[Case study] Decrypt strings using Dumpulator

kienmanowar.wordpress.com/2023/05/22/case-study-decrypt-strings-using-dumpulator/

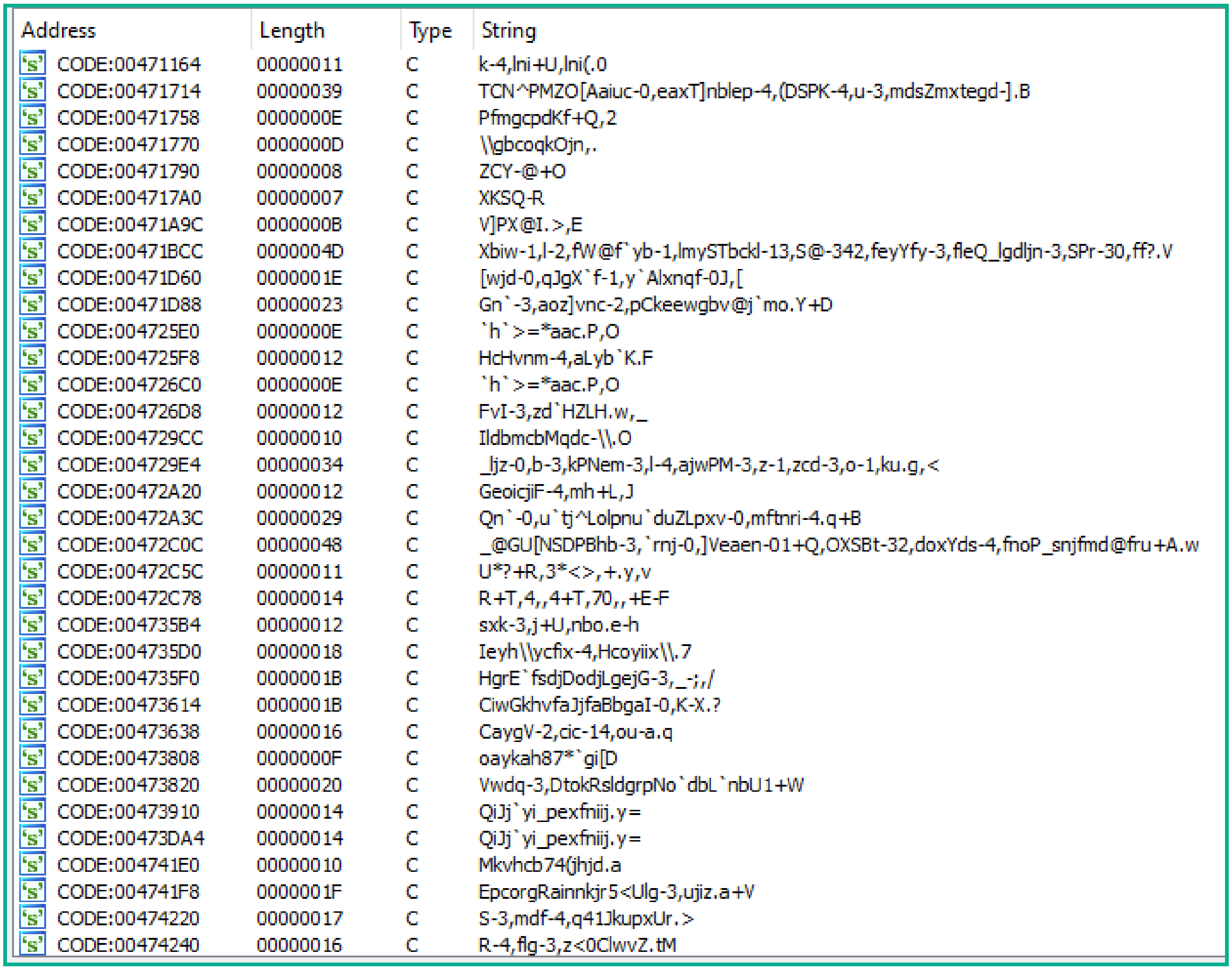
May 22, 2023

1. References

- Dumpulator (by **mrexodia** Duncan Ogilvie)
- Native function and Assembly Code Invocation
- OALABS Research
- And **@herrcore** (Thanks for his suggestion in private chat)

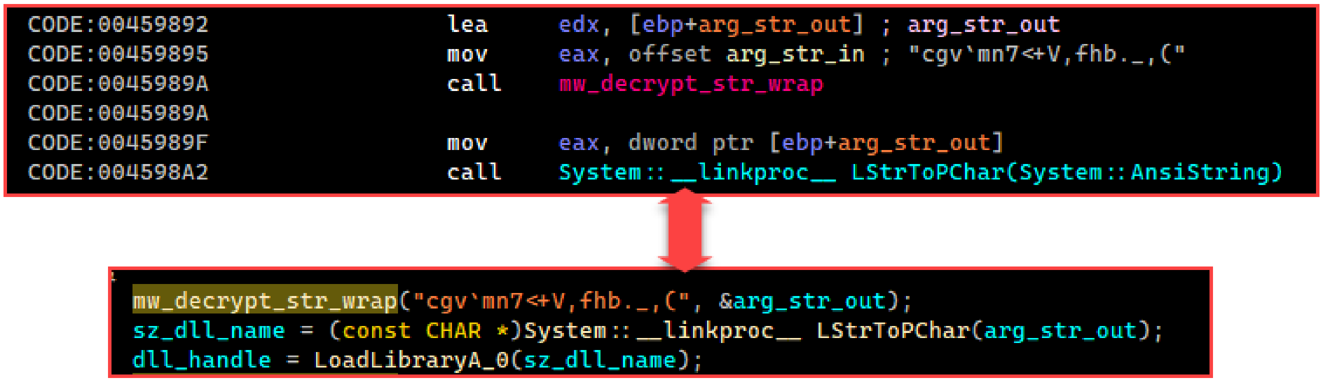
2. Code analysis

I received a suspicious Dll that needs to be analyzed. This Dll is packed. After unpacking it and throwing the Dll into IDA, IDA successfully analyzed it with over **7000** functions (*including API/library function calls*). Upon quickly examining at the **Strings tab**, I came across numerous strings in the following format:



Address	Length	Type	String
CODE:00471164	00000011	C	k-4,lni+U,lni(.0
CODE:00471714	00000039	C	TCN^PMZO[Aaiuc-0,eaxT]nblep-4,(DSPK-4,u-3,mdsZmxtegd-].B
CODE:00471758	0000000E	C	PfmgcpdKf+Q,2
CODE:00471770	0000000D	C	\\gbcoqkOjn,.
CODE:00471790	00000008	C	ZCY-@+O
CODE:004717A0	00000007	C	XKSQ-R
CODE:00471A9C	0000000B	C	V]PX@I.>,E
CODE:00471BCC	0000004D	C	Xbiw-1,l-2,fW@f`yb-1,lmySTbckl-13,S@-342,feyYfy-3,fleQ_lgdljn-3,SPr-30,ff?.V
CODE:00471D60	0000001E	C	[wjd-0,qJgX`f-1,y`Alxnqf-0J,[
CODE:00471D88	00000023	C	Gn`-3,aoz]vnc-2,pCkeewgbv@j`mo.Y+D
CODE:004725E0	0000000E	C	`h`>=*aac.P,O
CODE:004725F8	00000012	C	HcHvnm-4,aLyb`K.F
CODE:004726C0	0000000E	C	`h`>=*aac.P,O
CODE:004726D8	00000012	C	FvI-3,zd`HZLH.w,_
CODE:004729CC	00000010	C	IldbmcbMqdc-\\.O
CODE:004729E4	00000034	C	_ljz-0,b-3,kPNem-3,l-4,ajwPM-3,z-1,zcd-3,o-1,ku.g,<
CODE:00472A20	00000012	C	GeoicjiF-4,mh+L,J
CODE:00472A3C	00000029	C	Qn`-0,u`tj^Lolpnu`duZLpxv-0,mftnri-4.q+B
CODE:00472C0C	00000048	C	_@GU[NSDPBhb-3,`rnj-0,]Veaen-01+Q,OXSBt-32,doxYds-4,fnoP_snjfmd@fru+A.w
CODE:00472C5C	00000011	C	U*?+R,3*<>,+.y,v
CODE:00472C78	00000014	C	R+T,4,,4+T,70,,+E-F
CODE:004735B4	00000012	C	sxk-3,j+U,nbo.e-h
CODE:004735D0	00000018	C	Ieyh\\ycfix-4,Hcoyiix\\.7
CODE:004735F0	0000001B	C	HgrE`fsdjDodjLgejG-3,_-;,/
CODE:00473614	0000001B	C	CiwGkhvfaJjfaBbgaI-0,K-X.?
CODE:00473638	00000016	C	CaygV-2,cic-14,ou-a.q
CODE:00473808	0000000F	C	oaykah87*`gi[D
CODE:00473820	00000020	C	Vwdq-3,DtokRsldgrpNo`dbL`nbU1+W
CODE:00473910	00000014	C	QiJj`yi_pexfniij.y=
CODE:00473DA4	00000014	C	QiJj`yi_pexfniij.y=
CODE:004741E0	00000010	C	Mkvhcb74(jhjd.a
CODE:004741F8	0000001F	C	EpcorgRainnkjr5<Ulg-3,ujiz.a+V
CODE:00474220	00000017	C	S-3,mdf-4,q41JkupxUr.>
CODE:00474240	00000016	C	R-4,flg-3,z<0ClwvZ.tM

Based on the information provided, I believe these strings have definitely been encrypted. Going through the code snippet using an arbitrary string, I found the corresponding assembly code and pseudocode as follows (*function and variable names have been changed accordingly*):



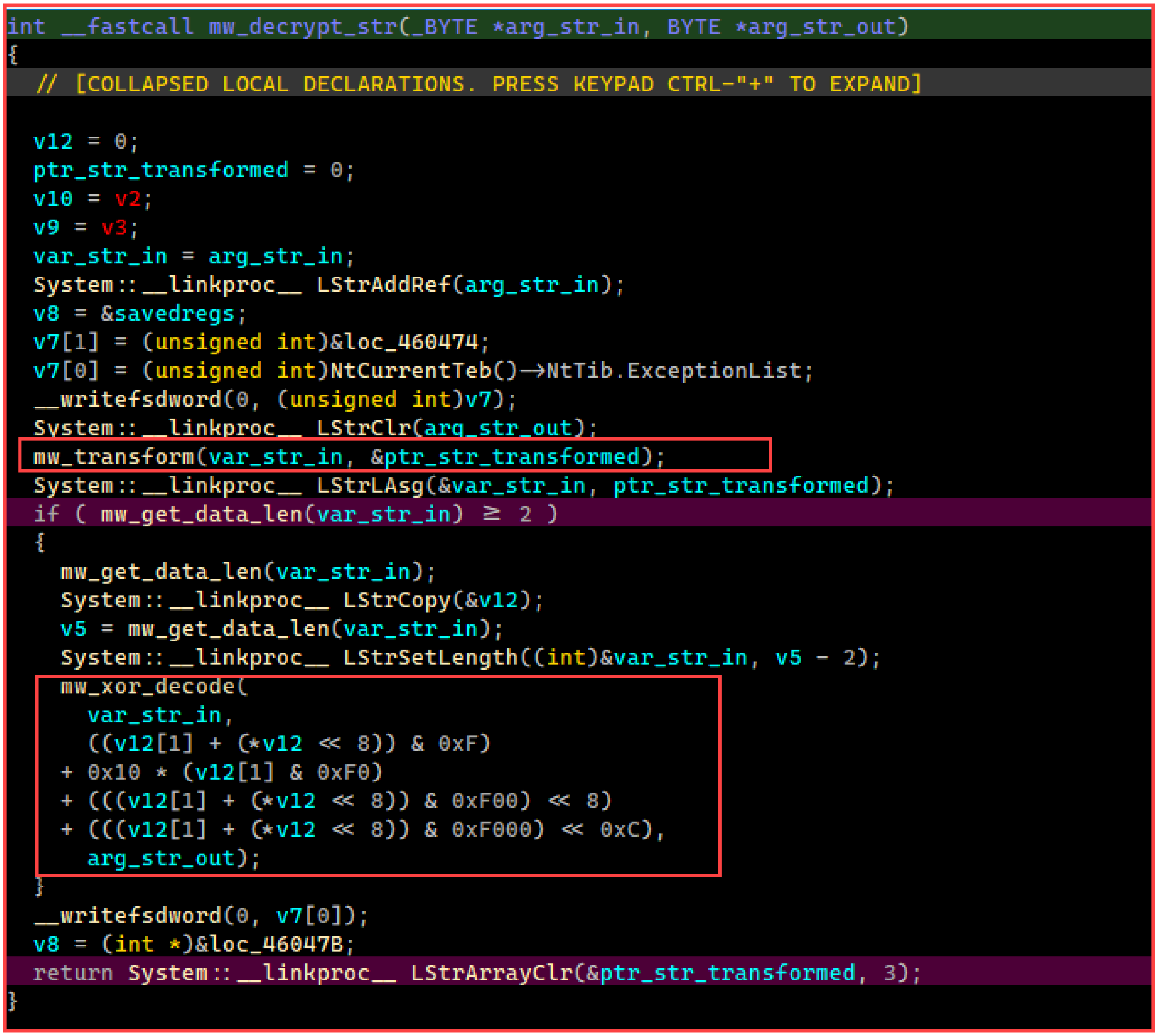
```
CODE:00459892                 lea     edx, [ebp+arg_str_out] ; arg_str_out
CODE:00459895                 mov     eax, offset arg_str_in ; "cgv`mn7<+V,fhb._,("
CODE:0045989A                 call    mw_decrypt_str_wrap
CODE:0045989A
CODE:0045989F                 mov     eax, dword ptr [ebp+arg_str_out]
CODE:004598A2                 call    System::__linkproc__ LStrToPChar(System::AnsiString)
```

```
mw_decrypt_str_wrap("cgv`mn7<+V,fhb._,(", &arg_str_out);
sz_dll_name = (const CHAR *)System::__linkproc__ LStrToPChar(arg_str_out);
dll_handle = LoadLibraryA_0(sz_dll_name);
```

With the image above, it is easy to see:

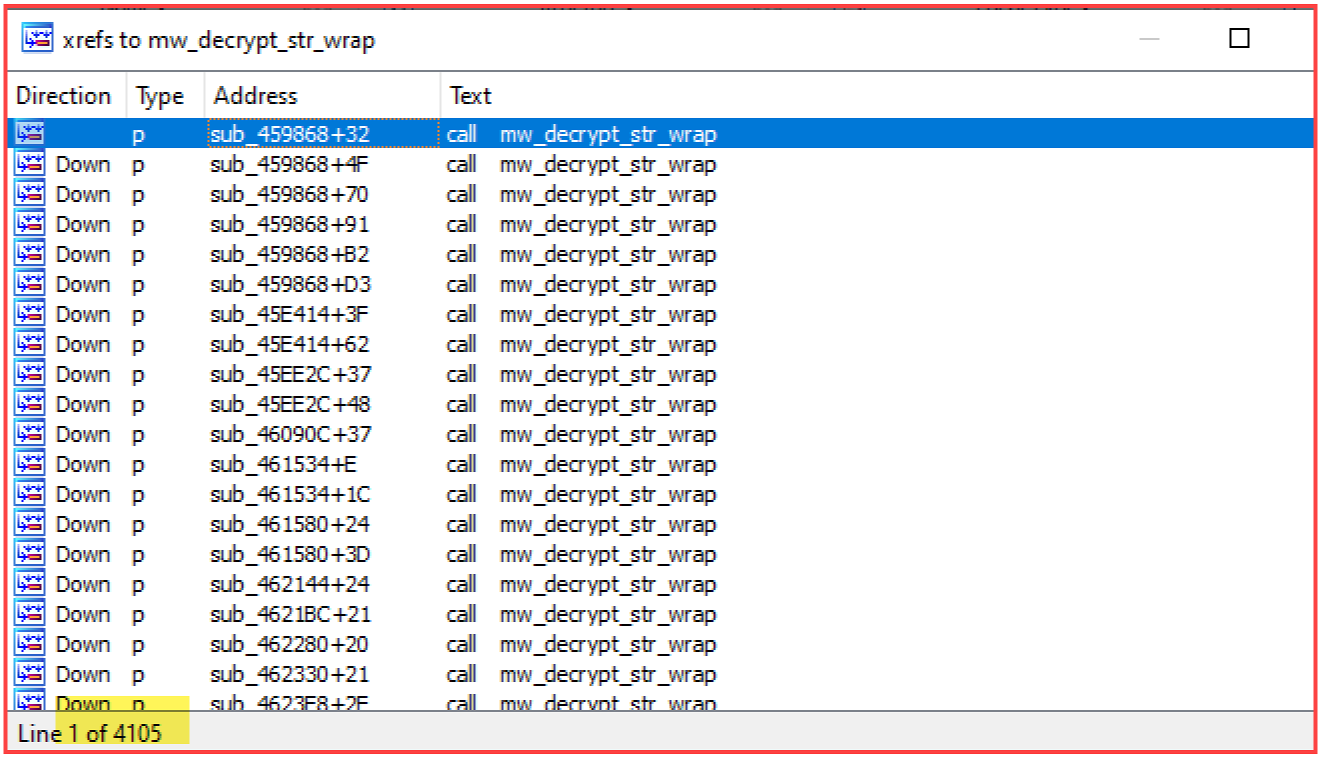
- The `EAX` register will hold the address of the encrypted string.
- The`EDX` register will hold the address of the string after decryption.
- The `mw_decrypt_str_wrap` function performs the task of decrypting the string.

Here, if any of you have the same idea of analyzing the `mw_decrypt_str_wrap` function to rewrite the IDApython code for decryption, congratulations to you 🙂 You share the same thought as me! The `mw_decrypt_str_wrap` function will call the `mw_decrypt_str` function.



```
int __fastcall mw_decrypt_str(_BYTE *arg_str_in, BYTE *arg_str_out)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v12 = 0;
  ptr_str_transformed = 0;
  v10 = v2;
  v9 = v3;
  var_str_in = arg_str_in;
  System::__linkproc__ LStrAddRef(arg_str_in);
  v8 = &savedregs;
  v7[1] = (unsigned int)&loc_460474;
  v7[0] = (unsigned int)NtCurrentTeb()->NtTib.ExceptionList;
  __writefsdword(0, (unsigned int)v7);
  System::__linkproc__ LStrClr(arg_str_out);
  mw_transform(var_str_in, &ptr_str_transformed);
  System::__linkproc__ LStrLAsg(&var_str_in, ptr_str_transformed);
  if ( mw_get_data_len(var_str_in) >= 2 )
  {
    mw_get_data_len(var_str_in);
    System::__linkproc__ LStrCopy(&v12);
    v5 = mw_get_data_len(var_str_in);
    System::__linkproc__ LStrSetLength((int)&var_str_in, v5 - 2);
    mw_xor_decode(
      var_str_in,
      ((v12[1] + (*v12 << 8)) & 0xF)
    + 0x10 * (v12[1] & 0xF0)
    + (((v12[1] + (*v12 << 8)) & 0xF00) << 8)
    + (((v12[1] + (*v12 << 8)) & 0xF000) << 0xC),
      arg_str_out);
  }
  __writefsdword(0, v7[0]);
  v8 = (int *)&loc_46047B;
  return System::__linkproc__ LStrArrayClr(&ptr_str_transformed, 3);
}
```

After going around various functions and thinking about how to code, I started feeling increasingly discouraged. Moreover, when examining the cross-references to the `mw_decrypt_str_wrap` function, I noticed that it was called over **4000 times** to decrypt strings… WTF 😐



Direction	Type	Address	Text
	p	sub_459868+32	call mw_decrypt_str_wrap
Down	p	sub_459868+4F	call mw_decrypt_str_wrap
Down	p	sub_459868+70	call mw_decrypt_str_wrap
Down	p	sub_459868+91	call mw_decrypt_str_wrap
Down	p	sub_459868+B2	call mw_decrypt_str_wrap
Down	p	sub_459868+D3	call mw_decrypt_str_wrap
Down	p	sub_45E414+3F	call mw_decrypt_str_wrap
Down	p	sub_45E414+62	call mw_decrypt_str_wrap
Down	p	sub_45EE2C+37	call mw_decrypt_str_wrap
Down	p	sub_45EE2C+48	call mw_decrypt_str_wrap
Down	p	sub_46090C+37	call mw_decrypt_str_wrap
Down	p	sub_461534+E	call mw_decrypt_str_wrap
Down	p	sub_461534+1C	call mw_decrypt_str_wrap
Down	p	sub_461580+24	call mw_decrypt_str_wrap
Down	p	sub_461580+3D	call mw_decrypt_str_wrap
Down	p	sub_462144+24	call mw_decrypt_str_wrap
Down	p	sub_4621BC+21	call mw_decrypt_str_wrap
Down	p	sub_462280+20	call mw_decrypt_str_wrap
Down	p	sub_462330+21	call mw_decrypt_str_wrap
Down	p	sub_4623E8+2E	call mw_decrypt_str_wrap

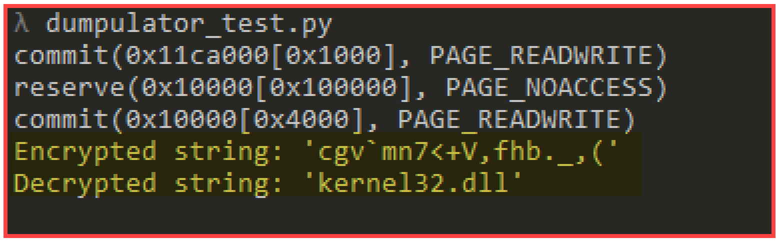
Line 1 of 4105

3. Use dumpulator

As shown in the above image, there are too many function calls to the decryption function. Moreover, rewriting this decryption function would be time-consuming and require code debugging for verification. I think I need to find a way to emulate this function to perform the decryption step and retrieve the decrypted string. Several solutions came to mind, and I also asked my brother, who suggested using x or y solutions. After some trial and error, I decided to try using **dumpulator**. To be able to use dumpulator, we first need to create a minidump file of this DLL (*dump when halted at DllEntryPoint*). After obtaining the dump file, I tested the following code snippet:

```
from dumpulator import Dumpulator

dec_str_fn = 0x02FE08C0
enc_str_offset = 0x02FD9988

dp = Dumpulator("mal_dll.dmp", quiet=True)
tmp_addr = dp.allocate(256)
dp.call(dec_str_fn, [], regs={'eax':enc_str_offset , 'edx': tmp_addr})
dec_str = dp.read_str(dp.read_long(tmp_addr))
print(f"Encrypted string: '{dp.read_str(enc_str_offset)}'")
print(f"Decrypted string: '{dec_str}'")
```

Result when executing the above code:



```
λ dumpulator_test.py
commit(0x11ca000[0x1000], PAGE_READWRITE)
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
Encrypted string: 'cgv`mn7<+V,fhb._,('
Decrypted string: 'kernel32.dll'
```

H0ly Sh1T… 😂 that’s exactly what I wanted.

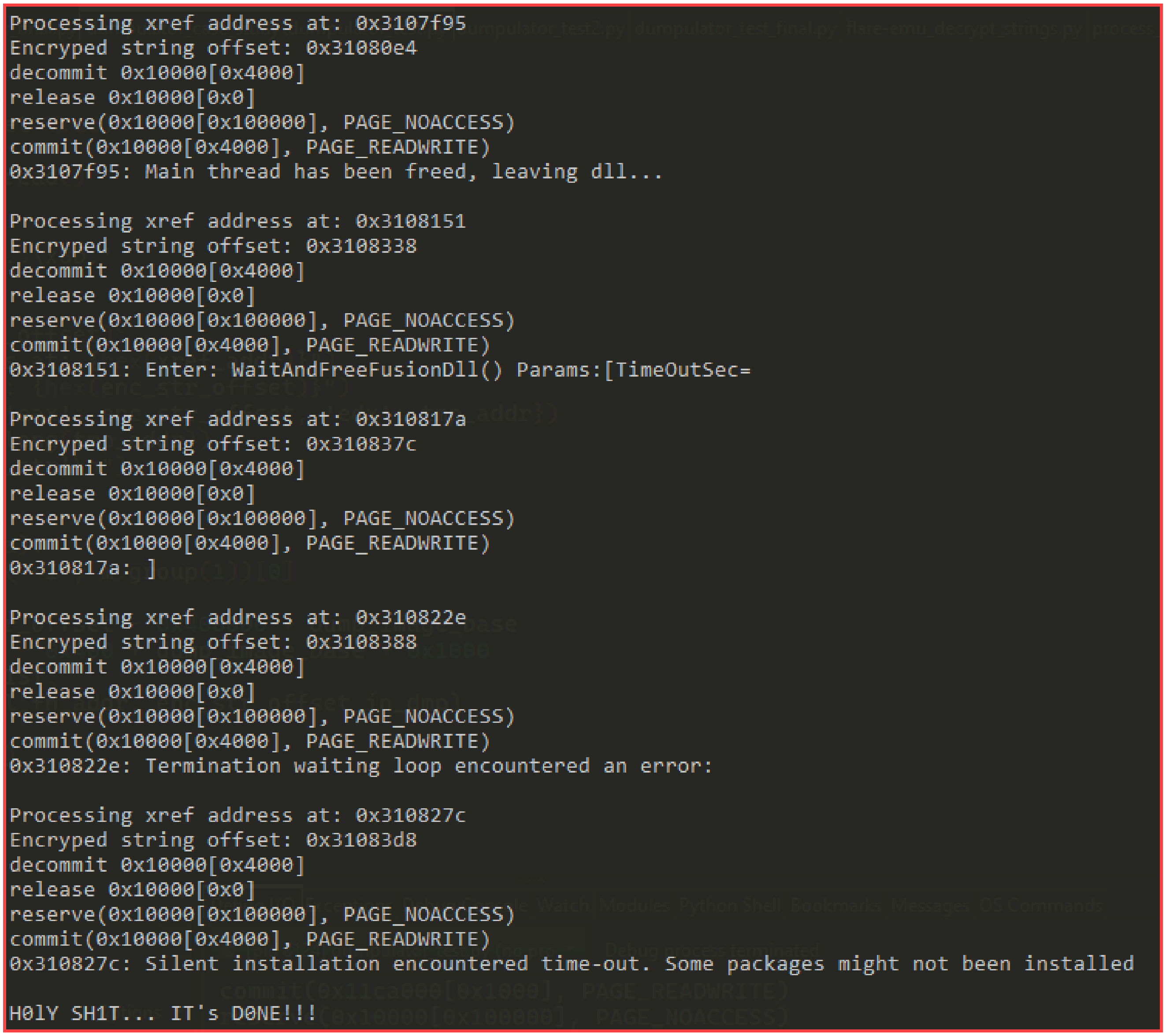
Next, I will rewrite the code according to my intention as follows:

- Use regex to search for patterns and extract all encoded string addresses.
- Filter out addresses that match the pattern but are not decryption functions or undefined addresses and add them to the `BLACK_LIST`.

Here’s a lame code snippet that meets my needs:

```
import re
import struct
import pefile
from dumpulator import Dumpulator

dump_image_base = 0x2F80000
dec_str_fn = 0x02FE08C0

BLACK_LIST = [0x3027520, 0x30380b6, 0x30380d0, 0x3039a08, 0x3039169, 0x303a6b6,
0x303aa0e, 0x303ab5c, 0x303bbf3, 0x3066075, 0x306661b, 0x3083e50,
              0x3084373, 0x30856d1, 0x30858aa, 0x308c7ac, 0x308d02d, 0x30acbfd,
0x30cd12e, 0x30cd187, 0x30cd670, 0x30cd6d4, 0x30cfe2f, 0x30d4cc4,
              0x3106da0]

FILE_PATH = 'dumped_dll.dll'
dp = Dumpulator("mal_dll.dmp", quiet=True)

file_data = open(FILE_PATH, 'rb').read()
pe = pefile.PE(data=file_data)

egg = rb'\x8D\x55.\xB8(....)\xE8....\x8b.'
tmp_addr = dp.allocate(256)

def decrypt_str(xref_addr, enc_str_offset):
    print(f"Processing xref address at: {hex(xref_addr)}")
    print(f"Encryped string offset: {hex(enc_str_offset)}")
    dp.call(dec_str_fn, [], regs={'eax': enc_str_offset, 'edx': tmp_addr})
    dec_str = dp.read_str(dp.read_long(tmp_addr))
    print(f"{hex(xref_addr)}: {dec_str}\n")
    return dec_str

for m in re.finditer(egg, file_data):
    enc_str_offset = struct.unpack('<I', m.group(1))[0]
    inst_offset = m.start()
    enc_str_offset_in_dmp = enc_str_offset - 0x400000 + dump_image_base
    call_fn_addr = inst_offset + 8 - 0x400 + dump_image_base + 0x1000
    if call_fn_addr not in BLACK_LIST:
        str_ret =  decrypt_str(call_fn_addr, enc_str_offset_in_dmp)

print(f"H0lY SH1T... IT's D0NE!!!")
```

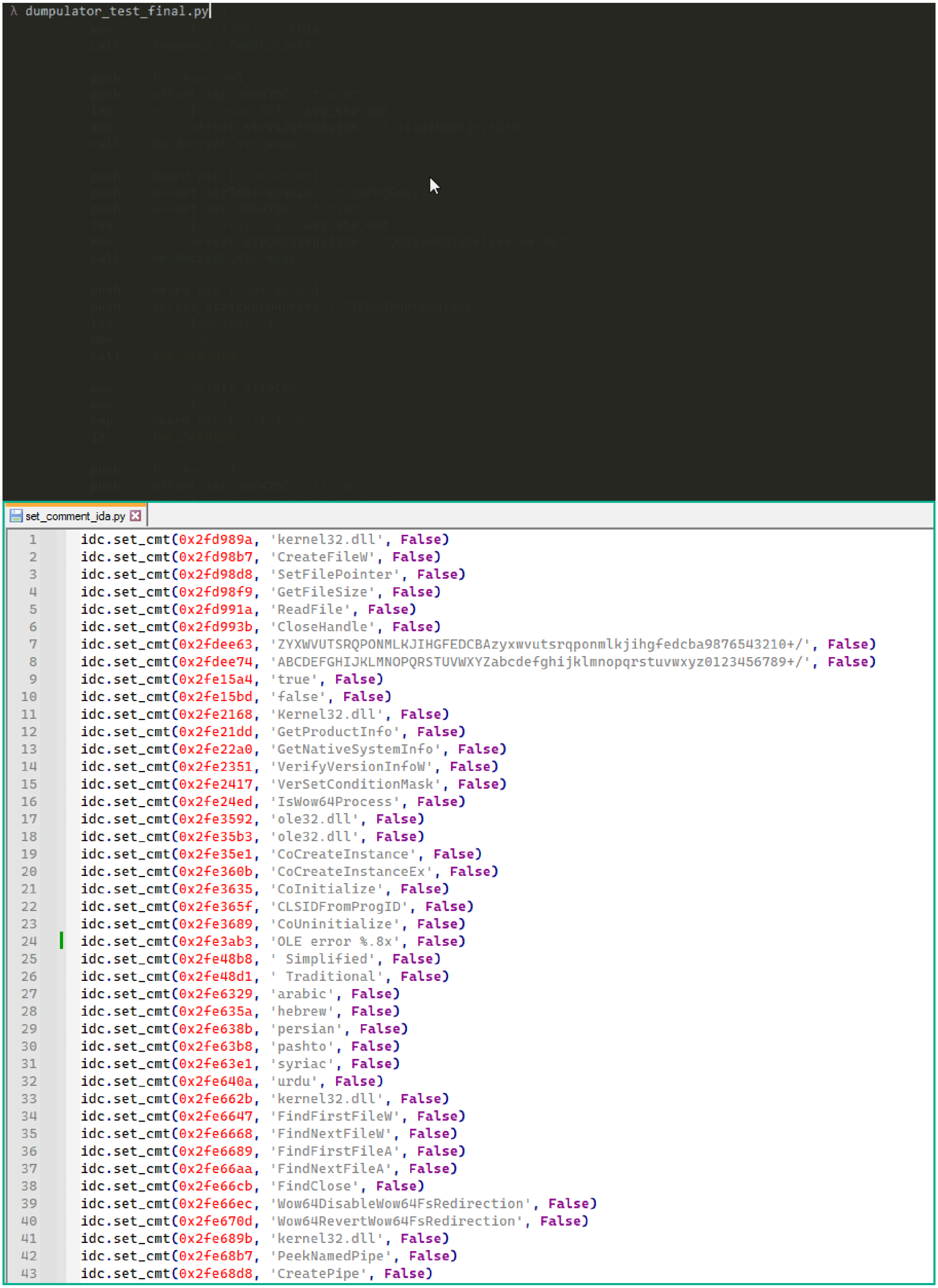
Result when executing the above script:



```
Processing xref address at: 0x3107f95
Encryped string offset: 0x31080e4
decommit 0x10000[0x4000]
release 0x10000[0x0]
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
0x3107f95: Main thread has been freed, leaving dll...

Processing xref address at: 0x3108151
Encryped string offset: 0x3108338
decommit 0x10000[0x4000]
release 0x10000[0x0]
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
0x3108151: Enter: WaitAndFreeFusionDll() Params:[TimeOutSec=

Processing xref address at: 0x310817a
Encryped string offset: 0x310837c
decommit 0x10000[0x4000]
release 0x10000[0x0]
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
0x310817a: ]

Processing xref address at: 0x310822e
Encryped string offset: 0x3108388
decommit 0x10000[0x4000]
release 0x10000[0x0]
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
0x310822e: Termination waiting loop encountered an error:

Processing xref address at: 0x310827c
Encryped string offset: 0x31083d8
decommit 0x10000[0x4000]
release 0x10000[0x0]
reserve(0x10000[0x100000], PAGE_NOACCESS)
commit(0x10000[0x4000], PAGE_READWRITE)
0x310827c: Silent installation encountered time-out. Some packages might not been installed

H0lY SH1T... IT's D0NE!!!
```

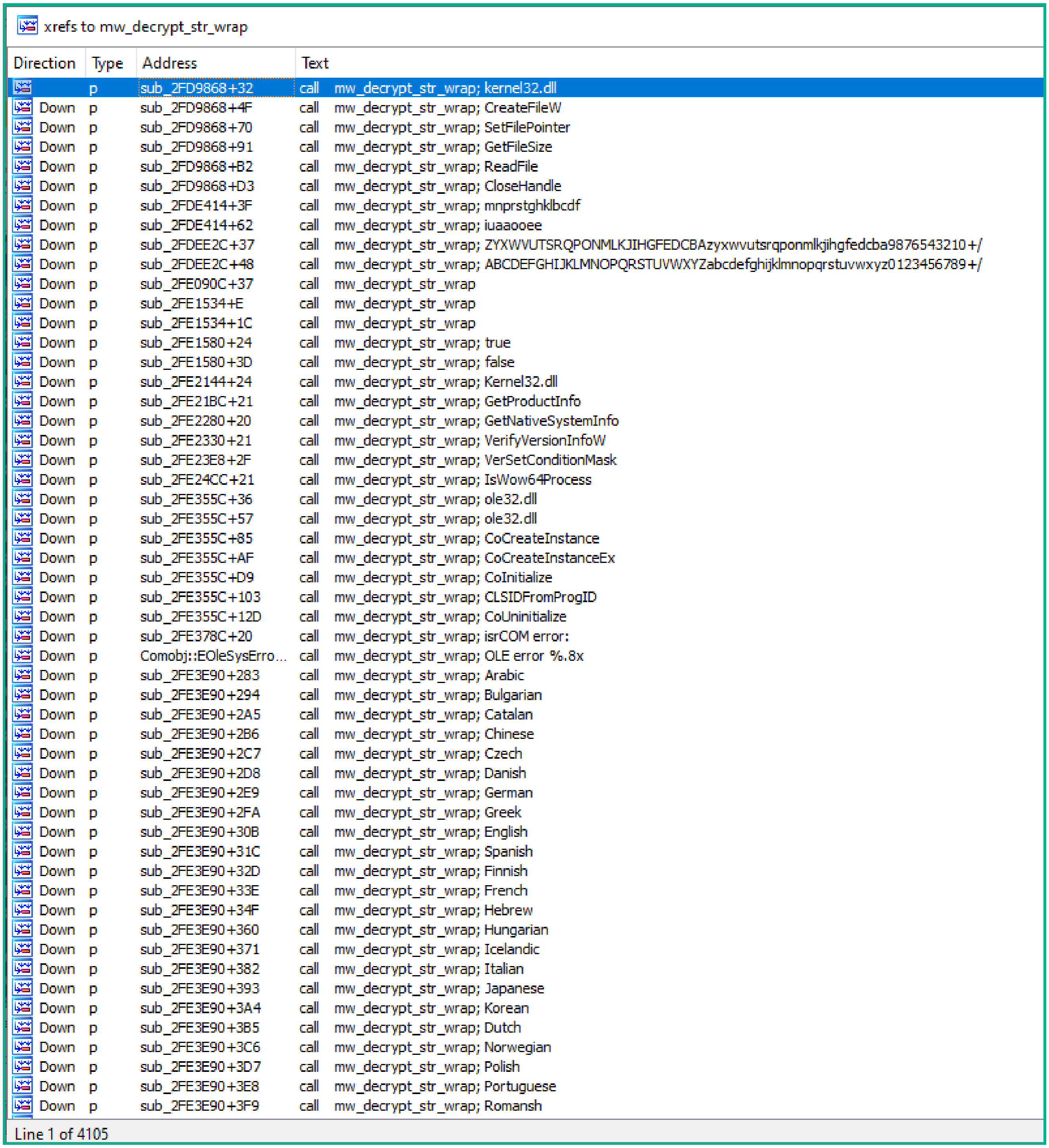
No errors whatsoever 😈!!! As a final step, I added a code snippet to this script that will output a Python file. This file will contain the `idc.set_cmt` commands to set comment for the decrypted strings above at the address where the decrypt function is called.

The final result is as follows:



```
λ dumpulator_test_final.py
```

```
idc.set_cmt(0x2fd989a, 'kernel32.dll', False)
idc.set_cmt(0x2fd98b7, 'CreateFileW', False)
idc.set_cmt(0x2fd98d8, 'SetFilePointer', False)
idc.set_cmt(0x2fd98f9, 'GetFileSize', False)
idc.set_cmt(0x2fd991a, 'ReadFile', False)
idc.set_cmt(0x2fd993b, 'CloseHandle', False)
idc.set_cmt(0x2fdee63, 'ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/', False)
idc.set_cmt(0x2fdee74, 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/', False)
idc.set_cmt(0x2fe15a4, 'true', False)
idc.set_cmt(0x2fe15bd, 'false', False)
idc.set_cmt(0x2fe2168, 'Kernel32.dll', False)
idc.set_cmt(0x2fe21dd, 'GetProductInfo', False)
idc.set_cmt(0x2fe22a0, 'GetNativeSystemInfo', False)
idc.set_cmt(0x2fe2351, 'VerifyVersionInfoW', False)
idc.set_cmt(0x2fe2417, 'VerSetConditionMask', False)
idc.set_cmt(0x2fe24ed, 'IsWow64Process', False)
idc.set_cmt(0x2fe3592, 'ole32.dll', False)
idc.set_cmt(0x2fe35b3, 'ole32.dll', False)
idc.set_cmt(0x2fe35e1, 'CoCreateInstance', False)
idc.set_cmt(0x2fe360b, 'CoCreateInstanceEx', False)
idc.set_cmt(0x2fe3635, 'CoInitialize', False)
idc.set_cmt(0x2fe365f, 'CLSIDFromProgID', False)
idc.set_cmt(0x2fe3689, 'CoUninitialize', False)
idc.set_cmt(0x2fe3ab3, 'OLE error %.8x', False)
idc.set_cmt(0x2fe48b8, ' Simplified', False)
idc.set_cmt(0x2fe48d1, ' Traditional', False)
idc.set_cmt(0x2fe6329, 'arabic', False)
idc.set_cmt(0x2fe635a, 'hebrew', False)
idc.set_cmt(0x2fe638b, 'persian', False)
idc.set_cmt(0x2fe63b8, 'pashto', False)
idc.set_cmt(0x2fe63e1, 'syriac', False)
idc.set_cmt(0x2fe640a, 'urdu', False)
idc.set_cmt(0x2fe662b, 'kernel32.dll', False)
idc.set_cmt(0x2fe6647, 'FindFirstFileW', False)
idc.set_cmt(0x2fe6668, 'FindNextFileW', False)
idc.set_cmt(0x2fe6689, 'FindFirstFileA', False)
idc.set_cmt(0x2fe66aa, 'FindNextFileA', False)
idc.set_cmt(0x2fe66cb, 'FindClose', False)
idc.set_cmt(0x2fe66ec, 'Wow64DisableWow64FsRedirection', False)
idc.set_cmt(0x2fe670d, 'Wow64RevertWow64FsRedirection', False)
idc.set_cmt(0x2fe689b, 'kernel32.dll', False)
idc.set_cmt(0x2fe68b7, 'PeekNamedPipe', False)
idc.set_cmt(0x2fe68d8, 'CreatePipe', False)
```



xrefs to mw_decrypt_str_wrap

Direction	Type	Address	Text
	p	sub_2FD9868+32	call mw_decrypt_str_wrap; kernel32.dll
Down	p	sub_2FD9868+4F	call mw_decrypt_str_wrap; CreateFileW
Down	p	sub_2FD9868+70	call mw_decrypt_str_wrap; SetFilePointer
Down	p	sub_2FD9868+91	call mw_decrypt_str_wrap; GetFileSize
Down	p	sub_2FD9868+B2	call mw_decrypt_str_wrap; ReadFile
Down	p	sub_2FD9868+D3	call mw_decrypt_str_wrap; CloseHandle
Down	p	sub_2FDE414+3F	call mw_decrypt_str_wrap; mnprstghklbcdf
Down	p	sub_2FDE414+62	call mw_decrypt_str_wrap; iuaaooee
Down	p	sub_2FDEE2C+37	call mw_decrypt_str_wrap; ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/
Down	p	sub_2FDEE2C+48	call mw_decrypt_str_wrap; ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
Down	p	sub_2FE090C+37	call mw_decrypt_str_wrap
Down	p	sub_2FE1534+E	call mw_decrypt_str_wrap
Down	p	sub_2FE1534+1C	call mw_decrypt_str_wrap
Down	p	sub_2FE1580+24	call mw_decrypt_str_wrap; true
Down	p	sub_2FE1580+3D	call mw_decrypt_str_wrap; false
Down	p	sub_2FE2144+24	call mw_decrypt_str_wrap; Kernel32.dll
Down	p	sub_2FE21BC+21	call mw_decrypt_str_wrap; GetProductInfo
Down	p	sub_2FE2280+20	call mw_decrypt_str_wrap; GetNativeSystemInfo
Down	p	sub_2FE2330+21	call mw_decrypt_str_wrap; VerifyVersionInfoW
Down	p	sub_2FE23E8+2F	call mw_decrypt_str_wrap; VerSetConditionMask
Down	p	sub_2FE24CC+21	call mw_decrypt_str_wrap; IsWow64Process
Down	p	sub_2FE355C+36	call mw_decrypt_str_wrap; ole32.dll
Down	p	sub_2FE355C+57	call mw_decrypt_str_wrap; ole32.dll
Down	p	sub_2FE355C+85	call mw_decrypt_str_wrap; CoCreateInstance
Down	p	sub_2FE355C+AF	call mw_decrypt_str_wrap; CoCreateInstanceEx
Down	p	sub_2FE355C+D9	call mw_decrypt_str_wrap; CoInitialize
Down	p	sub_2FE355C+103	call mw_decrypt_str_wrap; CLSIDFromProgID
Down	p	sub_2FE355C+12D	call mw_decrypt_str_wrap; CoUninitialize
Down	p	sub_2FE378C+20	call mw_decrypt_str_wrap; isrCOM error:
Down	p	Comobj::EOleSysErro...	call mw_decrypt_str_wrap; OLE error %.8x
Down	p	sub_2FE3E90+283	call mw_decrypt_str_wrap; Arabic
Down	p	sub_2FE3E90+294	call mw_decrypt_str_wrap; Bulgarian
Down	p	sub_2FE3E90+2A5	call mw_decrypt_str_wrap; Catalan
Down	p	sub_2FE3E90+2B6	call mw_decrypt_str_wrap; Chinese
Down	p	sub_2FE3E90+2C7	call mw_decrypt_str_wrap; Czech
Down	p	sub_2FE3E90+2D8	call mw_decrypt_str_wrap; Danish
Down	p	sub_2FE3E90+2E9	call mw_decrypt_str_wrap; German
Down	p	sub_2FE3E90+2FA	call mw_decrypt_str_wrap; Greek
Down	p	sub_2FE3E90+30B	call mw_decrypt_str_wrap; English
Down	p	sub_2FE3E90+31C	call mw_decrypt_str_wrap; Spanish
Down	p	sub_2FE3E90+32D	call mw_decrypt_str_wrap; Finnish
Down	p	sub_2FE3E90+33E	call mw_decrypt_str_wrap; French
Down	p	sub_2FE3E90+34F	call mw_decrypt_str_wrap; Hebrew
Down	p	sub_2FE3E90+360	call mw_decrypt_str_wrap; Hungarian
Down	p	sub_2FE3E90+371	call mw_decrypt_str_wrap; Icelandic
Down	p	sub_2FE3E90+382	call mw_decrypt_str_wrap; Italian
Down	p	sub_2FE3E90+393	call mw_decrypt_str_wrap; Japanese
Down	p	sub_2FE3E90+3A4	call mw_decrypt_str_wrap; Korean
Down	p	sub_2FE3E90+3B5	call mw_decrypt_str_wrap; Dutch
Down	p	sub_2FE3E90+3C6	call mw_decrypt_str_wrap; Norwegian
Down	p	sub_2FE3E90+3D7	call mw_decrypt_str_wrap; Polish
Down	p	sub_2FE3E90+3E8	call mw_decrypt_str_wrap; Portuguese
Down	p	sub_2FE3E90+3F9	call mw_decrypt_str_wrap; Romansh

Line 1 of 4105

End.

m4n0w4r