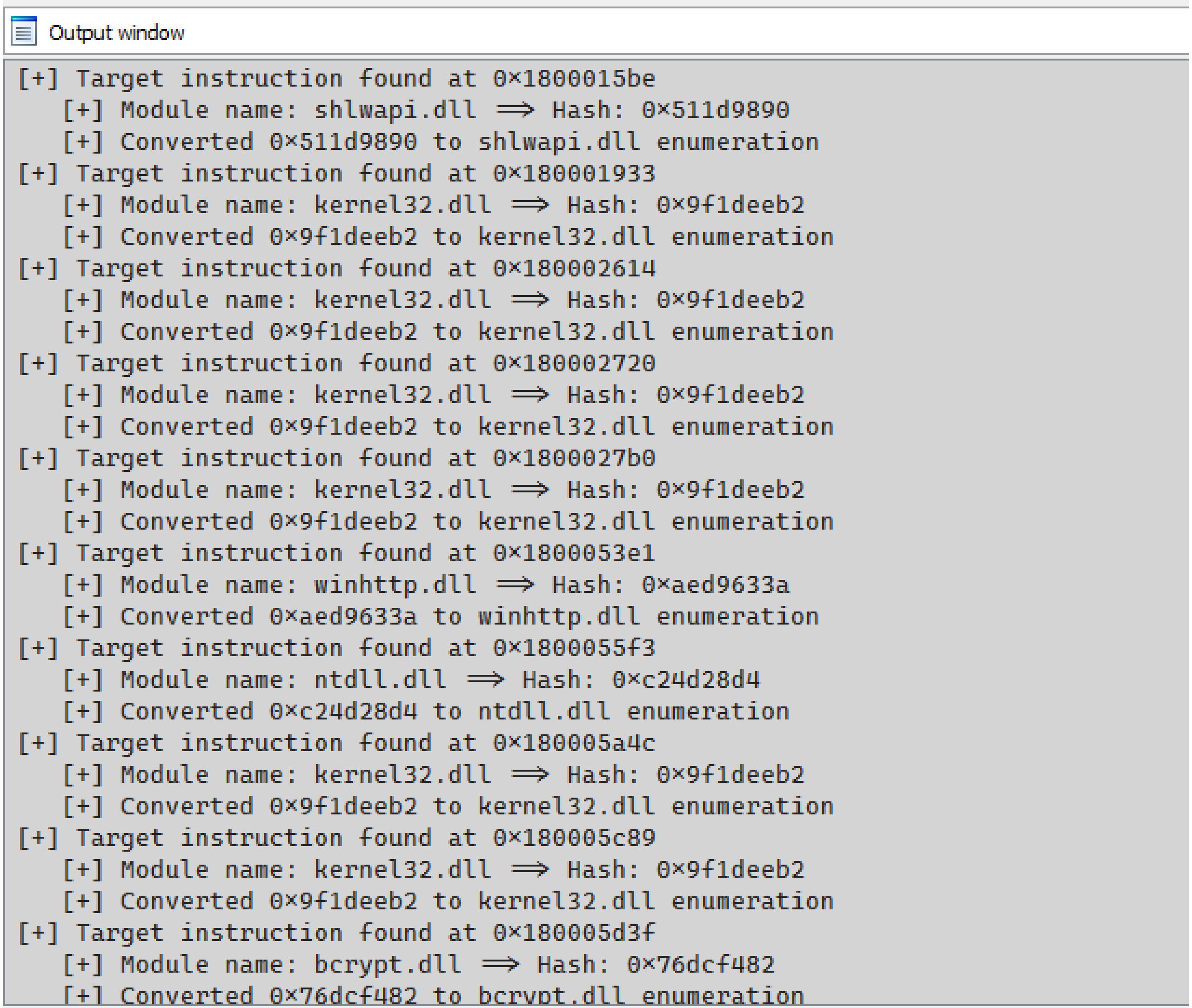
We can write an IDAPython script that recovers the names of the DLLs that Emotet uses from these pre-computed hashes. The script performs the following tasks:

1. Iterate all addresses refer to `et_retrieve_api_addr` function.
2. Find the address of the instruction that assigns the hash value of the Dll name and retrieve this hash value.

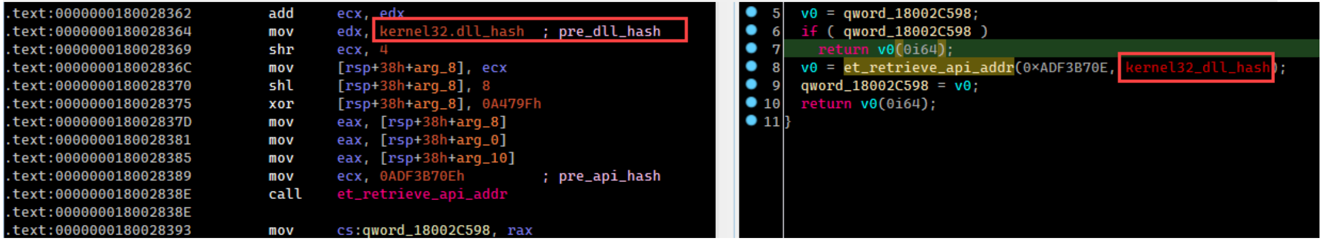
3. Calculate the hash value based on the list of common DLL names, then compare the calculated hash value with the hash value obtained in the previous step.
4. If equal, create a new enumeration that will store the hash-to-dll-name mapping, then convert this hash value back to the name of the Dll.

```
import idc, ida_enum, idautils, ida_bytes, idaapi, ida_bytes

most_common_dlls =
['kernel32.dll','user32.dll','ntdll.dll','shlwapi.dll','iphlpapi.dll','urlmon.dll','ws
 'comctl32.dll', 'comdlg32.dll', 'msvcrt.dll', 'oleaut32.dll', 'srsvc.dll',
'winhttp.dll', 'advpack.dll', 'combase.dll', 'ntoskrnl.exe']

#----------------------------------------------------------------------
def calc_hash(dll_name):
    """"""
    hash_value = 0x0
    module_name_list = []
    module_name_list = list(dll_name)
    for i in range(len(module_name_list)):
        ch = ord(module_name_list[i])
        hash_value = ((hash_value << 0x10) & 0xFFFFFFFF) + ((hash_value << 0x6) &
0xFFFFFFFF) + ch - hash_value
    # xored value need to change for each payload
    return ((hash_value ^ 0x106308C0) & 0xFFFFFFFF)

#----------------------------------------------------------------------
def get_enum_const(constant):
    """"""
    all_enums = ida_enum.get_enum_qty()
    for i in range(0, all_enums):
        enum_id = ida_enum.getn_enum(i)
        mask = ida_enum.get_first_bmask(enum_id)
        enum_constant = ida_enum.get_first_enum_member(enum_id, mask)
        name = ida_enum.get_enum_member_name(ida_enum.get_enum_member(enum_id,
enum_constant, 0, mask))
        if int(enum_constant) == constant: return [name, enum_id]
        while True:
            enum_constant = ida_enum.get_next_enum_member(enum_id, enum_constant,
mask)
            name = ida_enum.get_enum_member_name(ida_enum.get_enum_member(enum_id,
enum_constant, 0, mask))
            if enum_constant == 0xFFFFFFFF:
                break
            if int(enum_constant) == constant: return [name, enum_id]
    return None

#----------------------------------------------------------------------
def convert_offset_to_enum(addr):
    """"""
    n_operand = 0
    if idc.print_insn_mnem(addr) == "push":
        constant = idc.get_operand_value(addr, 0) & 0xFFFFFFFF
    elif idc.print_insn_mnem(addr) == "mov":
        constant = idc.get_operand_value(addr, 1) & 0xFFFFFFFF
        n_operand = 1
    enum_data = get_enum_const(constant)
    if enum_data:
        name, enum_id = enum_data
        idc.op_enum(addr, n_operand, enum_id, 0)
        return True
    else:
        return False

#----------------------------------------------------------------------
def enum_for_xrefs(func_addr, eid):
    """"""
    for x in idautils.XrefsTo(func_addr, flags=0):
        call_address = x.frm
        if ida_bytes.is_code(ida_bytes.get_full_flags(call_address)):
            #retrieve address of the instruction that assigns the Dll's hash value to
the variable
            pre_module_hash_addr = idaapi.get_arg_addrs(call_address)[1]

            if idc.print_insn_mnem(pre_module_hash_addr) == "mov" and
idc.get_operand_type(pre_module_hash_addr, 1) == idc.o_imm:
                print ("[+] Target instruction found at
0x{address:x}".format(address=pre_module_hash_addr))
                pre_module_hash = idc.get_operand_value(pre_module_hash_addr, 1) &
0xFFFFFFFF
                module_hash_addr = pre_module_hash_addr

            for dll_name in most_common_dlls:
                calced_hash = calc_hash(dll_name)
                if calced_hash == pre_module_hash:
                    print ('   [+] Module name: %s ==> Hash: 0x%x' %(dll_name,
calced_hash))
                    ida_enum.add_enum_member(eid, '%s_hash' % dll_name,
int(calced_hash), idaapi.BADADDR)
                    if convert_offset_to_enum(module_hash_addr):
                        print ("   [+] Converted 0x%x to %s enumeration" %
(idc.get_operand_value(module_hash_addr, 1) & 0xFFFFFFFF, dll_name))

#----------------------------------------------------------------------
def main():
    """"""
    target_function = 0x018000F174 #change address of function
    '''Adds enum name'''
    if ida_enum.get_enum("MODULE_HASHES") != 0xffffffffffffffff:
        print('Enum already exists ...')
        return 0xffffffffffffffff
    else:
        eid = ida_enum.add_enum(0, "MODULE_HASHES", ida_bytes.hex_flag())

    enum_for_xrefs(target_function, eid)

if __name__ == '__main__':
    main()
```

The following figures is the result after executing the script:



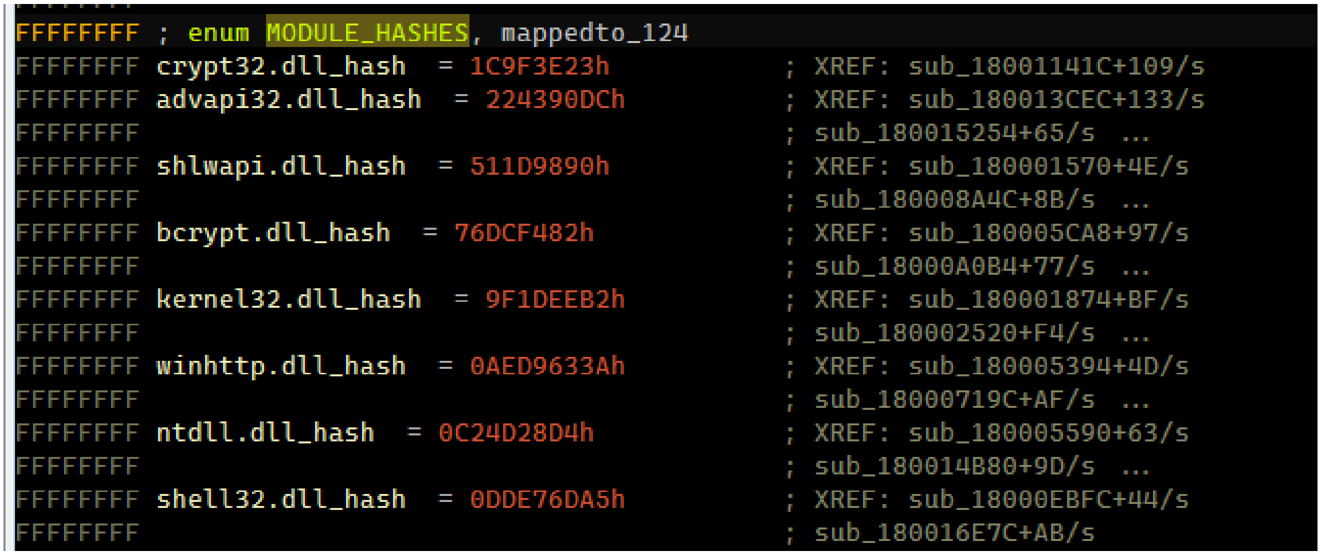
```
[+] Target instruction found at 0x1800015be
    [+] Module name: shlwapi.dll ⟹ Hash: 0x511d9890
    [+] Converted 0x511d9890 to shlwapi.dll enumeration
[+] Target instruction found at 0x180001933
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x180002614
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x180002720
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x1800027b0
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x1800053e1
    [+] Module name: winhttp.dll ⟹ Hash: 0xaed9633a
    [+] Converted 0xaed9633a to winhttp.dll enumeration
[+] Target instruction found at 0x1800055f3
    [+] Module name: ntdll.dll ⟹ Hash: 0xc24d28d4
    [+] Converted 0xc24d28d4 to ntdll.dll enumeration
[+] Target instruction found at 0x180005a4c
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x180005c89
    [+] Module name: kernel32.dll ⟹ Hash: 0x9f1deeb2
    [+] Converted 0x9f1deeb2 to kernel32.dll enumeration
[+] Target instruction found at 0x180005d3f
    [+] Module name: bcrypt.dll ⟹ Hash: 0x76dcf482
    [+] Converted 0x76dcf482 to bcrypt.dll enumeration
```



```
.text:0000000180028362         add     ecx, edx
.text:0000000180028364         mov     edx, kernel32.dll_hash  ; pre_dll_hash
.text:0000000180028369         shr     ecx, 4
.text:000000018002836C         mov     [rsp+38h+arg_8], ecx
.text:0000000180028370         shl     [rsp+38h+arg_8], 8
.text:0000000180028375         xor     [rsp+38h+arg_8], 0A479Fh
.text:000000018002837D         mov     eax, [rsp+38h+arg_8]
.text:0000000180028381         mov     eax, [rsp+38h+arg_0]
.text:0000000180028385         mov     eax, [rsp+38h+arg_10]
.text:0000000180028389         mov     ecx, 0ADF3B70Eh         ; pre_api_hash
.text:000000018002838E         call    et_retrieve_api_addr
.text:000000018002838E
.text:0000000180028393         mov     cs:qword_18002C598, rax
```

```
  v0 = qword_18002C598;
  if ( qword_18002C598 )
    return v0(0i64);
  v0 = et_retrieve_api_addr(0xADF3B70E, kernel32_dll_hash);
  qword_18002C598 = v0;
  return v0(0i64);
}
```

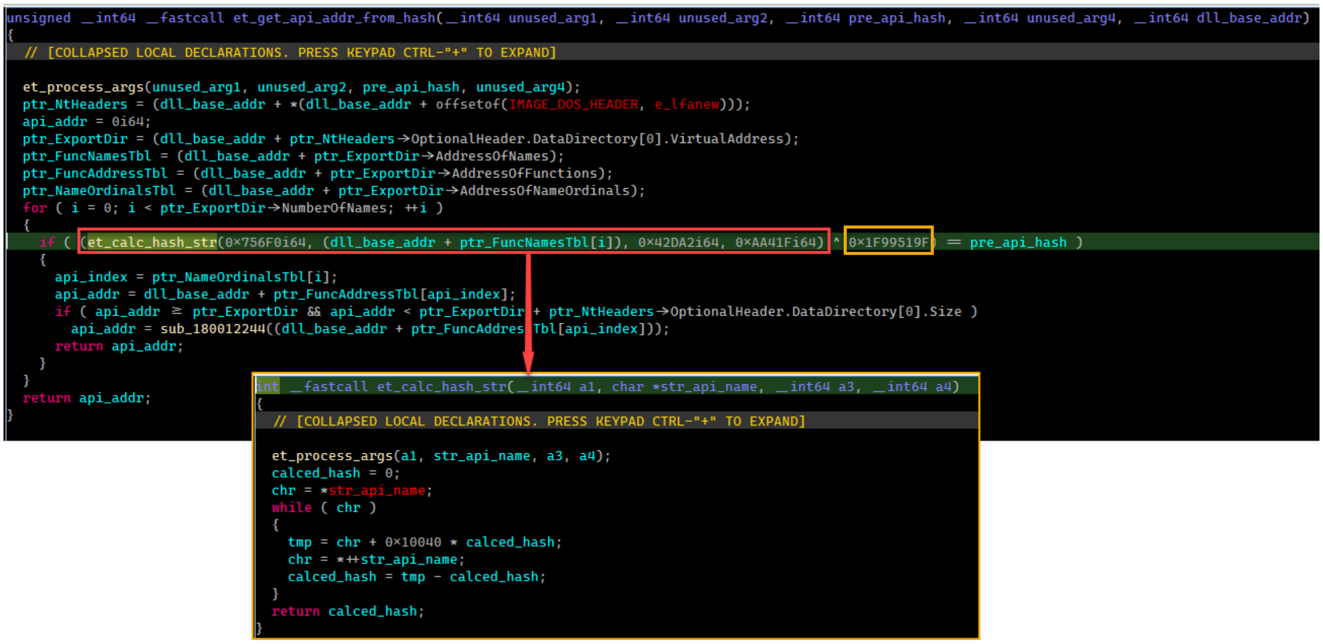
We get the full list of Dlls that Emotet will use during execution:



```
FFFFFFFF ; enum MODULE_HASHES, mappedto_124
FFFFFFFF crypt32.dll_hash  = 1C9F3E23h          ; XREF: sub_18001141C+109/s
FFFFFFFF advapi32.dll_hash  = 224390DCh         ; XREF: sub_180013CEC+133/s
FFFFFFFF                                        ; sub_180015254+65/s ...
FFFFFFFF shlwapi.dll_hash  = 511D9890h          ; XREF: sub_180001570+4E/s
FFFFFFFF                                        ; sub_180008A4C+8B/s ...
FFFFFFFF bcrypt.dll_hash  = 76DCF482h           ; XREF: sub_180005CA8+97/s
FFFFFFFF                                        ; sub_18000A0B4+77/s ...
FFFFFFFF kernel32.dll_hash  = 9F1DEEB2h         ; XREF: sub_180001874+BF/s
FFFFFFFF                                        ; sub_180002520+F4/s ...
FFFFFFFF winhttp.dll_hash  = 0AED9633Ah         ; XREF: sub_180005394+4D/s
FFFFFFFF                                        ; sub_18000719C+AF/s ...
FFFFFFFF ntdll.dll_hash  = 0C24D28D4h           ; XREF: sub_180005590+63/s
FFFFFFFF                                        ; sub_180014B80+9D/s ...
FFFFFFFF shell32.dll_hash  = 0DDE76DA5h         ; XREF: sub_18000EBFC+44/s
FFFFFFFF                                        ; sub_180016E7C+AB/s
```

2.2. Recover API name from pre-calculated hash

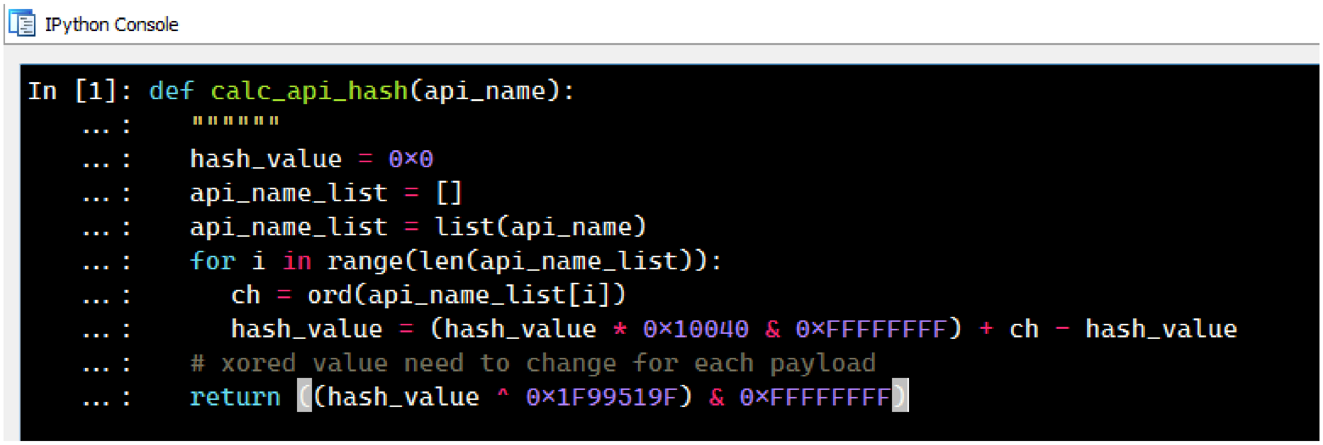
The pseudocode at the `et_get_api_addr_from_hash (0x0180025D84)` function does the following task:



```
unsigned __int64 __fastcall et_get_api_addr_from_hash(__int64 unused_arg1, __int64 unused_arg2, __int64 pre_api_hash, __int64 unused_arg4, __int64 dll_base_addr)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  et_process_args(unused_arg1, unused_arg2, pre_api_hash, unused_arg4);
  ptr_NtHeaders = (dll_base_addr + *(dll_base_addr + offsetof(IMAGE_DOS_HEADER, e_lfanew)));
  api_addr = 0i64;
  ptr_ExportDir = (dll_base_addr + ptr_NtHeaders->OptionalHeader.DataDirectory[0].VirtualAddress);
  ptr_FuncNamesTbl = (dll_base_addr + ptr_ExportDir->AddressOfNames);
  ptr_FuncAddressTbl = (dll_base_addr + ptr_ExportDir->AddressOfFunctions);
  ptr_NameOrdinalsTbl = (dll_base_addr + ptr_ExportDir->AddressOfNameOrdinals);
  for ( i = 0; i < ptr_ExportDir->NumberOfNames; ++i )
  {
    if ( (et_calc_hash_str(0x756F0i64, (dll_base_addr + ptr_FuncNamesTbl[i]), 0x42DA2i64, 0xAA41Fi64) ^ 0x1F99519F) == pre_api_hash )
    {
      api_index = ptr_NameOrdinalsTbl[i];
      api_addr = dll_base_addr + ptr_FuncAddressTbl[api_index];
      if ( api_addr >= ptr_ExportDir && api_addr < ptr_ExportDir + ptr_NtHeaders->OptionalHeader.DataDirectory[0].Size )
        api_addr = sub_180012244((dll_base_addr + ptr_FuncAddressTbl[api_index]));
      return api_addr;
    }
  }
  return api_addr;
}
```

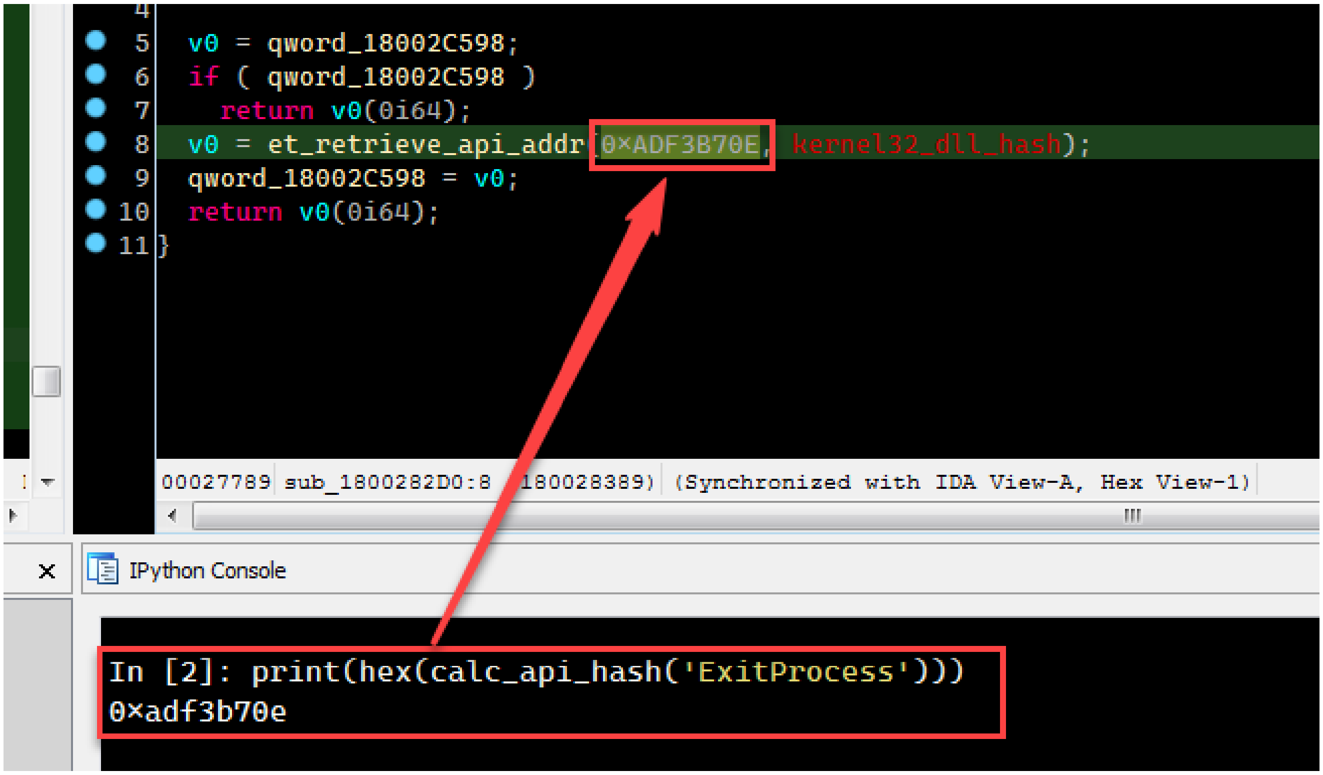
```
int __fastcall et_calc_hash_str(__int64 a1, char *str_api_name, __int64 a3, __int64 a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  et_process_args(a1, str_api_name, a3, a4);
  calced_hash = 0;
  chr = *str_api_name;
  while ( chr )
  {
    tmp = chr + 0x10040 * calced_hash;
    chr = *++str_api_name;
    calced_hash = tmp - calced_hash;
  }
  return calced_hash;
}
```

Based on the above pseudocode, it can be seen that this hash function is similar to the hash function for Dll name above, we can rewrite it in Python in another way as follows:



```
In [1]: def calc_api_hash(api_name):
   ...:     """"""
   ...:     hash_value = 0x0
   ...:     api_name_list = []
   ...:     api_name_list = list(api_name)
   ...:     for i in range(len(api_name_list)):
   ...:        ch = ord(api_name_list[i])
   ...:        hash_value = (hash_value * 0x10040 & 0xFFFFFFFF) + ch - hash_value
   ...:     # xored value need to change for each payload
   ...:     return ((hash_value ^ 0x1F99519F) & 0xFFFFFFFF)
```

Double-check with the API name is `ExitProcess`:



```
  v0 = qword_18002C598;
  if ( qword_18002C598 )
    return v0(0i64);
  v0 = et_retrieve_api_addr(0xADF3B70E, kernel32_dll_hash);
  qword_18002C598 = v0;
  return v0(0i64);
}
```

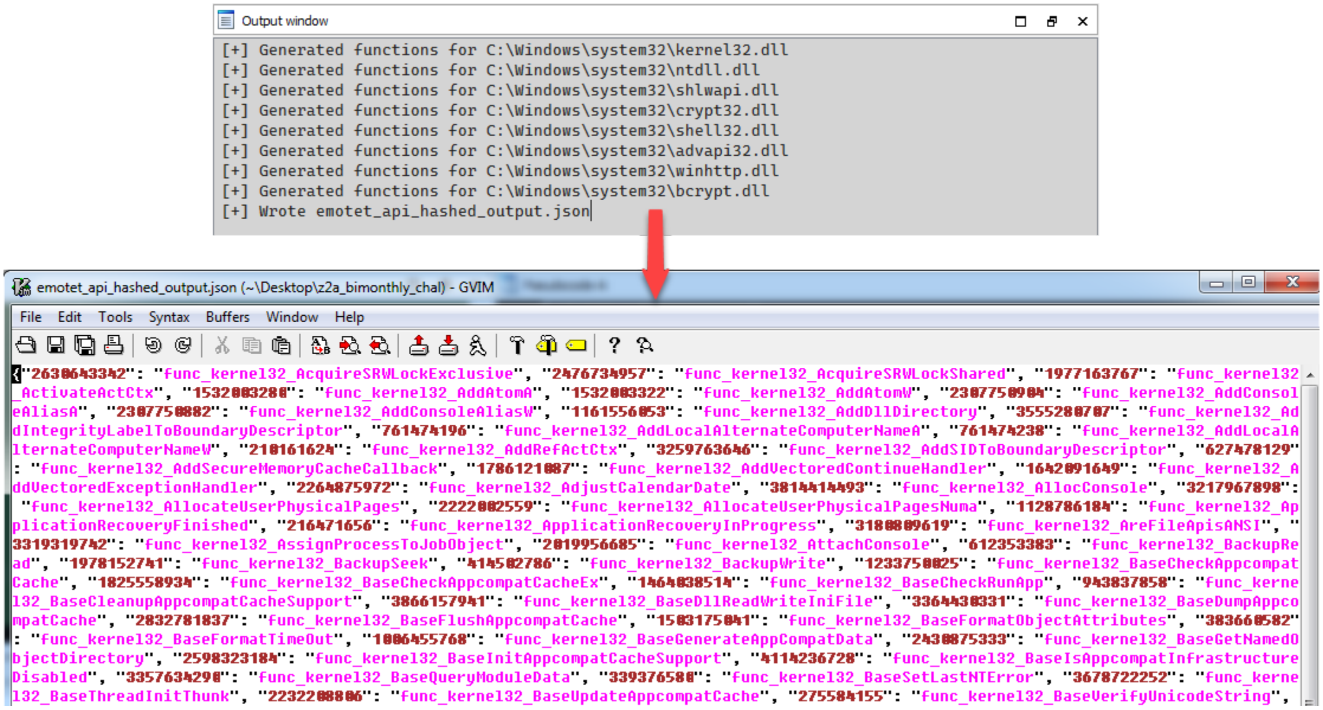
```
00027789 sub_1800282D0:8 180028389) (Synchronized with IDA View-A, Hex View-1)
```

```
In [2]: print(hex(calc_api_hash('ExitProcess')))
0xadf3b70e
```

Following this article, we can write python script to perform the following tasks:

1. Get the list of exported API functions from the list of Dlls obtained above.
2. Calculate the hash, and write the results to a JSON-formatted file as follows: `"api_hash_value": "api_name"`

Results after script runs:



Output window

```
[+] Generated functions for C:\Windows\system32\kernel32.dll
[+] Generated functions for C:\Windows\system32\ntdll.dll
[+] Generated functions for C:\Windows\system32\shlwapi.dll
[+] Generated functions for C:\Windows\system32\crypt32.dll
[+] Generated functions for C:\Windows\system32\shell32.dll
[+] Generated functions for C:\Windows\system32\advapi32.dll
[+] Generated functions for C:\Windows\system32\winhttp.dll
[+] Generated functions for C:\Windows\system32\bcrypt.dll
[+] Wrote emotet_api_hashed_output.json
```

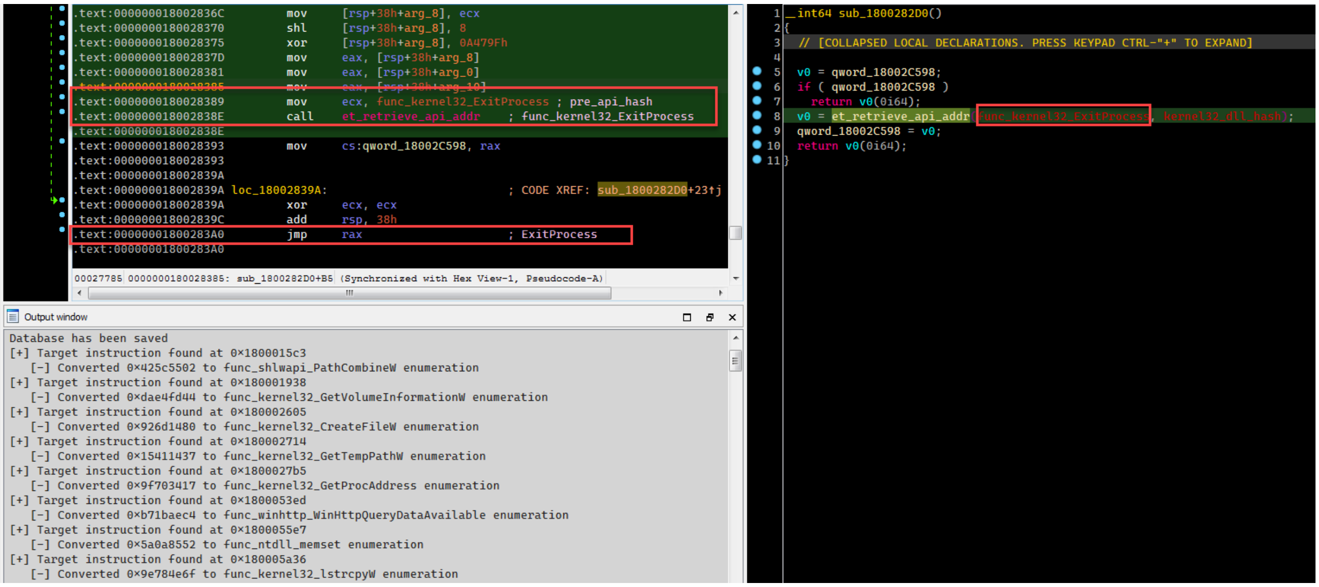
emotet_api_hashed_output.json (~\Desktop\z2a_bimonthly_chal) - GVIM

```
{"2630643342": "func_kernel32_AcquireSRWLockExclusive", "2476734957": "func_kernel32_AcquireSRWLockShared", "1977163767": "func_kernel32
_ActivateActCtx", "1532003280": "func_kernel32_AddAtomA", "1532003322": "func_kernel32_AddAtomW", "2307750904": "func_kernel32_AddConsol
eAliasA", "2307750882": "func_kernel32_AddConsoleAliasW", "1161556053": "func_kernel32_AddDllDirectory", "3555280707": "func_kernel32_Ad
dIntegrityLabelToBoundaryDescriptor", "761474196": "func_kernel32_AddLocalAlternateComputerNameA", "761474238": "func_kernel32_AddLocalA
lternateComputerNameW", "210161624": "func_kernel32_AddRefActCtx", "3259763646": "func_kernel32_AddSIDToBoundaryDescriptor", "627478129"
: "func_kernel32_AddSecureMemoryCacheCallback", "1786121087": "func_kernel32_AddVectoredContinueHandler", "1642091649": "func_kernel32_A
ddVectoredExceptionHandler", "2264875972": "func_kernel32_AdjustCalendarDate", "3814414493": "func_kernel32_AllocConsole", "3217967898":
 "func_kernel32_AllocateUserPhysicalPages", "2222002559": "func_kernel32_AllocateUserPhysicalPagesNuma", "1128786184": "func_kernel32_Ap
plicationRecoveryFinished", "216471656": "func_kernel32_ApplicationRecoveryInProgress", "3180809619": "func_kernel32_AreFileApisANSI", "
3319319742": "func_kernel32_AssignProcessToJobObject", "2019956685": "func_kernel32_AttachConsole", "612353383": "func_kernel32_BackupRe
ad", "1978152741": "func_kernel32_BackupSeek", "414502786": "func_kernel32_BackupWrite", "1233750025": "func_kernel32_BaseCheckAppcompat
Cache", "1825558934": "func_kernel32_BaseCheckAppcompatCacheEx", "1464038514": "func_kernel32_BaseCheckRunApp", "943837858": "func_kerne
l32_BaseCleanupAppcompatCacheSupport", "3866157941": "func_kernel32_BaseDllReadWriteIniFile", "3364430331": "func_kernel32_BaseDumpAppco
mpatCache", "2832781837": "func_kernel32_BaseFlushAppcompatCache", "1503175041": "func_kernel32_BaseFormatObjectAttributes", "383660582"
: "func_kernel32_BaseFormatTimeOut", "1006455768": "func_kernel32_BaseGenerateAppCompatData", "2430875333": "func_kernel32_BaseGetNamedO
bjectDirectory", "2598323184": "func_kernel32_BaseInitAppcompatCacheSupport", "4114236728": "func_kernel32_BaseIsAppcompatInfrastructure
Disabled", "3357634290": "func_kernel32_BaseQueryModuleData", "3393765080": "func_kernel32_BaseSetLastNTError", "3678722252": "func_kerne
l32_BaseThreadInitThunk", "2232200806": "func_kernel32_BaseUpdateAppcompatCache", "275584155": "func_kernel32_BaseVerifyUnicodeString",
```

Once JSON file has been generated, we can write another IDAPython script (similar to above script or refer to this code) does the following tasks:

1. Read the JSON data from the previously created into a dict variable.
2. Iterate all addresses refer to `et_retrieve_api_addr` function.
3. Find the address of the instruction that assigns the hash value of the Dll name and retrieve this hash value.
4. Check the hash value if present in the above dict variable, create a new enumeration that will store the hash-to-function-name mapping, then convert our hash back to its enumeration name.

Here are the results after script runs:



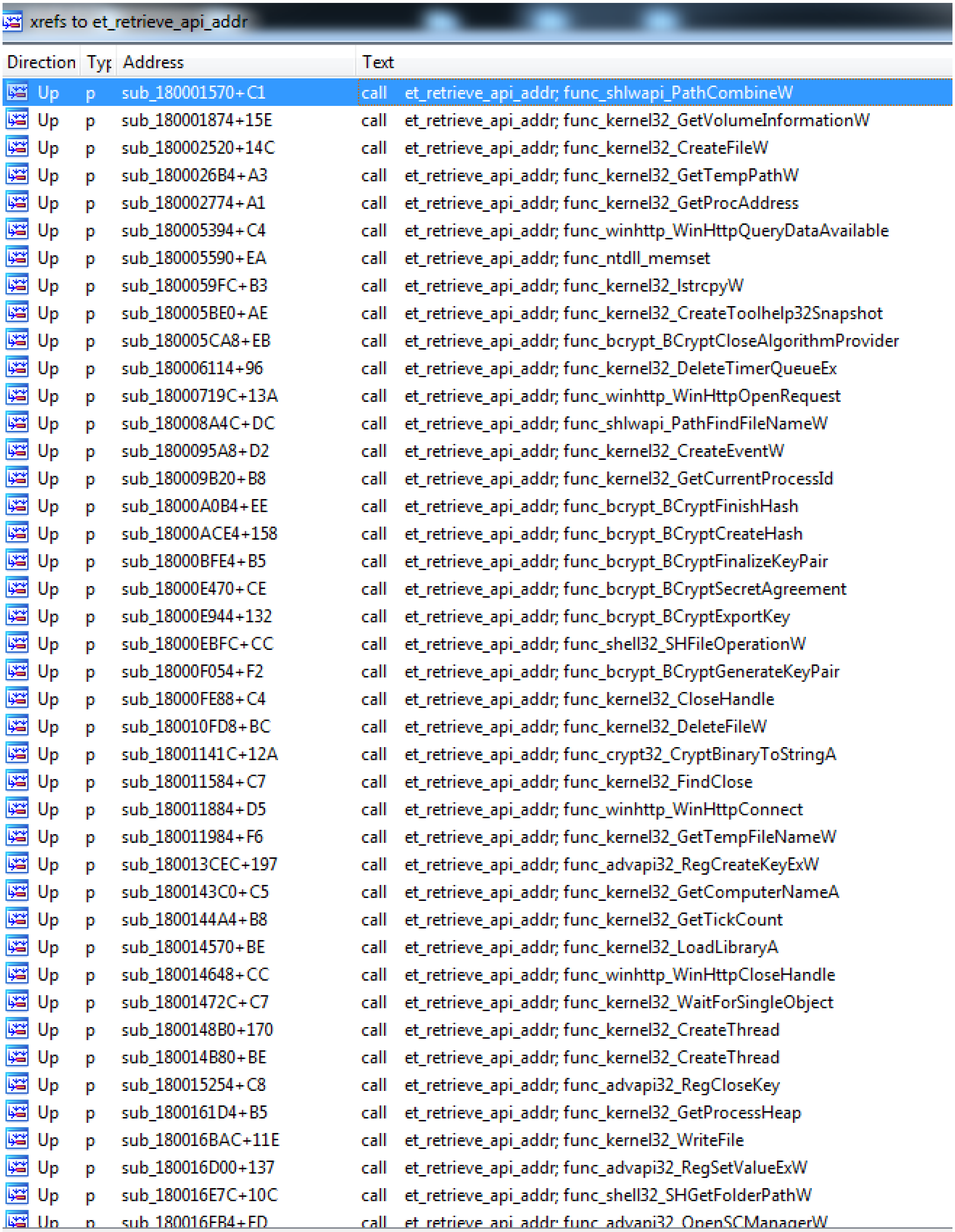
```
.text:000000018002836C          mov     [rsp+38h+arg_8], ecx
.text:0000000180028370          shl     [rsp+38h+arg_8], 8
.text:0000000180028375          xor     [rsp+38h+arg_8], 0A479Fh
.text:000000018002837D          mov     eax, [rsp+38h+arg_8]
.text:0000000180028381          mov     eax, [rsp+38h+arg_0]
.text:0000000180028385          mov     eax, [rsp+38h+arg_10]
.text:0000000180028389          mov     ecx, func_kernel32_ExitProcess ; pre_api_hash
.text:000000018002838E          call    et_retrieve_api_addr    ; func_kernel32_ExitProcess
.text:000000018002838E
.text:0000000180028393          mov     cs:qword_18002C598, rax
.text:0000000180028393
.text:000000018002839A
.text:000000018002839A loc_18002839A:                          ; CODE XREF: sub_1800282D0+23↑j
.text:000000018002839A          xor     ecx, ecx
.text:000000018002839C          add     rsp, 38h
.text:00000001800283A0          jmp     rax                     ; ExitProcess
.text:00000001800283A0
```

00027785 0000000180028385: sub_1800282D0+B5 (Synchronized with Hex View-1, Pseudocode-A)

```
__int64 sub_1800282D0()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v0 = qword_18002C598;
  if ( qword_18002C598 )
    return v0(0i64);
  v0 = et_retrieve_api_addr(func_kernel32_ExitProcess, kernel32_dll_hash);
  qword_18002C598 = v0;
  return v0(0i64);
}
```

Output window

```
Database has been saved
[+] Target instruction found at 0x1800015c3
   [-] Converted 0x425c5502 to func_shlwapi_PathCombineW enumeration
[+] Target instruction found at 0x180001938
   [-] Converted 0xdae4fd44 to func_kernel32_GetVolumeInformationW enumeration
[+] Target instruction found at 0x180002605
   [-] Converted 0x926d1480 to func_kernel32_CreateFileW enumeration
[+] Target instruction found at 0x180002714
   [-] Converted 0x15411437 to func_kernel32_GetTempPathW enumeration
[+] Target instruction found at 0x1800027b5
   [-] Converted 0x9f703417 to func_kernel32_GetProcAddress enumeration
[+] Target instruction found at 0x1800053ed
   [-] Converted 0xb71baec4 to func_winhttp_WinHttpQueryDataAvailable enumeration
[+] Target instruction found at 0x1800055e7
   [-] Converted 0x5a0a8552 to func_ntdll_memset enumeration
[+] Target instruction found at 0x180005a36
   [-] Converted 0x9e784e6f to func_kernel32_lstrcpyW enumeration
```

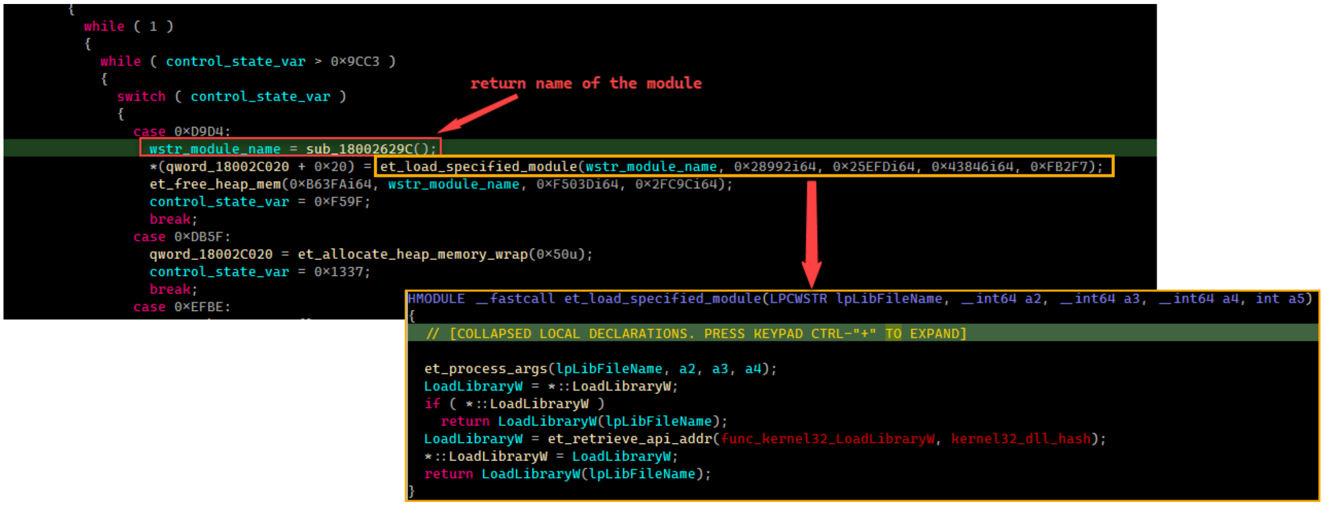


xrefs to et_retrieve_api_addr

Direction	Typ	Address	Text
Up	p	sub_180001570+C1	call et_retrieve_api_addr; func_shlwapi_PathCombineW
Up	p	sub_180001874+15E	call et_retrieve_api_addr; func_kernel32_GetVolumeInformationW
Up	p	sub_180002520+14C	call et_retrieve_api_addr; func_kernel32_CreateFileW
Up	p	sub_1800026B4+A3	call et_retrieve_api_addr; func_kernel32_GetTempPathW
Up	p	sub_180002774+A1	call et_retrieve_api_addr; func_kernel32_GetProcAddress
Up	p	sub_180005394+C4	call et_retrieve_api_addr; func_winhttp_WinHttpQueryDataAvailable
Up	p	sub_180005590+EA	call et_retrieve_api_addr; func_ntdll_memset
Up	p	sub_1800059FC+B3	call et_retrieve_api_addr; func_kernel32_lstrcpyW
Up	p	sub_180005BE0+AE	call et_retrieve_api_addr; func_kernel32_CreateToolhelp32Snapshot
Up	p	sub_180005CA8+EB	call et_retrieve_api_addr; func_bcrypt_BCryptCloseAlgorithmProvider
Up	p	sub_180006114+96	call et_retrieve_api_addr; func_kernel32_DeleteTimerQueueEx
Up	p	sub_18000719C+13A	call et_retrieve_api_addr; func_winhttp_WinHttpOpenRequest
Up	p	sub_180008A4C+DC	call et_retrieve_api_addr; func_shlwapi_PathFindFileNameW
Up	p	sub_1800095A8+D2	call et_retrieve_api_addr; func_kernel32_CreateEventW
Up	p	sub_180009B20+B8	call et_retrieve_api_addr; func_kernel32_GetCurrentProcessId
Up	p	sub_18000A0B4+EE	call et_retrieve_api_addr; func_bcrypt_BCryptFinishHash
Up	p	sub_18000ACE4+158	call et_retrieve_api_addr; func_bcrypt_BCryptCreateHash
Up	p	sub_18000BFE4+B5	call et_retrieve_api_addr; func_bcrypt_BCryptFinalizeKeyPair
Up	p	sub_18000E470+CE	call et_retrieve_api_addr; func_bcrypt_BCryptSecretAgreement
Up	p	sub_18000E944+132	call et_retrieve_api_addr; func_bcrypt_BCryptExportKey
Up	p	sub_18000EBFC+CC	call et_retrieve_api_addr; func_shell32_SHFileOperationW
Up	p	sub_18000F054+F2	call et_retrieve_api_addr; func_bcrypt_BCryptGenerateKeyPair
Up	p	sub_18000FE88+C4	call et_retrieve_api_addr; func_kernel32_CloseHandle
Up	p	sub_180010FD8+BC	call et_retrieve_api_addr; func_kernel32_DeleteFileW
Up	p	sub_18001141C+12A	call et_retrieve_api_addr; func_crypt32_CryptBinaryToStringA
Up	p	sub_180011584+C7	call et_retrieve_api_addr; func_kernel32_FindClose
Up	p	sub_180011884+D5	call et_retrieve_api_addr; func_winhttp_WinHttpConnect
Up	p	sub_180011984+F6	call et_retrieve_api_addr; func_kernel32_GetTempFileNameW
Up	p	sub_180013CEC+197	call et_retrieve_api_addr; func_advapi32_RegCreateKeyExW
Up	p	sub_1800143C0+C5	call et_retrieve_api_addr; func_kernel32_GetComputerNameA
Up	p	sub_1800144A4+B8	call et_retrieve_api_addr; func_kernel32_GetTickCount
Up	p	sub_180014570+BE	call et_retrieve_api_addr; func_kernel32_LoadLibraryA
Up	p	sub_180014648+CC	call et_retrieve_api_addr; func_winhttp_WinHttpCloseHandle
Up	p	sub_18001472C+C7	call et_retrieve_api_addr; func_kernel32_WaitForSingleObject
Up	p	sub_1800148B0+170	call et_retrieve_api_addr; func_kernel32_CreateThread
Up	p	sub_180014B80+BE	call et_retrieve_api_addr; func_kernel32_CreateThread
Up	p	sub_180015254+C8	call et_retrieve_api_addr; func_advapi32_RegCloseKey
Up	p	sub_1800161D4+B5	call et_retrieve_api_addr; func_kernel32_GetProcessHeap
Up	p	sub_180016BAC+11E	call et_retrieve_api_addr; func_kernel32_WriteFile
Up	p	sub_180016D00+137	call et_retrieve_api_addr; func_advapi32_RegSetValueExW
Up	p	sub_180016E7C+10C	call et_retrieve_api_addr; func_shell32_SHGetFolderPathW
Up	p	sub_180016EB4+FD	call et_retrieve_api_addr; func_advapi32_OpenSCManagerW

3. Decrypt strings

To find the function that decrypt the strings, the fastest way is to find the function that calls the `LoadLibraryW` API because this function will take as an argument the name of the module to be loaded.

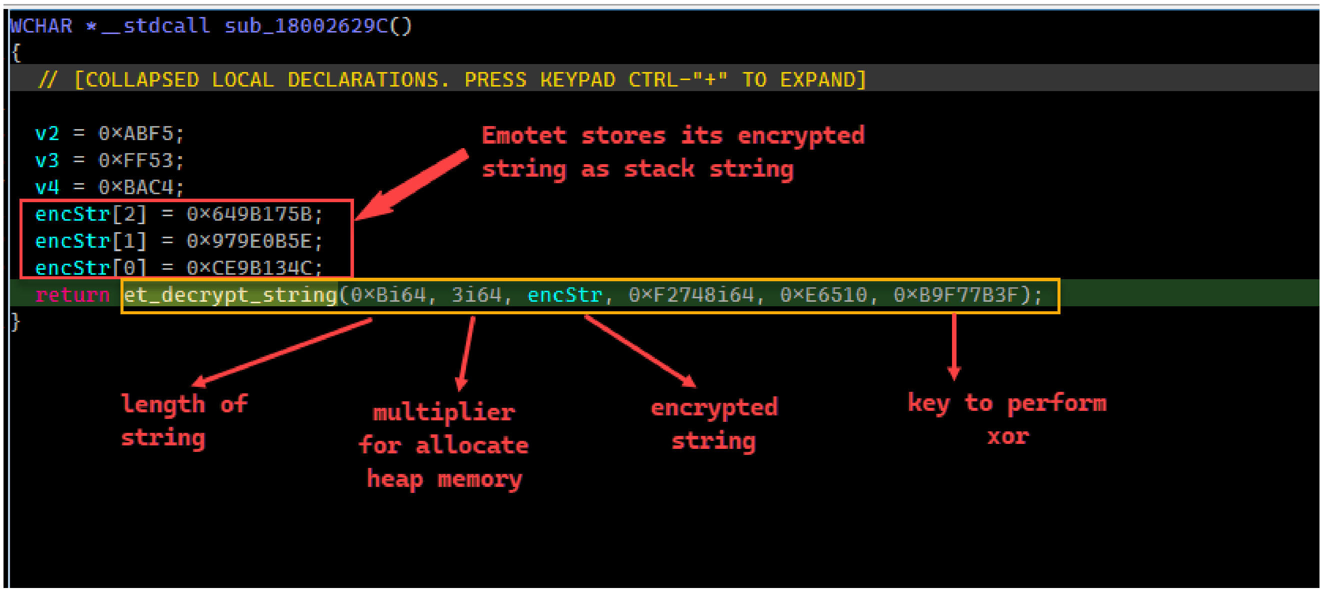


return name of the module

```
{
  while ( 1 )
  {
    while ( control_state_var > 0×9CC3 )
    {
      switch ( control_state_var )
      {
        case 0×D9D4:
          wstr_module_name = sub_18002629C();
          *(qword_18002C020 + 0×20) = et_load_specified_module(wstr_module_name, 0×28992i64, 0×25EFDi64, 0×43846i64, 0×FB2F7);
          et_free_heap_mem(0×B63FAi64, wstr_module_name, 0×F503Di64, 0×2FC9Ci64);
          control_state_var = 0×F59F;
          break;
        case 0×DB5F:
          qword_18002C020 = et_allocate_heap_memory_wrap(0×50u);
          control_state_var = 0×1337;
          break;
        case 0×EFBE:
```

```
HMODULE __fastcall et_load_specified_module(LPCWSTR lpLibFileName, __int64 a2, __int64 a3, __int64 a4, int a5)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  et_process_args(lpLibFileName, a2, a3, a4);
  LoadLibraryW = *::LoadLibraryW;
  if ( *::LoadLibraryW )
    return LoadLibraryW(lpLibFileName);
  LoadLibraryW = et_retrieve_api_addr(func_kernel32_LoadLibraryW, kernel32_dll_hash);
  *::LoadLibraryW = LoadLibraryW;
  return LoadLibraryW(lpLibFileName);
}
```

As the figure above, **sub_18002629C** will return the name of the module. The pseudocode at **sub_18002629C** stores its encrypted string as stack string, then calls the **et_decrypt_string (0x180025C58)** function to decrypt:



Emotet stores its encrypted string as stack string

```
WCHAR *__stdcall sub_18002629C()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = 0×ABF5;
  v3 = 0×FF53;
  v4 = 0×BAC4;
  encStr[2] = 0×649B175B;
  encStr[1] = 0×979E0B5E;
  encStr[0] = 0×CE9B134C;
  return et_decrypt_string(0×Bi64, 3i64, encStr, 0×F2748i64, 0×E6510, 0×B9F77B3F);
}
```

length of string

multiplier for allocate heap memory

encrypted string

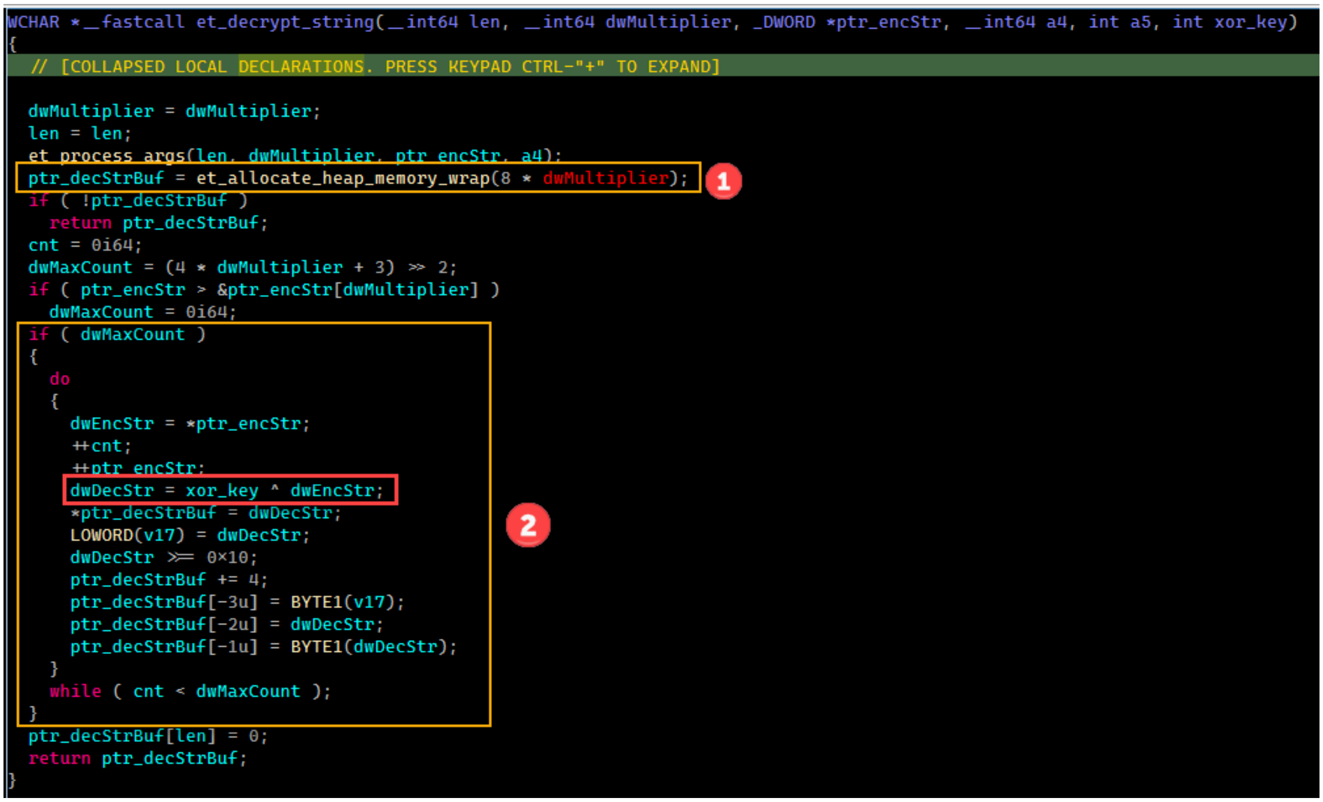
key to perform xor

The **et_decrypt_string** function accepts parameters for the decryption process, including:

1. Length of decrypted string.
2. Multiplier (used for allocating heap memory to store the decoded string).
3. Encrypted string stored as a stack string. These values are all dynamically calculated by Emotet and then stored on the stack.
4. Key used for decryption.

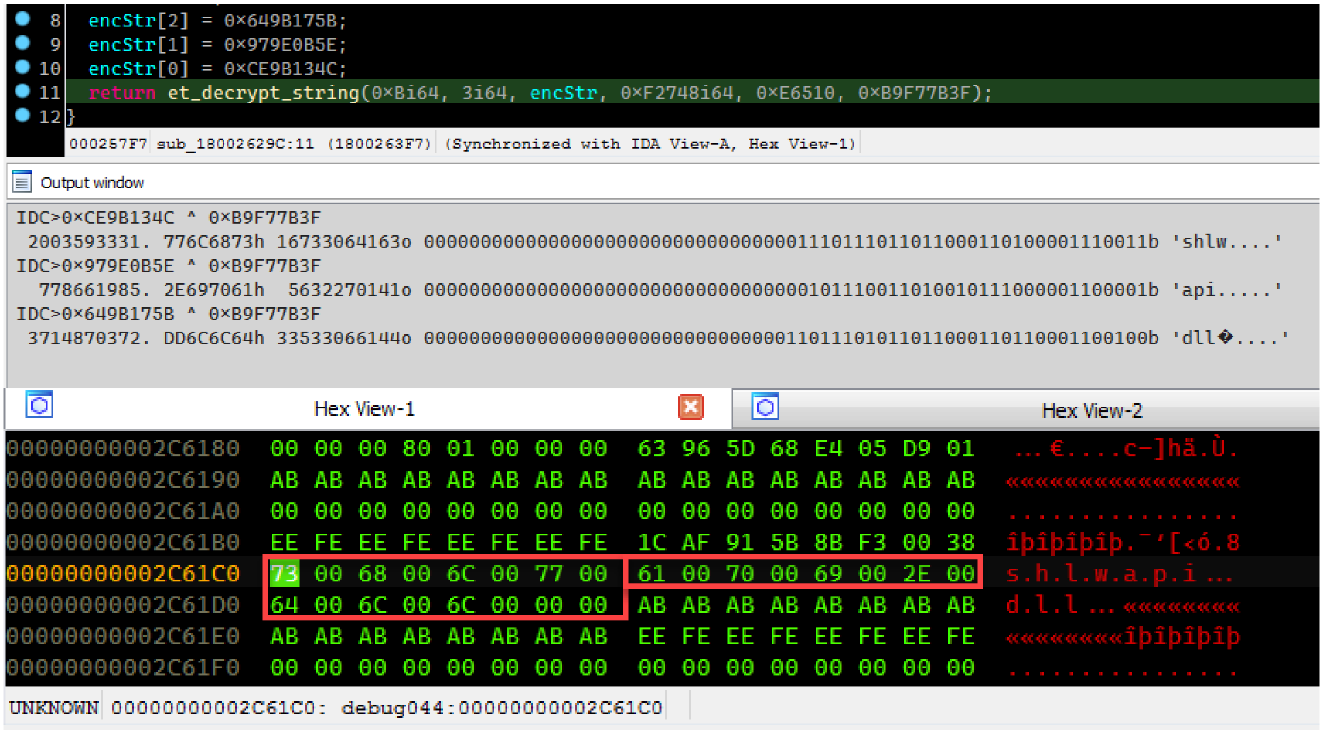
The pseudocode of the function as shown below:

1. Allocate heap memory to store the decrypted string.
2. Execute the loop, load each dword of the encrypted string, perform the xor operation with the decryption key, and then assign the value after decryption to the allocated memory.



```
WCHAR *__fastcall et_decrypt_string(__int64 len, __int64 dwMultiplier, _DWORD *ptr_encStr, __int64 a4, int a5, int xor_key)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  dwMultiplier = dwMultiplier;
  len = len;
  et_process_args(len, dwMultiplier, ptr_encStr, a4);
  ptr_decStrBuf = et_allocate_heap_memory_wrap(8 * dwMultiplier);
  if ( !ptr_decStrBuf )
    return ptr_decStrBuf;
  cnt = 0i64;
  dwMaxCount = (4 * dwMultiplier + 3) >> 2;
  if ( ptr_encStr > &ptr_encStr[dwMultiplier] )
    dwMaxCount = 0i64;
  if ( dwMaxCount )
  {
    do
    {
      dwEncStr = *ptr_encStr;
      ++cnt;
      ++ptr_encStr;
      dwDecStr = xor_key ^ dwEncStr;
      *ptr_decStrBuf = dwDecStr;
      LOWORD(v17) = dwDecStr;
      dwDecStr >>= 0x10;
      ptr_decStrBuf += 4;
      ptr_decStrBuf[-3u] = BYTE1(v17);
      ptr_decStrBuf[-2u] = dwDecStr;
      ptr_decStrBuf[-1u] = BYTE1(dwDecStr);
    }
    while ( cnt < dwMaxCount );
  }
  ptr_decStrBuf[len] = 0;
  return ptr_decStrBuf;
}
```

To verify we can do xor each value as below or through debugging:



```
  encStr[2] = 0x649B175B;
  encStr[1] = 0x979E0B5E;
  encStr[0] = 0xCE9B134C;
  return et_decrypt_string(0xBi64, 3i64, encStr, 0xF2748i64, 0xE6510, 0xB9F77B3F);
}
000257F7 sub_18002629C:11 (1800263F7) (Synchronized with IDA View-A, Hex View-1)
```

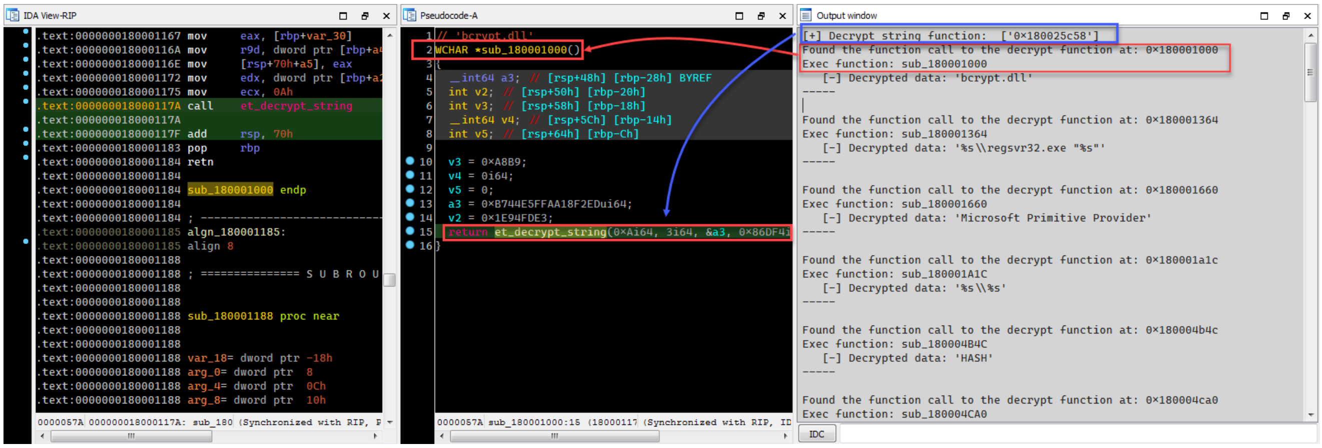
```
IDC>0xCE9B134C ^ 0xB9F77B3F
 2003593331. 776C6873h 16733064163o 0000000000000000000000000000000001110111011011000110100001110011b 'shlw....'
IDC>0x979E0B5E ^ 0xB9F77B3F
  778661985. 2E697061h  5632270141o 0000000000000000000000000000000000101110011010010111000001100001b 'api.....'
IDC>0x649B175B ^ 0xB9F77B3F
 3714870372. DD6C6C64h 33533066144o 0000000000000000000000000000000011011101011011000110110001100100b 'dll�....'
```

```
00000000002C6180  00 00 00 80 01 00 00 00  63 96 5D 68 E4 05 D9 01  ...€....c–]hä.Ù.
00000000002C6190  AB AB AB AB AB AB AB AB  AB AB AB AB AB AB AB AB  ««««««««««««««««
00000000002C61A0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000000002C61B0  EE FE EE FE EE FE EE FE  1C AF 91 5B 8B F3 00 38  îþîþîþîþ.¯‘[‹ó.8
00000000002C61C0  73 00 68 00 6C 00 77 00  61 00 70 00 69 00 2E 00  s.h.l.w.a.p.i...
00000000002C61D0  64 00 6C 00 6C 00 00 00  AB AB AB AB AB AB AB AB  d.l.l...««««««««
00000000002C61E0  AB AB AB AB AB AB AB AB  EE FE EE FE EE FE EE FE  ««««««««îþîþîþîþ
00000000002C61F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
UNKNOWN 00000000002C61C0: debug044:00000000002C61C0
```

As mentioned above, the encrypted string has a variable length and the values of the encrypted string are dynamically calculated by Emotet before being stored to the stack. Therefore, it is difficult to get these values for writing script to perform decryption. Therefore, one of the most possible ways is to write a script that uses IDA Appcall feature to execute a call to the decryption function and receive the decrypted string as the return result.

```
import idc, idautils, idaapi

#------------------------------------------------------------------------
def clean_data(data):
    data = data.rstrip(b'\x00')
    if b'\x00\x00' in data:
        data = data.split(b'\x00\x00')[0].replace(b'\x00', b'')
    else:
        if data.count(b'\x00') == 1:
            data = data.split(b'\x00')[0]
        else:
            data = data.replace(b'\x00', b'')

    data = data.decode('latin-1')
    return data

#------------------------------------------------------------------------
def find_and_decrypt_data(func_addr):
    for call_addr in idautils.CodeRefsTo(func_addr, 1):
        func_call_addr = idaapi.get_func(call_addr).start_ea
        print ("Found the function call to the decrypt function at: 0x%x" %
func_call_addr)

        dec_func_name = idc.get_func_name(func_call_addr)
        print ("Exec function: %s" % dec_func_name)
        dec_func_proto = "wchar_t * __fastcall {:s}();".format(dec_func_name)
        dec_func = idaapi.Appcall.proto(dec_func_name, dec_func_proto)

        #Call function to decrypt data and clean the decrypted data
        try:
            dec_data = dec_func()
            dec_data = clean_data(dec_data)
            if dec_data:
                print("   [-] Decrypted data: %s" % repr(dec_data))
                print('-----\n')
        except Exception as e:
            print("FAILED: appcall failed: {}".format(e))
            continue

        #Set comment
        try:
            idc.set_cmt(call_addr, repr(dec_data), idc.SN_NOWARN)
            idc.set_func_cmt(func_call_addr, repr(dec_data), 1)
        except:
            print("FAILED: to add comment")
            continue

#------------------------------------------------------------------------
def main():
    """"""
    dec_str_funcs = [0x0180025C58]
    print('[+] Decrypt string function: ', ['0x%x' % routine for routine in
dec_str_funcs])
    for func_addr in dec_str_funcs:
        find_and_decrypt_data(func_addr)
#-----------------------------------------------------------------------
if __name__ == '__main__':
    main()
```

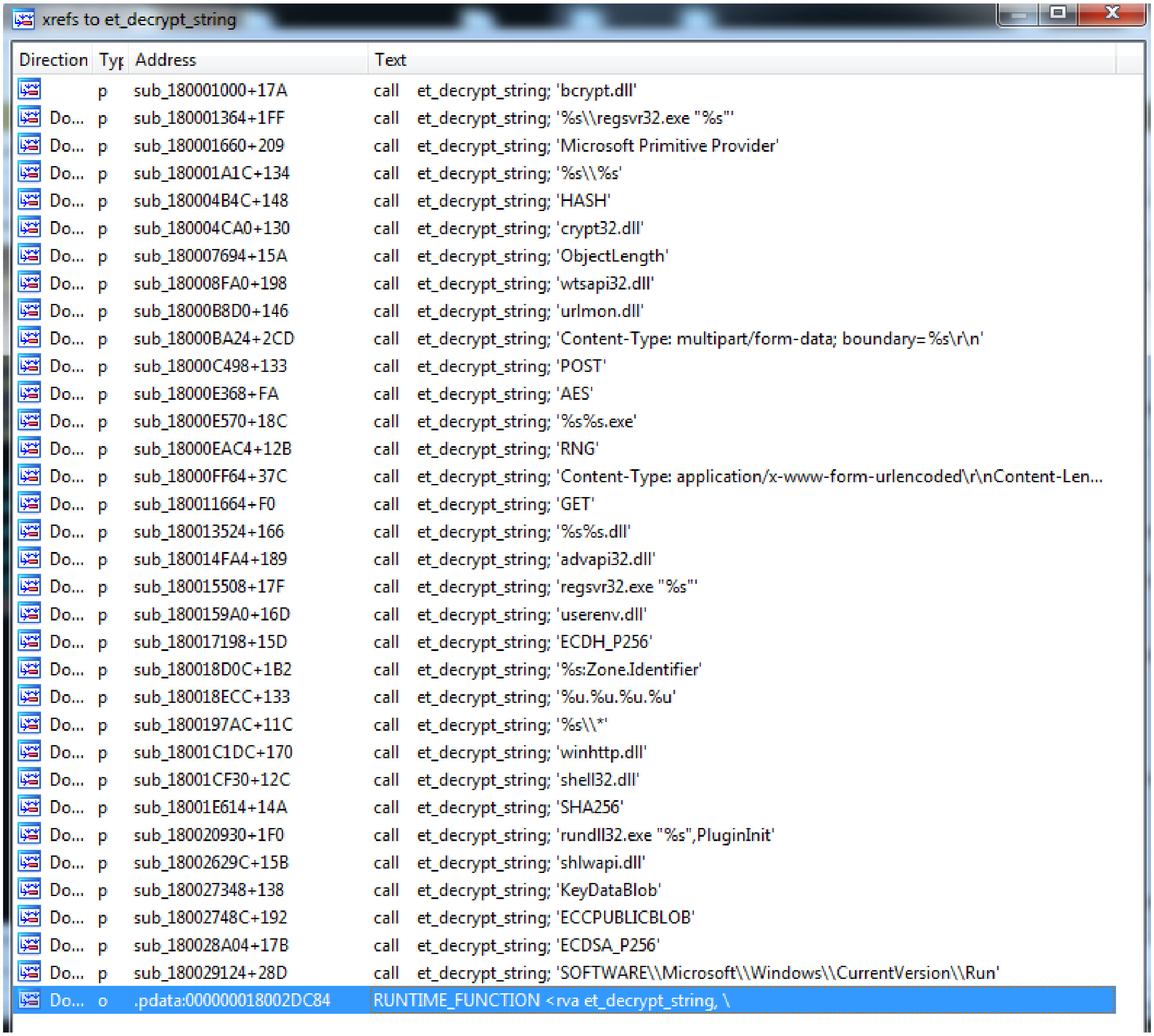
The final result after script runs:



```
.text:0000000180001167 mov     eax, [rbp+var_30]
.text:000000018000116A mov     r9d, dword ptr [rbp+a4
.text:000000018000116E mov     [rsp+70h+a5], eax
.text:0000000180001172 mov     edx, dword ptr [rbp+a2
.text:0000000180001175 mov     ecx, 0Ah
.text:000000018000117A call    et_decrypt_string
.text:000000018000117A
.text:000000018000117F add     rsp, 70h
.text:0000000180001183 pop     rbp
.text:0000000180001184 retn
.text:0000000180001184
.text:0000000180001184 sub_180001000 endp
.text:0000000180001184
.text:0000000180001184 ; ----------------------------
.text:0000000180001185 algn_180001185:
.text:0000000180001185 align 8
.text:0000000180001188
.text:0000000180001188 ; =============== S U B R O U
.text:0000000180001188
.text:0000000180001188
.text:0000000180001188 sub_180001188 proc near
.text:0000000180001188
.text:0000000180001188
.text:0000000180001188 var_18= dword ptr -18h
.text:0000000180001188 arg_0= dword ptr  8
.text:0000000180001188 arg_4= dword ptr  0Ch
.text:0000000180001188 arg_8= dword ptr  10h
0000057A 000000018000117A: sub_180 (Synchronized with RIP, P
```

```
// 'bcrypt.dll'
WCHAR *sub_180001000()
{
  __int64 a3; // [rsp+48h] [rbp-28h] BYREF
  int v2; // [rsp+50h] [rbp-20h]
  int v3; // [rsp+58h] [rbp-18h]
  __int64 v4; // [rsp+5Ch] [rbp-14h]
  int v5; // [rsp+64h] [rbp-Ch]

  v3 = 0xA8B9;
  v4 = 0i64;
  v5 = 0;
  a3 = 0xB744E5FFAA18F2EDui64;
  v2 = 0x1E94FDE3;
  return et_decrypt_string(0xAi64, 3i64, &a3, 0x86DF4i
}
0000057A sub_180001000:15 (18000117 (Synchronized with RIP, ID
```

```
[+] Decrypt string function:  ['0x180025c58']
Found the function call to the decrypt function at: 0x180001000
Exec function: sub_180001000
   [-] Decrypted data: 'bcrypt.dll'
-----

Found the function call to the decrypt function at: 0x180001364
Exec function: sub_180001364
   [-] Decrypted data: '%s\\regsvr32.exe "%s"'
-----

Found the function call to the decrypt function at: 0x180001660
Exec function: sub_180001660
   [-] Decrypted data: 'Microsoft Primitive Provider'
-----

Found the function call to the decrypt function at: 0x180001a1c
Exec function: sub_180001A1C
   [-] Decrypted data: '%s\\%s'
-----

Found the function call to the decrypt function at: 0x180004b4c
Exec function: sub_180004B4C
   [-] Decrypted data: 'HASH'
-----

Found the function call to the decrypt function at: 0x180004ca0
Exec function: sub_180004CA0
```



Direction	Typ	Address	Text
	p	sub_180001000+17A	call et_decrypt_string; 'bcrypt.dll'
Do...	p	sub_180001364+1FF	call et_decrypt_string; '%s\\regsvr32.exe "%s"'
Do...	p	sub_180001660+209	call et_decrypt_string; 'Microsoft Primitive Provider'
Do...	p	sub_180001A1C+134	call et_decrypt_string; '%s\\%s'
Do...	p	sub_180004B4C+148	call et_decrypt_string; 'HASH'
Do...	p	sub_180004CA0+130	call et_decrypt_string; 'crypt32.dll'
Do...	p	sub_180007694+15A	call et_decrypt_string; 'ObjectLength'
Do...	p	sub_180008FA0+198	call et_decrypt_string; 'wtsapi32.dll'
Do...	p	sub_18000B8D0+146	call et_decrypt_string; 'urlmon.dll'
Do...	p	sub_18000BA24+2CD	call et_decrypt_string; 'Content-Type: multipart/form-data; boundary=%s\r\n'
Do...	p	sub_18000C498+133	call et_decrypt_string; 'POST'
Do...	p	sub_18000E368+FA	call et_decrypt_string; 'AES'
Do...	p	sub_18000E570+18C	call et_decrypt_string; '%s%s.exe'
Do...	p	sub_18000EAC4+12B	call et_decrypt_string; 'RNG'
Do...	p	sub_18000FF64+37C	call et_decrypt_string; 'Content-Type: application/x-www-form-urlencoded\r\nContent-Len...
Do...	p	sub_180011664+F0	call et_decrypt_string; 'GET'
Do...	p	sub_180013524+166	call et_decrypt_string; '%s%s.dll'
Do...	p	sub_180014FA4+189	call et_decrypt_string; 'advapi32.dll'
Do...	p	sub_180015508+17F	call et_decrypt_string; 'regsvr32.exe "%s"'
Do...	p	sub_1800159A0+16D	call et_decrypt_string; 'userenv.dll'
Do...	p	sub_180017198+15D	call et_decrypt_string; 'ECDH_P256'
Do...	p	sub_180018D0C+1B2	call et_decrypt_string; '%s:Zone.Identifier'
Do...	p	sub_180018ECC+133	call et_decrypt_string; '%u.%u.%u.%u'
Do...	p	sub_1800197AC+11C	call et_decrypt_string; '%s*'
Do...	p	sub_18001C1DC+170	call et_decrypt_string; 'winhttp.dll'
Do...	p	sub_18001CF30+12C	call et_decrypt_string; 'shell32.dll'
Do...	p	sub_18001E614+14A	call et_decrypt_string; 'SHA256'
Do...	p	sub_180020930+1F0	call et_decrypt_string; 'rundll32.exe "%s",PluginInit'
Do...	p	sub_18002629C+15B	call et_decrypt_string; 'shlwapi.dll'
Do...	p	sub_180027348+138	call et_decrypt_string; 'KeyDataBlob'
Do...	p	sub_18002748C+192	call et_decrypt_string; 'ECCPUBLICBLOB'
Do...	p	sub_180028A04+17B	call et_decrypt_string; 'ECDSA_P256'
Do...	p	sub_180029124+28D	call et_decrypt_string; 'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run'
Do...	o	.pdata:000000018002DC84	RUNTIME_FUNCTION <rva et_decrypt_string, \

4. References

End.

m4n0w4r