PROSOL MRLAB
TECHNOLOGY POWER HOUSE

Azərbaycan Respublikası Xüsusi Rabitə və İnformasiya Təhlükəsizliyi Dövlət Xidməti - Kompüter İnsidentlərinə qarşı Mübarizə Mərkəzi - Malware Research Lab - F.Cəfərov, S.Abasov

Tapşırıq 1

Bu tapşırıqda iştirakçılara aşağıda ki, rəsm fayl təqdim edilmişdir.

Şəkildə iştirakçılara ascii kodlar təqdim edilib. Kiçik bir python skripti yazaraq məlumatın nə olduğunu təyin etmək olar.

>>> data = [97, 68,
77,120,77,84,66,102,86,122,66,121,98,69,82,65,81,122,78,121,100,67,53,110,77,72,89,117,89,88,111,61]

>>>

>>> ba = bytearray(data)

>>> ba

bytearray(b'aDMxMTBfVzBybERAQzNydC5nMHYuYXo=')

>>> import base64

>>> base64.b64decode(ba)

h3110_W0rlD@C3rt.g0v.az : 027cde3cb5ea8935e84ee82728d412d0

Tapşırıq 2

Bu tapşırıq zamanı iştirakçılardan tərkibində gizli kod daşıyan pdf faylı analiz edərək içərisindən bayrağı əldə etməkdir.

İlkin olaraq fayl obyektləri haqqında qısa məlumat alırıq. Bunun üçün pdf-python alətindən istifadə etmək olar.

pdf-parser.py ctf2.pdf -w > pdf_objects.txt

Daha sonra içərisində stream daşıyan obyetkləri dump edirik.

obj **15** 0

Type:

Referencing:

Contains stream

<<

/Filter [/FlateDecode]

/Length 462

>>

Bundan sonra -c əmri ilə obyektin saxladığı məlumata baxırıq.

```
obj 15 0
 Type:
 Referencing:
 Contains stream

  <<
    /Filter [/FlateDecode]
    /Length 462
  >>
 "H\x89\x9c\x93]o\xda0\x14\x86\xaf\xa9\xd4\xff`E\xda\x94\x08\x94\xd1@\xd9\xbaB%`\xf4\x13\xe8\x80\x8e\xb5\x9bv\x11\xec\x9
3\xc44\xb13\xc7\x01\n\xeb\x7f\x9f\x13\xc3Rv\xb1i;\x17\xd1\xf19\xc7\xef\xf3\xdaI\xbc\x94aI9Cm\xc2g\xd0\x8bf@\x08\x10\xd3:
<\xd8\x94\x16\xae@\x84\xfaT&\xa8\x85\x8c\xea\x91S\xab\x1f7\xde\xbe;iw\xba\x1fz\xe7\x17\x97W\xd77\xfd\xc1\xf0\xf6\xe3h<\x
b9\xfb4\xfd|\xff\xf0\xc5\x9da\x02\x9e\x1f\xd0\xf9c\x181\x1e\x7f\x17\x89L\x17\xcb\xd5\xd3\xba\xfc\xc6\xa8\x1c\x1e \x158\x
15\x15\x14\x0bXT\xb4\xf80\x8d\xb6\x1d\xda\xaan3\x01I\x1aJE\xfd\xfaMUJ\x12V\xd9\xc2\xc0\x17\xe3\xe8\xbev\xed\x91\xf9p\x02
\xd3\x93\xf0a\x1a\xc7\xa3\xde`\x8d\xd9(\x1d\xaf;N?:o\x18\xa7j\x83VY\x064\x04dR\xd4D\x99\x82\x1d\x02\xf3e`mt\xb7\xa4\x8c(
Q}@\x9b2\x02\xab[\xcf\xcc\x07q\xe0\x8a\xb64\xa9e\x9d\xea\xd9,vf\xd5\x1e\x8a^\xa1\xfa\x8bV\x91%K*q`\xeef\xad\xa2\xb3)\xd2
\x12v\x13@G\xef\x8bJ^\xd5\x87\xb6\xe34\t\xcc\x89\x14\x94\xf9\xb6'x\xd4Uf\xba\x9c\x80\x99\xdd\x19j6\x91\x83~d\x97\x88\xce
\xceP}\xcf\xe1.f\x02\xdc\xc7\xdf\xea9\xd2\xf9W\xa4f\xbeF\xd5U\xd5\xb32v\xbd`;\x7f`\xef_\xca/~\xed\x7f\xf9\xb5\x1c\xde\xd
0\xf0\xbfp\xf7\xa2\x18|.\xd2\\\xb4\x95I\xbd\x10\xa2\xe5\xf2v\xa5\x9fj>\xff\x01\x94E5\xbb5:\xe7\x94\x99\x86\x91\x19\xc0\x
9c%<\x04;\xe4\xbe\xa9\xba\xeaM\x13\x8e\xd3\x08\x98\xb4}\x90\xbd\x10\xb2\xb4\xf3tEL\x83@\xc4\rK}b\x0c\xc4\xe5\xdd\xa0\xaf
\xf5\x94\xc6\xf3O\x01\x06\x00\xd2\xff\xe9\xba\r\n"
```

-f əmri ilə məlumat decode edilir (FlatDecode).

```
 b'function AdobeEmbedded()\r\n{\tvar digits = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/",\r\n
  cur, prev, digitNum,\r\n    i=0,\r\n    result = [],\r\n\ttext = "cGRmX3JfdjNSeV9lYVppQEMzcnQuRzB2LmF6";\r\n\t\r\n
while (i < text.length){\r\n    \tcur = digits.indexOf(text.charAt(i));\r\n        digitNum = i % 4;\r\n        \r\n
    switch(digitNum)\r\n        {\r\n        \tcase 1:\r\n            \tresult.push(String.fromCharCode(prev << 2 | cur
>> 4));\r\n                break;\r\n            case 2:\r\n            \tresult.push(String.fromCharCode((prev & 0x0f)
<< 4 | cur >> 2));\r\n                break;    \r\n            case 3:\r\n            \tresult.push(String.fromCharCode
((prev & 3) << 6 | cur));\r\n                break;                \r\n        }\r\n        prev = cur;\r\n        i++;
\r\n    \r\n    }\r\n\tvar res = result.join("");\r\nconsole.log(res)\r\ndocument.getElementById("demo").innerHTML = res
;\r\n}'
```

-o 15 -f -d script.js əmri ilə decode edilən məlumat fayla yazılır.

```
function AdobeEmbedded()
{   var digits =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/",
   cur, prev, digitNum,
   i=0,
   result = [],
   text = "cGRmX3JfdjNSeV9lYVppQEMzcnQuRzB2LmF6";

   while (i < text.length){
      cur = digits.indexOf(text.charAt(i));
      digitNum = i % 4;

      switch(digitNum)
      {
         case 1:
            result.push(String.fromCharCode(prev << 2 | cur >> 4));
            break;
         case 2:
            result.push(String.fromCharCode((prev & 0x0f) << 4 | cur >> 2));
            break;
         case 3:
            result.push(String.fromCharCode((prev & 3) << 6 | cur));
            break;
      }
      prev = cur;
      i++;

   }
   var res = result.join("");
console.log(res)
document.getElementById("demo").innerHTML = res;
}
```

Skripti analiz etdiyimiz zaman text dəyişənində olan məlumatı base64 metodu ilə decode etdiyini görə bilərik.

```
>>> base64.b64decode(b'cGRmX3JfdjNSeV9lYVppQEMzcnQuRzB2LmF6')
```

pdf_r_v3Ry_eaZi@C3rt.G0v.az : 8374ad8377673851ed2f8db43ab4b6d2

Tapşırıq 3 - Casuslar körpüsünə xoş gəlmisiniz, cənab James Donovan

Yarışma zamanı iştirakçılara cert.gov.az adında 64 bit arxitekturalı icra edilə bilən fayl təqdim edilmişdir.

MD5: A99E25AB17FDD920B2E31D784BC8A634

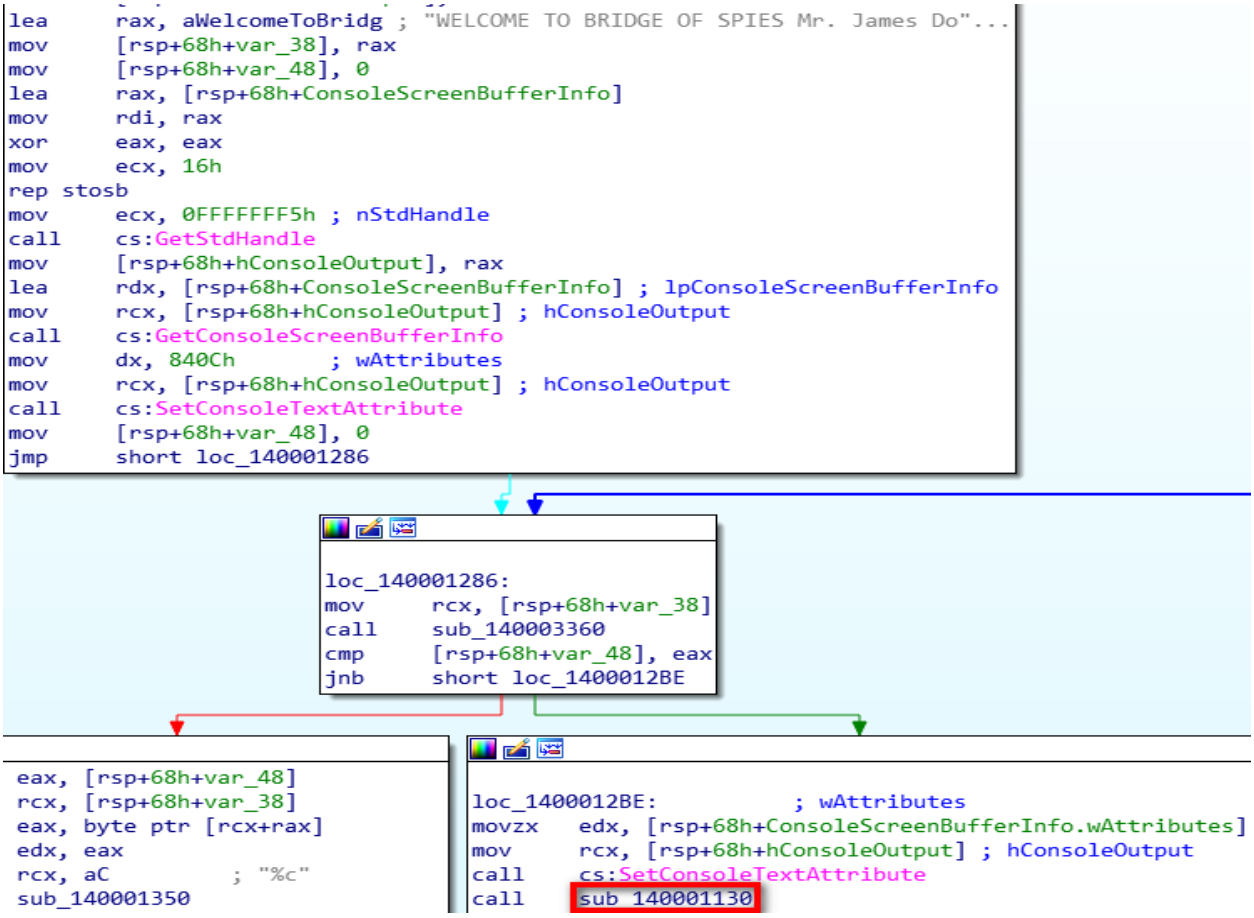
SHA-1: 46353661A58E630892155E03905446F6DA5FB5E2

Proqramı komanda xətti (command line) üzərindən icra edərkən iştirakçılar aşağıda ki, mətn ilə qarşılaşacaqlar.

```
C:\Users\      \Desktop\CTF\PROSOL\t1>certgovaz.exe
WELCOME TO BRIDGE OF SPIES Mr. James Donovan
```

Bu mesaj əslində iştirakçılara ipucu vermək məqsədi ilə komanda xətti ekranına yazılır. Daha ətraflı: https://en.wikipedia.org/wiki/Bridge_of_Spies_(film)

Əsas funksiyanın analizi - "main" funksiyası olaraq adlandırılan funksiya yuxarıda gördüyünüz mesajı ekrana yazdırdıqdan sonra sub_140001130 funksiyasını çağırır.



Bu funksiya isə öz növbəsində ilk olaraq CreateToolhelp32Snapshot funksiyasını çağırır. Əməliyyat zamanı funksiya aşağıda ki, parametrlər ilə çağrılır.

```
xor     edx, edx        ; th32ProcessID
mov     ecx, 2          ; dwFlags
call    CreateToolhelp32Snapshot
```

dwFlags = 2 (TH32CS_SNAPPROCESS)
th32ProcessID = 0 (fəaliyyət göstərən bütün proseslər)

Ardınca isə Process32First və Process32Next funksiyalarına müraciət olunur. Proqram bu funksiyaların köməkliyi ilə sistemdə fəaliyyət göstərən proseslərin siyahısını alır. Daha sonra lea rcx, [rsp+178h+pe.szExeFile] ilə **PROCESSENTRY32** strukturuna yazılan proses adlarını oxuyur və bunları prosol.exe mətni ilə qarşılaşdırır - yəni sistemdə fəaliyyət göstərən proseslər arasında prosol.exe adında proses axtarır.

Tapılmadığı təqdirdə proqram özünü sonlandırır. Əgər sistemdə prosol.exe adında proses fəaliyyət göstərir isə, **sub_140001000** funksiyası çağrılır.

Bu funksiya ilk olaraq CreateFile köməkliyi ilə \\\\.\\pipe\\bridgeofspies borusuna (pipe - bridge 😊) qoşulmağa cəhd edir. Əgər qoşulma prosesi uğurlu olarsa WriteFile funksiyası üzərindən boruya lokal buferdə olan 20 baytlıq məlumatı yazılır. Əks halda proqram özünü sonlandırır.

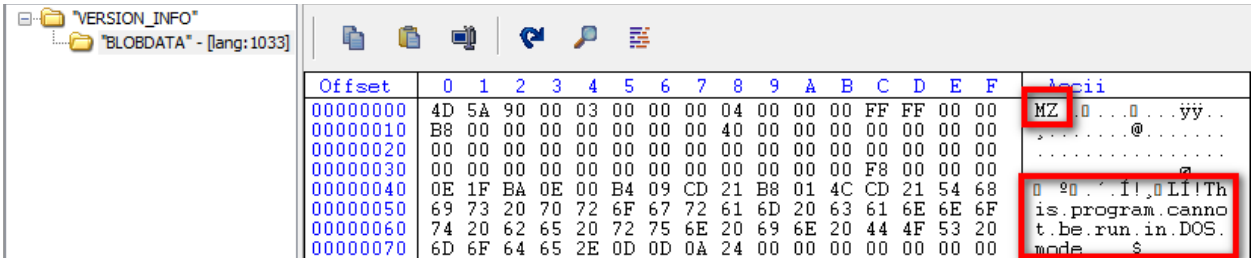
0x12, 0x0E, 0x26, 0x08, 0x0D, 0x39, 0x1F, 0x56, 0x13, 0x39, 0x0B, 0x14, 0x39, 0x02, 0x56, 0x08, 0x56, 0x10, 0x26, 0x08

```
mov     [rsp+78h+NumberOfBytesWritten], 0
mov     [rsp+78h+Buffer], 12h
mov     [rsp+78h+var_27], 0Eh
mov     [rsp+78h+var_26], 26h
mov     [rsp+78h+var_25], 8
mov     [rsp+78h+var_24], 0Dh
mov     [rsp+78h+var_23], 39h
mov     [rsp+78h+var_22], 1Fh
mov     [rsp+78h+var_21], 56h
mov     [rsp+78h+var_20], 13h
mov     [rsp+78h+var_1F], 39h
mov     [rsp+78h+var_1E], 0Bh
mov     [rsp+78h+var_1D], 14h
mov     [rsp+78h+var_1C], 39h
mov     [rsp+78h+var_1B], 2
mov     [rsp+78h+var_1A], 56h
mov     [rsp+78h+var_19], 8
mov     [rsp+78h+var_18], 56h
mov     [rsp+78h+var_17], 10h
mov     [rsp+78h+var_16], 26h
mov     [rsp+78h+var_15], 8
mov     [rsp+78h+hTemplateFile], 0 ; hTemplateFile
mov     [rsp+78h+dwFlagsAndAttributes], 80h ; dwFlagsAndAttributes
mov     [rsp+78h+dwCreationDisposition], 3 ; dwCreationDisposition
xor     r9d, r9d        ; lpSecurityAttributes
mov     r8d, 2          ; dwShareMode
mov     edx, 40000000h  ; dwDesiredAccess
lea     rcx, FileName   ; "\\\\.\\pipe\\bridgeofspies"
call    cs:CreateFileA
mov     [rsp+78h+hFile], rax
cmp     [rsp+78h+hFile], 0FFFFFFFFFFFFFFFFh
jnz     short loc_1400010CB
jmp     short loc_14000110B
-----------------------------------------------------------------

                        ; CODE XREF: sub_140001000+C7↑j
mov     qword ptr [rsp+78h+dwCreationDisposition], 0 ; lpOverlapped
lea     r9, [rsp+78h+NumberOfBytesWritten] ; lpNumberOfBytesWritten
mov     r8d, 14h        ; nNumberOfBytesToWrite
lea     rdx, [rsp+78h+Buffer] ; lpBuffer
mov     rcx, [rsp+78h+hFile] ; hFile
call    cs:WriteFile
```

cert.gov.az faylının icra etdiyi əməliyyatlar bunlardan ibarətdir. Lakin burada kiçik bir problem var. Boruya göndərilən şifrəli məlumatın kimin tərəfindən deşifrə ediləcəyi məlum deyil. Yuxarıda qeyd edildiyi kimi proqram sistemdə fəaliyyət göstərən proseslər arasında prosol.exe adında proses axtarır. İştirakçı növbəti mərhələdə məhz bu faylı tapıb yarışmaya davam etməlidir.

Prosol.exe

Prosol.exe faylı cert.gov.az faylının resurs bölməsində gizlədilmişdir. İştirakçı resurs bölməsindən prosol.exe faylını tapmalı və növbəti əməliyyatları bu fayl üzərində həyata keçirtməlidir.



Resurs bölməsində saxlanılan 64 bitlik prosol.exe.

MD5: 05AED870FD2DCBC6BD5F36D78CFFF319

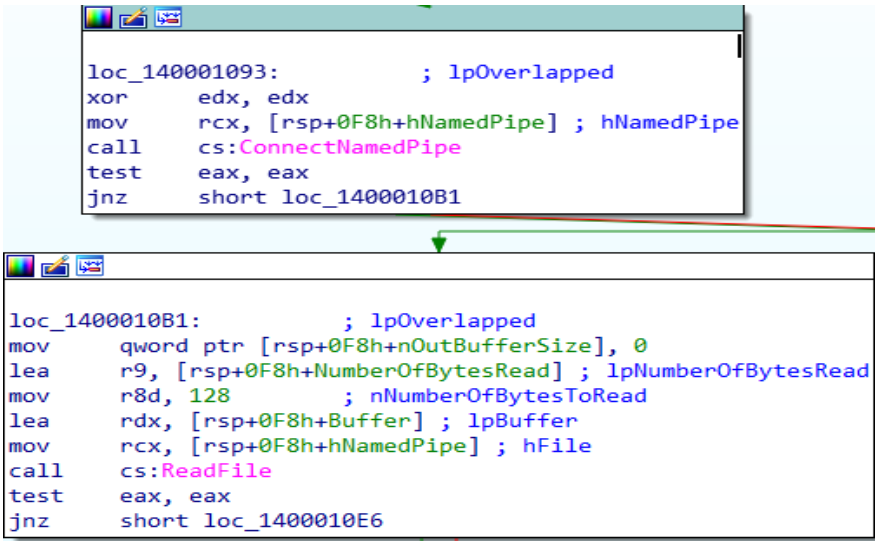
SHA-1: 7A169DB95CE60FF66E700AFDE028F9E7894D6DFF

prosol.exe "**main**" funksiyası. Main funksiyasında proqram ilk olaraq **CreateNamedPipe** köməkliyi ilə sistemdə bridgeofspies adında boru obyekti yaradır.

```
lea     rcx, Name       ; "\\\\.\\pipe\\bridgeofspies"
call    cs:CreateNamedPipeA
```

```
C:\Users\      \Desktop>pipelist64 | findstr "bridge."
bridgeofspies                                  1
```

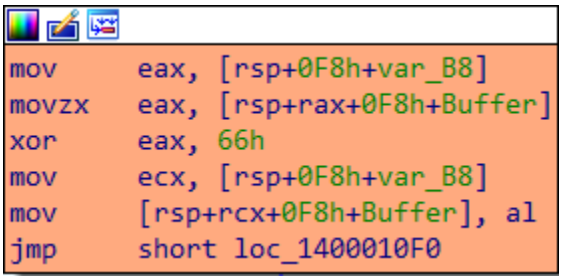
Daha sonra ConnectNamedPipe ilə obyektə (boru) qoşulur və ardınca isə ReadFile funksiyası ilə borudan məlumat oxumağa cəhd edir. Hər hansı xəta baş verər isə proqram özünü sonlandırır.



```
loc_140001093:          ; lpOverlapped
xor     edx, edx
mov     rcx, [rsp+0F8h+hNamedPipe] ; hNamedPipe
call    cs:ConnectNamedPipe
test    eax, eax
jnz     short loc_1400010B1
```

```
loc_1400010B1:          ; lpOverlapped
mov     qword ptr [rsp+0F8h+nOutBufferSize], 0
lea     r9, [rsp+0F8h+NumberOfBytesRead] ; lpNumberOfBytesRead
mov     r8d, 128        ; nNumberOfBytesToRead
lea     rdx, [rsp+0F8h+Buffer] ; lpBuffer
mov     rcx, [rsp+0F8h+hNamedPipe] ; hFile
call    cs:ReadFile
test    eax, eax
jnz     short loc_1400010E6
```

Bundan sonra isə boru üzərindən əldə etdiyi məlumatı lokal bufer üzərində xor (exclusive or) əməliyyatı ilə deşifrə edir.
Açar = 0x66



```
mov     eax, [rsp+0F8h+var_B8]
movzx   eax, [rsp+rax+0F8h+Buffer]
xor     eax, 66h
mov     ecx, [rsp+0F8h+var_B8]
mov     [rsp+rcx+0F8h+Buffer], al
jmp     short loc_1400010F0
```

Lakin prosol.exe prosesi deşifrələmə əməliyyatını bir başa bufer üzərində icra edir və iştirakçıya bu haqda heç bir məlumat vermir. İştirakçı deşifrə edilən məlumatı görmək üçün proqramı debug etməli və ya məlumatı əl ilə xor əməliyyatı üzərindən deşifrə etməlidir.

Bayraq:

Address	Hex	ASCII
0000000E1AFEFDC0	74 68 40 6E 6B 5F 79 30 75 5F 6D 72 5F 64 30 6E	th@nk_y0u_mr_d0n
0000000E1AFEFDD0	30 76 40 6E 00 00 00 00 00 00 00 00 00 00 00 00	0v@n............

th@nk_y0u_mr_d0n0v@n : 81c359d26a0c35b46c4ba06b0073bc0e

Tapşırıq 4

Yarışma zamanı iştirakçılara *cert.gov.az* adında 64 bitlik icra edilə bilən fayl təqdim edilmişdir

MD5: E750702B8229874EB0BB7FE142CC07BB

SHA-1: 1A56BC786E5443F3FD55F87ACD858954C31991A2

İştirakçı fayl icra etməyə cəhd edərkən proqramın xəta verdiyini görəcək.

```
Faulting application name: cert.gov.az.exe, version: 0.0.0.0, time stamp: 0x631bd8f3
Faulting module name: unknown, version: 0.0.0.0, time stamp: 0x00000000
Exception code: 0xc0000005
Fault offset: 0x0000000000000000
Faulting process id: 0x24f0
Faulting application start time: 0x01d8cae6b5ece0d0
Faulting application path: C:\Users\ \Desktop\CTF\PROSOL\t2\cert.gov.az.exe
Faulting module path: unknown
Report Id: 15bd1c69-06e1-4f6d-82d5-6b0f633c3787
Faulting package full name:
Faulting package-relative application ID:
```

```
Fault bucket 1941990531927469364, type 5
Event Name: BEX64
Response: Not available
Cab Id: 0

Problem signature:
P1: cert.gov.az.exe
P2: 0.0.0.0
P3: 631bd8f3
P4: StackHash_ac46
P5: 0.0.0.0
P6: 00000000
P7: PCH_34_FROM_ntdll+0x000000000009DB54
P8: c0000005
P9: 0000000000000008
P10:
```

Xətanın hansı səbəbdən və harada baş verdiyini anlamaq üçün proqramı debug etmək lazımdır. Xəta baş verdikdən sonra debugger üzərindən proqram call stack məlumatlarına baxırıq.

```
0000007EE02FF868  00007FF65D1413EF  60  return to cert.gov.az.00007FF65D1413EF from ???
```

Bu adresə gedərək xətanın baş vermə səbəbini öyrənə bilərik.

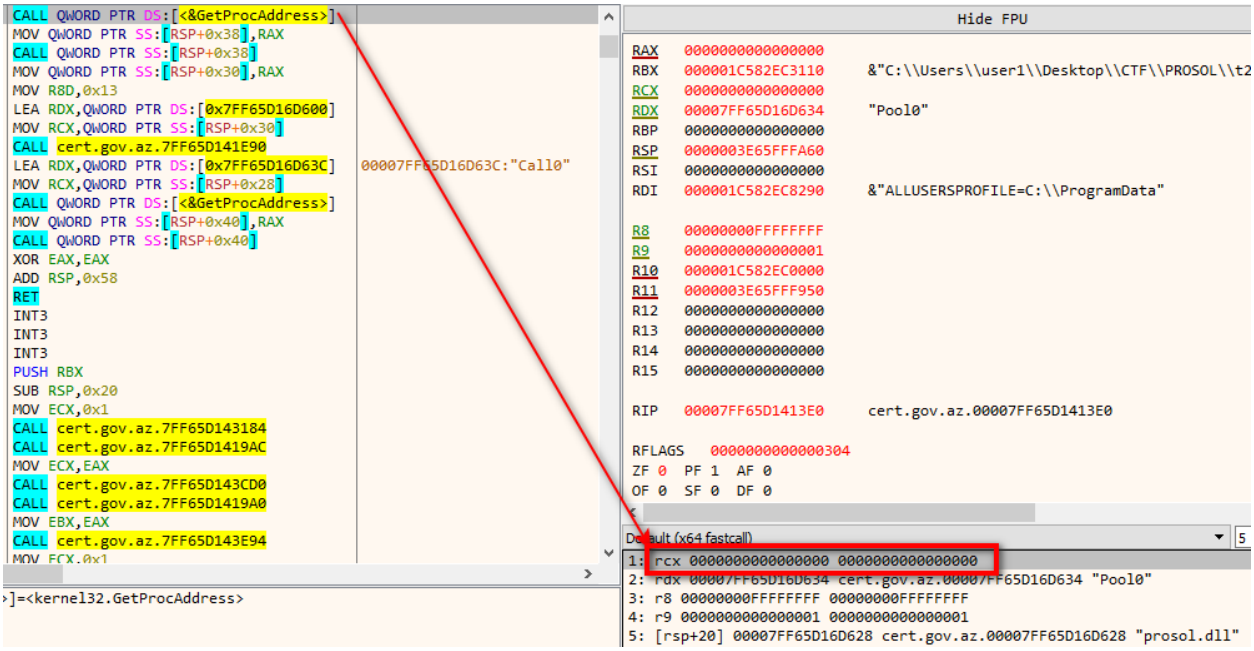
```
00007FF65D1413BC   48:8B5424 20      MOV RDX,QWORD PTR SS:[RSP+0x20]
00007FF65D1413C1   88040A            MOV BYTE PTR DS:[RDX+RCX],AL
00007FF65D1413C4   48:8B4C24 20      MOV RCX,QWORD PTR SS:[RSP+0x20]
00007FF65D1413C9   FF15 51CC0000     CALL QWORD PTR DS:[<&LoadLibraryA>]
00007FF65D1413CF   48:894424 28      MOV QWORD PTR SS:[RSP+0x28],RAX
00007FF65D1413D4   48:8D15 59C20200  LEA RDX,QWORD PTR DS:[0x7FF65D16D634]   00007FF65D16D634:"Pool0"
00007FF65D1413DB   48:8B4C24 28      MOV RCX,QWORD PTR SS:[RSP+0x28]
00007FF65D1413E0   FF15 32CC0000     CALL QWORD PTR DS:[<&GetProcAddress>]
00007FF65D1413E6   48:894424 38      MOV QWORD PTR SS:[RSP+0x38],RAX
00007FF65D1413EB   FF5424 38         CALL QWORD PTR SS:[RSP+0x38]
00007FF65D1413EF   48:894424 30      MOV QWORD PTR SS:[RSP+0x30],RAX
```

Burada funksiya xətasına səbəb olan əməliyyat

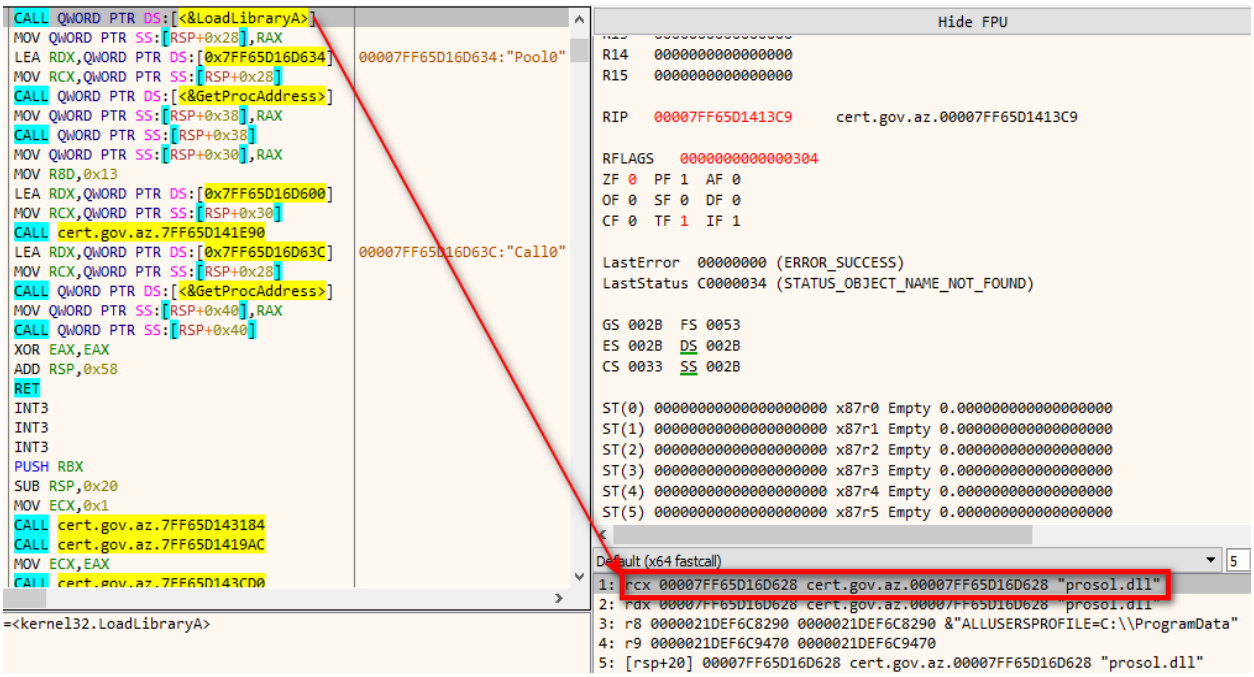
00007FF65D1413EB FF5424 38 CALL QWORD PTR SS:[RSP+0x38]

əməliyyatıdır.

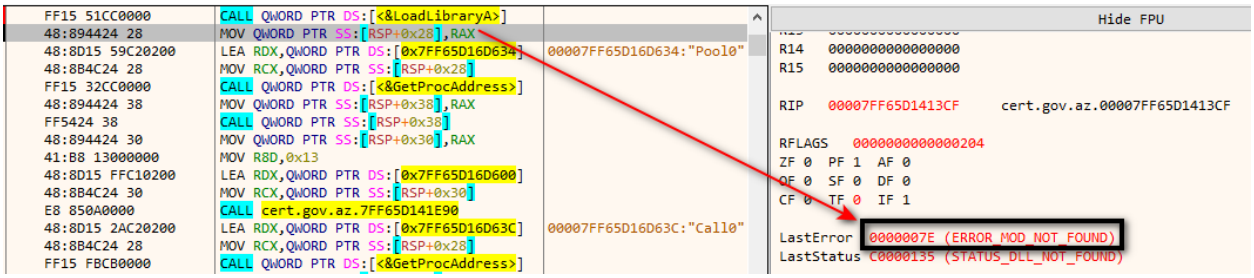
Funksiya ilk olaraq GetProcAddress ilə kitabxana faylının içərisində olan Pool0 adlı funksiya adresini alır və bu adresi lokal dəyişəndə saxlayır. Daha sonra isə həmin funksiyanı çağırır. Məsələni tam olaraq anlamaq üçün GetProcAddress funksiyasına breakpoint təyin edib nələr olduğuna baxaq. Funksiya çağrılarkən parametrləri diqqət ilə analiz etsək burada kiçik bir problemin (anomaliya) olduğunu təyin edə bilərik.



Burada GetProcAddress funksiyasına göndərilən ilk parameter Pool0 funksiyasının olduğu kitabxana faylının HANDLE (HMODULE) dəyəri olmalıdır. Lakin gördüyünüz kimi 0 (NULL) dəyəri göndərilib. Restart edərək proqramı yenidən debug edirik. Lakin bu dəfə LoadLibrary funksiyasına breakpoint təyin edərək.

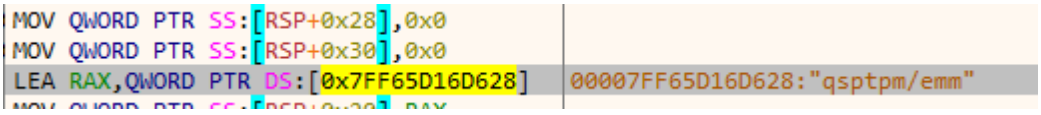


Funksiya LoadLibrary funksiyasının köməkliyi ilə prosol.dll kitabxana faylını öz yaddaşına yükləməyə cəhd edir lakin prosol.dll faylı mövcud olmadığı üçün xəta baş verir. F8 komandası ilə bir növbəti əmri icra edərkən isə xətanın hansı səbəbdən qaynaqlandığını daha aydın şəkildə görürük.



LoadLibrary funksiyasının icrası zamanı **ERROR_MOD_NOT_FOUND** xətası baş verir Proqram prosol.dll kitabxana faylını yükləməyə cəhd edir, lakin kitabxana faylı mövcud olmadığı üçün xəta baş verir.

P.S - Proqram prosol.dll mətnini birbaşa LoadLibrary funksiyasına göndərmir. Bunun üçün kiçik bir kodlaşdırmadan istifadə edir.

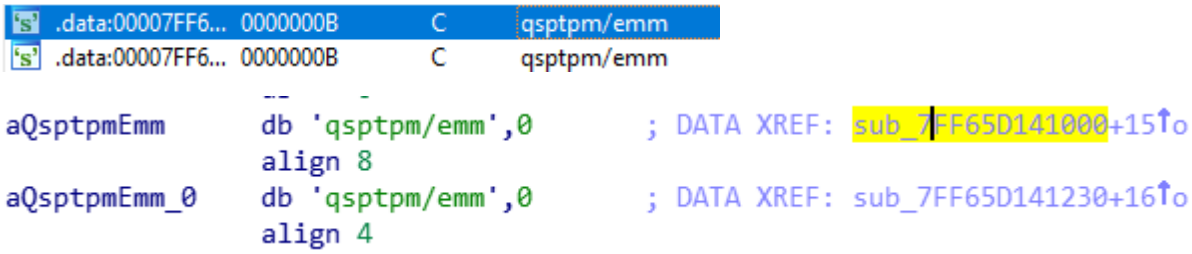


Burada prosol.dll mətnini “**qsptpm/emm**” mətni üzərində kiçik riyazi əməliyyat apardıqdan sonra alır.

Bunun üçün “**qsptpm/emm**” mətninin hər bir xarakteri üzərində -1 əməliyyatını aparır.

dllname[0]	-= 1; //q = p
dllname[1]	-= 1; //s = r
dllname[2]	-= 1; //p = o
dllname[3]	-= 1; //t = s
dllname[4]	-= 1; //p = o
dllname[5]	-= 1; //m = l
dllname[6]	-= 1; /// = .
dllname[7]	-= 1; //e = d
dllname[8]	-= 1; //m = l
dllname[9]	-= 1; //m = l

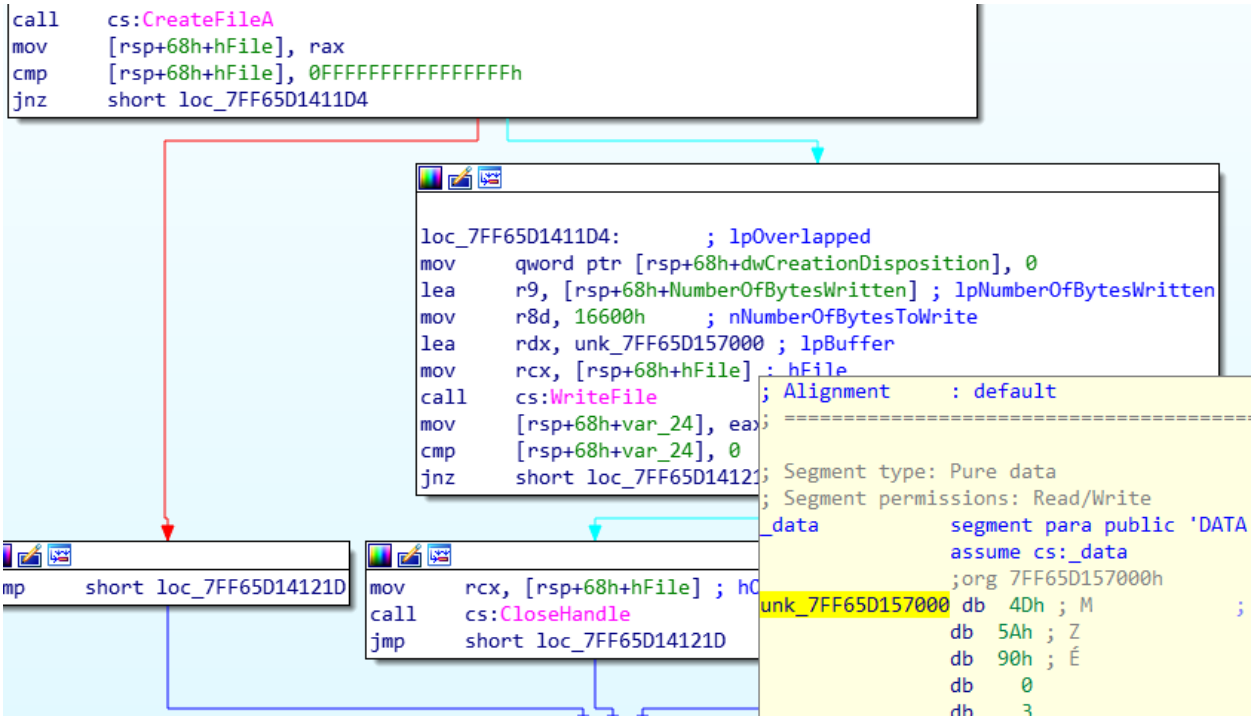
Xətanın baş vermə səbəbini ğyrəndikdən sonra iştirakçılar prosol.dll faylını tapmalıdır. Bunun üçün iştirakçı proqram içərisində “qsptpm/emm” mətnini axtara bilər. Mətinlər bölməsində “qsptpm/emm” mətninə 2 yerdə referans göstərildiyini görürük.



Bunlardan biri bizim hal hazırda analiz etdiyim funksiyadır hansıki LoadLibrary fuksiyası ilə prosol.dll faylını yükləməyə cəhd edir. Digər isə olduqca maraqlı funksiyadır.
sub_7FF65D141000

Funksiya var olmasına baxmayaraq proqram içərisində heç bir yerdə çağrılmır.

Funksiyanı analiz etsək **CreateFile** funksiyası ilə prosol.dll adında fayl yaradılır və **WriteFile** funksiyası ilə fayla **0x16600 – 91648** bayt həcmində **unk_7FF65D157000** buferində saxlanılan məlumatın (PE64) yazılılır.



İştirakçı məlumatı dump edərək prosol.dll faylını yaratmalı və cari proqramın olduğu qovluğa köçürməlidir.

Bundan sonra proqramı yenidən işə salıb analiz etmək lazımdır. LoadLibrary funksiyası ilə prosol.dll kitabxanası yüklənir və GetProcAddress ilə Pool0 funksiyası çağrılır. Gəlin funksiya içərisində hansı əməliyyatların icra edildiyinə baxaq.

```
SUB RSP,0x28
MOV ECX,0x400
CALL <prosol.sub_7FFE4BDC32D8>
MOV QWORD PTR DS:[0x7FFE4BDD6A00],RAX
CMP QWORD PTR DS:[0x7FFE4BDD6A00],0x0
JE prosol.7FFE4BDC103E
MOV R8D,0x400
XOR EDX,EDX
MOV RCX,QWORD PTR DS:[0x7FFE4BDD6A00]
CALL <prosol.sub_7FFE4BDC20E0>
MOV RAX,QWORD PTR DS:[0x7FFE4BDD6A00]
JMP prosol.7FFE4BDC1040
JMP prosol.7FFE4BDC1040
XOR EAX,EAX
ADD RSP,0x28
RET
```

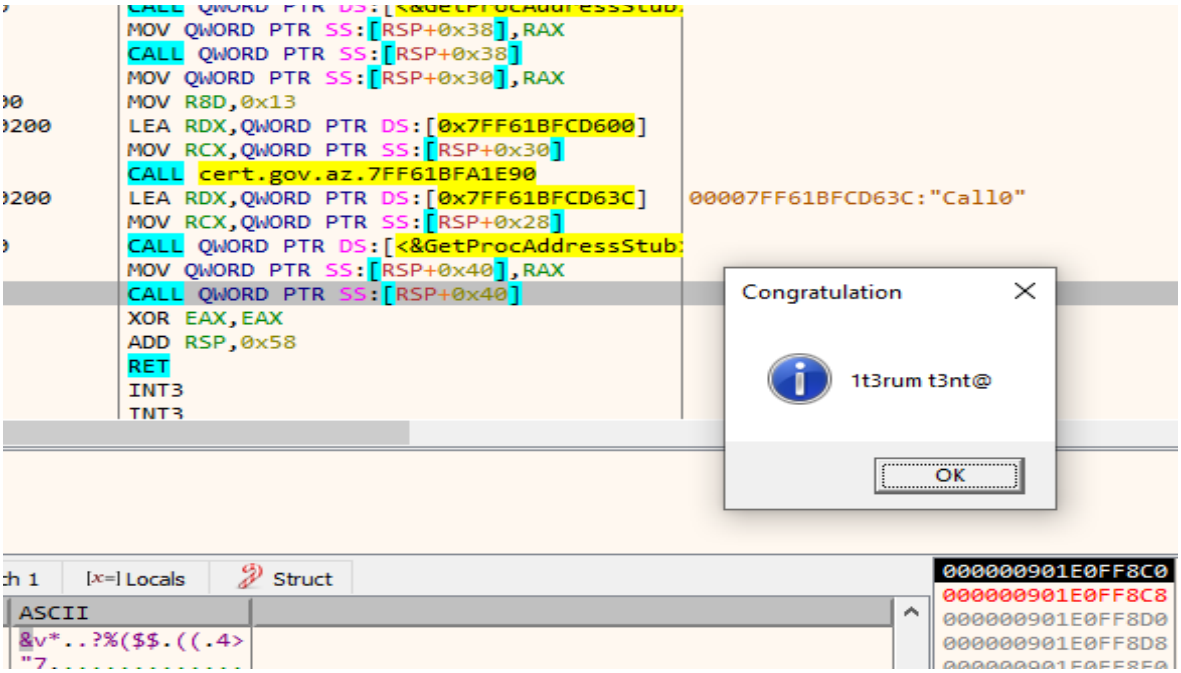
Burada ECX registerinə 0x400 - 1024 dəyəri mənimsədilir və dll içərsiində 7FFE4BDC32D8 funksiyası çağrılır. Funksiyanı analiz etsək RtlAllocateHeap funksiyasını çağırdığını görə bilərik. DLL proqram yaddaşında 1024 baytlıq yer ayırır və yeni yaradılan yaddaş bölməsinin adresini RAX registeri üzərindən proqrama geri qaytarır.

Bundan sonra isə proqram

00007FF65D141406 E8 850A0000 CALL cert.gov.az.7FF65D141E90
əmri ilə yeni yaradılan bölgəyə 19 baytlıq məlumat köçürür və prosol.dll içərisində olan call0 funksiyasını çağırır.

```
26 76 2A 16 0C 3F 25 28 24 24 07 28 28 13 34 3E  &v*..?%($$.((.4>
22 37 18 00 00 00 00 00 00 00 00 00 00 00 00 00  "7..............
```

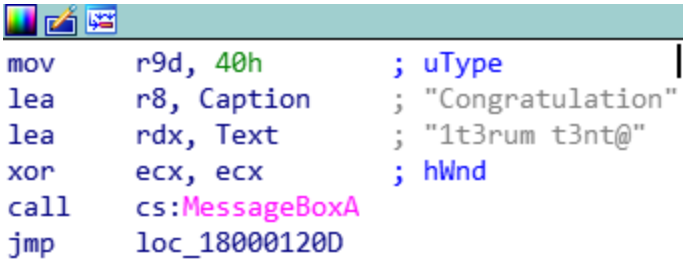
Bu funksiya çağrıldıqdan sonra isə ekrana aşağıda ki, mesaj yazdırılır.



“**Iterum Tenta**” latın dilindən dilimizə “bir daha sınayın” kimi tərcümə edilir 😊

Bayrağı əldə etmək üçün Call0 funksiyası diqqətli şəkildə analiz edilməlidir. Sözü gedən funksiya ilk olaraq **GetComputerName** ilə cari kompüter adını əldə edir. Daha sonra isə əldə edilən kompüter adının uzunluğunun 8 -ə bərabər olub olmadığı yoxlanılır.

Əgər uzunluq 8-ə bərabər olmasa aşağıda ki, kodlar icra edilir.

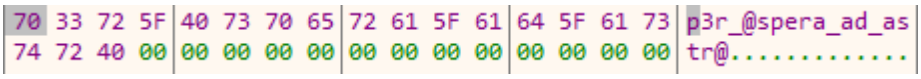


```
mov     r9d, 40h        ; uType
lea     r8, Caption     ; "Congratulation"
lea     rdx, Text       ; "1t3rum t3nt@"
xor     ecx, ecx        ; hWnd
call    cs:MessageBoxA
jmp     loc_18000120D
```

Əgər uzunluq 8-ə bərabərdirsə kompüter adını “VEXILLUM” mətni ilə qarşılaşdırır.

```
mov     eax, 1
imul    rax, 0
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'V'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 1
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'E'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 2
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'X'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 3
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'I'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 4
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'L'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 5
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'L'
jnz     loc_18000120D
mov     eax, 1
imul    rax, 6
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'U'
jnz     short loc_18000120D
mov     eax, 1
imul    rax, 7
movsx   eax, [rsp+rax+488h+Buffer]
cmp     eax, 'M'
```

Daha sonra modul kompüter adını açar olaraq istifadə edir və ayrılan yaddaş bölgəsində mövcud olan məlumatı bu açar ilə (xor) deşifrə edir. İştirakçılar bayrağı əldə edə bilmək üçün kompüter adını “VEXILLUM” olaraq dəyişdirməli və ya GetComputerNameA funksiyasına müdaxilə etməlidirlər.



p3r_@spera_ad_astr@ : 34789629f78925937a67e93226742fcc

Tapşırıq 5

Bu tapşırıq zamanı iştirakçılara 2 ədəd fayl təqdim edilir. **blabla.exe** və **qr.zip**

İştirakçılardan bu tapşırıq zamanı zədələnmiş zip faylını bərpa etmələri gözlənilir.

blabla.exe 32 bitlik icra edilə bilən fayldır. qr.zip məhz bu 32bitlik proqram tərfindən korlanmışdır. İştirakçı blabla.exe faylını analiz edərək zip faylının necə korlandığını öyrənib faylı bərpa etməlidir. Burada PKZIP formatı iştirakçılara kömək edə bilər.

https://en.wikipedia.org/wiki/ZIP_(file_format)

Proqramı debug edərkən ilkin olaraq CreateFile funksiyası ilə qr.zip faylını açmaq istədiyini görə bilərik. GetFileSize funksiyası ilə faylın ölçüsü əldə edilir. Daha sonra ReadFile funksiya ilə fayldan 0x1E = 30 bayt həcmində məlumat oxunulur.

```
.   FF15 00D0E200       CALL DWORD PTR DS:[<&CreateFileW>]
.   8945 D8             MOV DWORD PTR SS:[EBP-0x28],EAX
.   837D D8 FF          CMP DWORD PTR SS:[EBP-0x28],0xFFFFFFFF
.v  75 08               JNE damager.E21091
.   83C8 FF             OR EAX,0xFFFFFFFF
.v  E9 57010000         JMP damager.E211E8
>   6A 00               PUSH 0x0
.   8B4D D8             MOV ECX,DWORD PTR SS:[EBP-0x28]
.   51                  PUSH ECX
.   FF15 04D0E200       CALL DWORD PTR DS:[<&GetFileSize>]
.   8945 BC             MOV DWORD PTR SS:[EBP-0x44],EAX
.   6A 00               PUSH 0x0
.   8D55 C4             LEA EDX,DWORD PTR SS:[EBP-0x3C]
.   52                  PUSH EDX
.   6A 1E               PUSH 0x1E
.   8D45 DC             LEA EAX,DWORD PTR SS:[EBP-0x24]
.   50                  PUSH EAX
.   8B4D D8             MOV ECX,DWORD PTR SS:[EBP-0x28]
.   51                  PUSH ECX
.   FF15 08D0E200       CALL DWORD PTR DS:[<&ReadFile>]
```

Oxunan 30 baytlıq məlumat ZIP Headeridir. Aşağıda PKZIP header strukturu göstərilmişdir.

```
struct ZIPFILERECORD
{
    char frSignature[4];
    unsigned short frVersion;
    unsigned short frFlags;
    unsigned short frCompression;
    unsigned short frFileTime;
    unsigned short frFileDate;
    unsigned int frCrc;
    unsigned int frCompressedSize;
    unsigned int frUncompressedSize;
    unsigned short frFileNameLength;
    unsigned short frExtraFieldLength;
};
```

Strukturu debugger proqram təminatına mənimsədib davam edirik. Növbəti əməliyyat ZIPFILERECORD strukturundan frCompressedSize dəyərini götürür və eyni ölçüdə yaddaşda

yer ayırır. Bundan sonra kiçik bir hesablama ilə sıxışdırılmış məlumatın başladığı offsetə gedərək buradakı məlumatı biraz öncə ayırdığı yaddaş bölgəsinə köçürür.

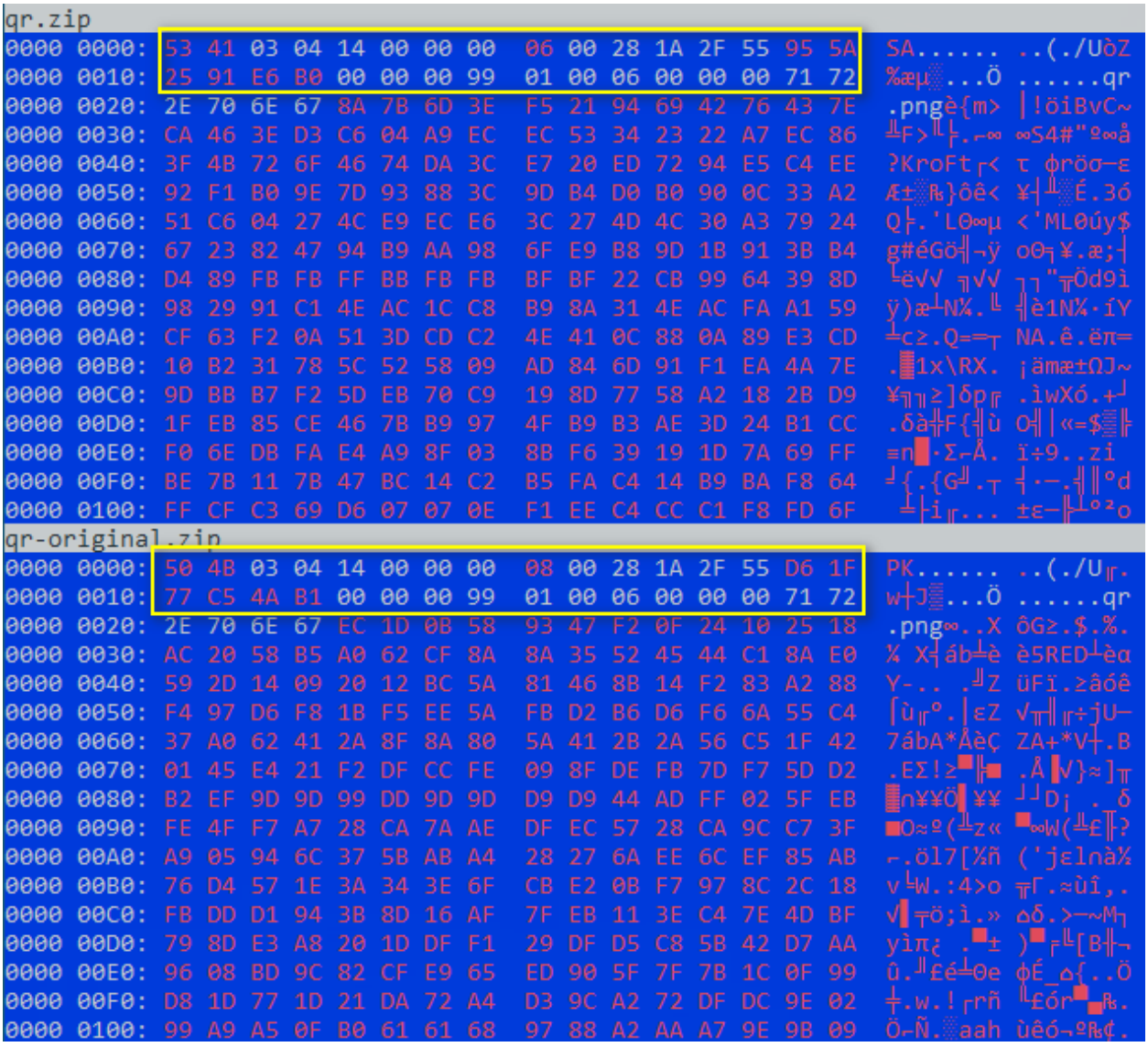
```
CALL <damager.sub_E22C60>                                malloc
ADD ESP,0x4
MOV DWORD PTR SS:[EBP-0x30],EAX
MOVZX ECX,WORD PTR SS:[EBP-0xA]
MOVZX EDX,WORD PTR SS:[EBP-0x8]
LEA EAX,DWORD PTR DS:[ECX+EDX+0x1E]
MOV DWORD PTR SS:[EBP-0x38],EAX
PUSH 0x0
PUSH 0x0
MOV ECX,DWORD PTR SS:[EBP-0x38]
PUSH ECX
MOV EDX,DWORD PTR SS:[EBP-0x28]
PUSH EDX
CALL DWORD PTR DS:[<&SetFilePointer>]
PUSH 0x0
LEA EAX,DWORD PTR SS:[EBP-0x3C]
PUSH EAX
MOV ECX,DWORD PTR SS:[EBP-0x12]
PUSH ECX
MOV EDX,DWORD PTR SS:[EBP-0x30]
PUSH EDX
MOV EAX,DWORD PTR SS:[EBP-0x28]
PUSH EAX
CALL DWORD PTR DS:[<&ReadFile>]
```

Məlumatı oxuduqdan sonra isə eyni bufer üzərində sıxışdırılmış datanı 0x66 açarı ilə şifrələyir və şifrələnmiş məlumatı orjınal məlumat ilə dəyişdirir. Bundan sonra yeniden 0-cı ofsetə qayıdaraq ZIPFILEHEADER strukturu üzərində dəyişiklər edir və dəyişiklikləri fayla yazır. Beləliklə FILEHEADER məlumatlarını korlayır.

```
MOV ECX,0x1
IMUL EDX,ECX,0x0
MOV BYTE PTR SS:[EBP+EDX-0x24],0x53                      53:'S'
MOV EAX,0x1
SHL EAX,0x0
MOV BYTE PTR SS:[EBP+EAX-0x24],0x41                      41:'A'
MOVZX ECX,WORD PTR SS:[EBP-0x1C]
SUB ECX,0x2
MOV WORD PTR SS:[EBP-0x1C],CX
MOV EDX,DWORD PTR SS:[EBP-0x12]
SUB EDX,0x64
MOV DWORD PTR SS:[EBP-0x12],EDX
MOV EAX,DWORD PTR SS:[EBP-0x16]
XOR EAX,0x54524543
MOV DWORD PTR SS:[EBP-0x16],EAX
PUSH 0x0
LEA ECX,DWORD PTR SS:[EBP-0x40]
PUSH ECX
PUSH 0x1E
LEA EDX,DWORD PTR SS:[EBP-0x24]
PUSH EDX
MOV EAX,DWORD PTR SS:[EBP-0x28]
PUSH EAX
CALL DWORD PTR DS:[<&WriteFile>]
```

Proqramın ZIPFILEHEADER strukturunda nələri dəyişdirdiyinə baxaq.

İlkin olaraq ilk 2 bayt ‘S’ və ‘A’ xarakterləri ilə əvəzlənir. Defolt olaraq PKZIP formatında bu 2 bayt ‘PK’ dır. Daha sonra PKZIP strukturunda olan frCompression dəyərinindən 2 rəqəmini çıxır. Orijinal dəyəri tapmaq üçün zədələnmiş faylda var olan frCompression dəyərin üzərinə 2 əlavə etmək lazımdır. Bundan sonra frCompressedSize dəyərindən 100 çıxılır. Orijinal dəyəri tapmaq üçün zədələnmiş faylda olan dəyərin üzərinə 100 əlavə etmək lazımdır. Son olaraq blabla.exe PKZIP headerində olan frCRC dəyərini 0x54524543 ilə xorlayır və bu məlumatları orijinal fayla yazır. Tapşırığı həll etmək üçün iştirakçılar başlanğıc olaraq PKZIP headerini bərpa etməlidirlər.

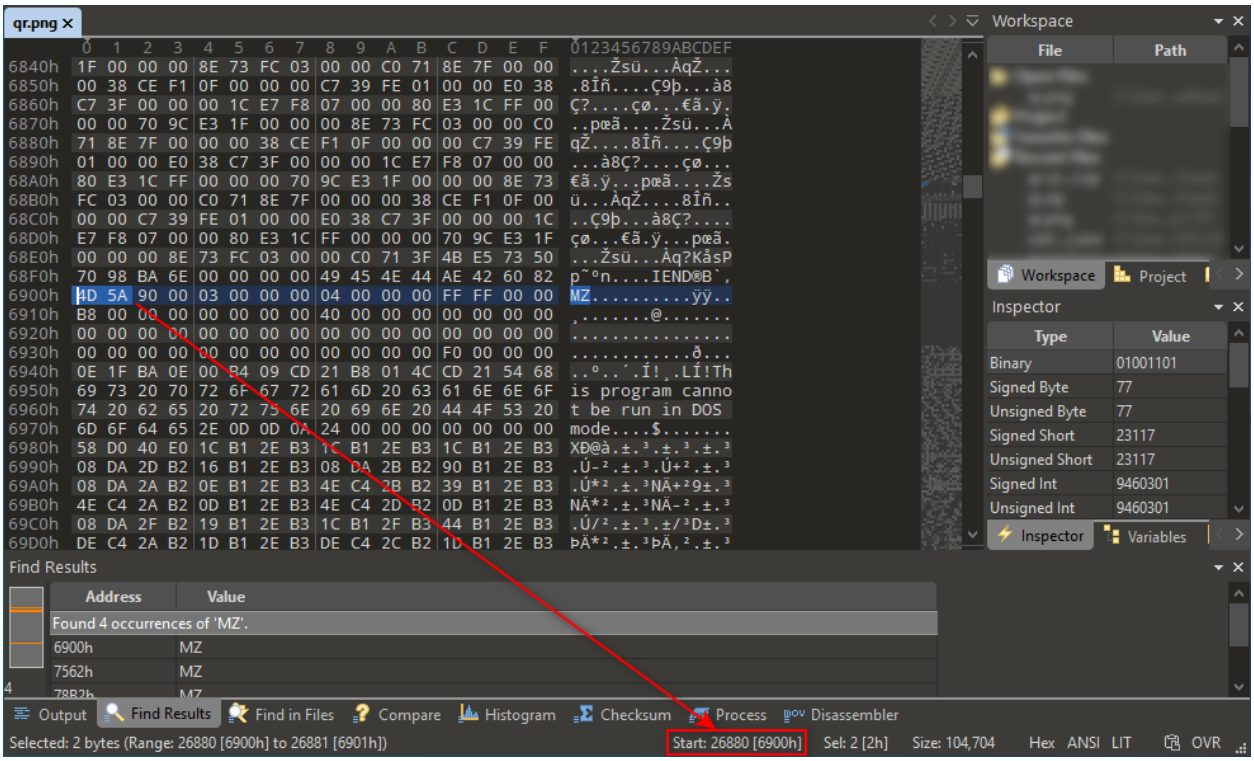


Header bərpa edildikdən sonra isə sıxışdırılmış məlumat deşifrə (xor key = 0x66) edilərək fayla yazılır və qr.zip tam olaraq bərpa edilir. İştirakçı zip faylını açdığı halda aşağıda ki, rəsm ilə qarşılaşacaq.

QR kod skan edildiyi zaman iştirakçı “Məndə sığar iki cahan, mən bu cahanə sığmazam” mesajı ilə qarşılaşacaq. Bu mesaj növbəti əməliyyat üçün iştirakçıya ipucu vermək məqsədi ilə istifadə edilib və rəsm faylının içərisində başqa bir vacib məlumatın olduğuna işarə edir. İştirakçının növbəti hədəfi rəsm fayl içərisində gizlədilən “icra edilə bilən” faylı tapmaqdır. Bunun üçün string dump alətlərindəın istifadə etmək olar.

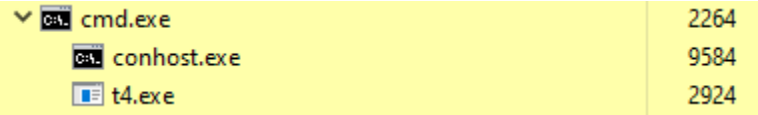
```
ShowWindow
MessageBoxA
USER32.dll
GetStdHandle
IsDebuggerPresent
CloseHandle
GetLastError
WaitForSingleObject
CreateMutexA
Sleep
GetCurrentProcess
```

İştirakçı icra edilə bilən faylı tapdıqdan sonra məlumatı çıxardaraq analizi bu fayl üzərindən davam etməlidir.



Filename	C:/Users/
File Type	Portable Executable 32
MD5	5644A5FE88C1C4039B15A20672B079FC
SHA1	947D77710EB88D2A5AADCE32EAEBF73590A6EB29
SHA256	1D975840167DD4DFF00A01A41E214F444495EB84F171766B4CF192EB27EE3E1B
ssdeep	1536:eSJZ5gRYFpWzltkOpmMoj3biAnfVH2GgtzCF5wEaWisWecdFLiBH14n:F5fult
File Size	77824
Created	Tue Sep 20 05:26:27 2022
Modified	Tue Sep 20 05:26:27 2022
Accessed	Wed Sep 21 23:14:43 2022

Faylı komanda xətti üzərindən icra etməyə cəhd etdiyimiz zaman komanda xətti pəncərəsi ekrandan itəcəkdir. Bu ilk baxışda iştirakçıya proqramın sonlandığı təsirini bağışlasada əslində özünü gizlətmək üçün istifadə edilən kiçik bir fənddir.



Eyni faylı 2-ci dəfə işə salmağa cəhd etsəniz pəncərə gizlənməsədə əməliyyatın icra olunmadığını görəcəksiniz. Nələr baş verdiyini anlamaq üçün kodları disassembly etmək lazımdır.

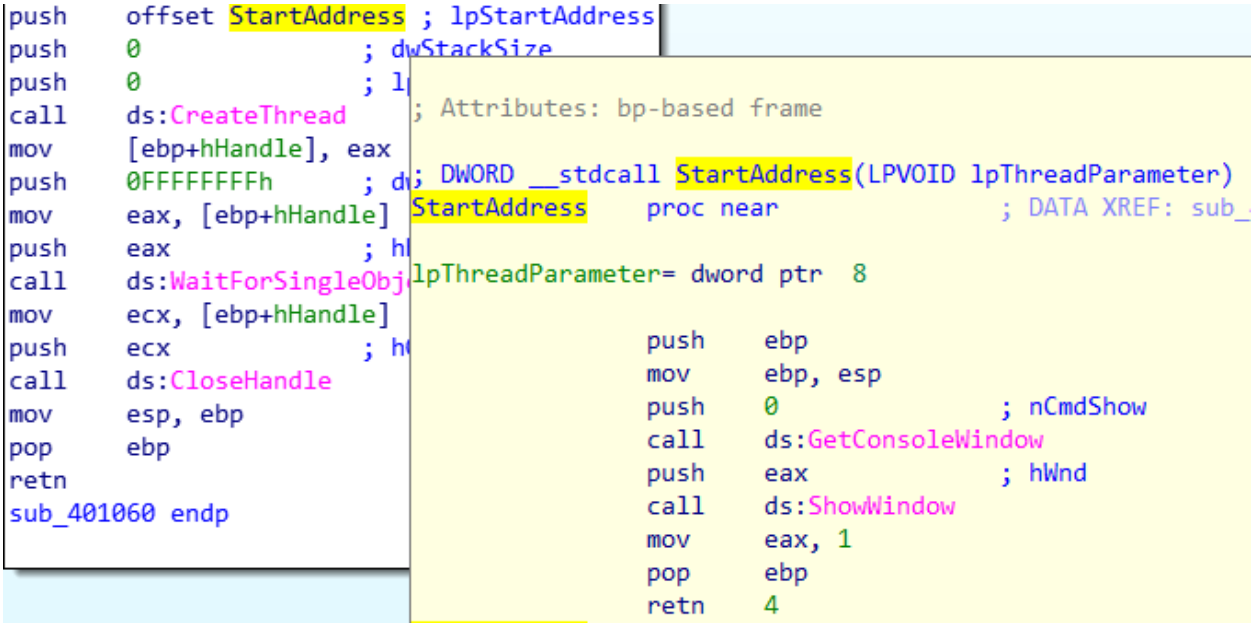
Əsas funksiya sub_401000 içərisinə baxdığımız zaman burada proqramın 2-ci dəfə niyə icra edilmədiyini görürük.

```
push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], 0
push    offset Name     ; "ALONE"
push    1               ; bInitialOwner
push    0               ; lpMutexAttributes
call    ds:CreateMutexA
mov     [ebp+var_4], eax
call    ds:GetLastError
cmp     eax, 0B7h
jnz     short loc_40102E
xor     eax, eax
jmp     short loc_401033
-----------------------------------------------------------

                        ; CODE XREF: sub_401000+28↑j
mov     eax, 1

                        ; CODE XREF: sub_401000+2C↑j
mov     esp, ebp
pop     ebp
retn
endp
```

Proqram ilk dəfə işə salınarkən sistemdə **ALONE** adında mutant obyekt yaradır. Əgər obyekt öncədən mövcud olarsa geriyə 0 dəyərini qaytarır və proqram sonlanır. Əgər proqram ilk olaraq işə salınıbsa növbəti əməliyyalar icra edilir.

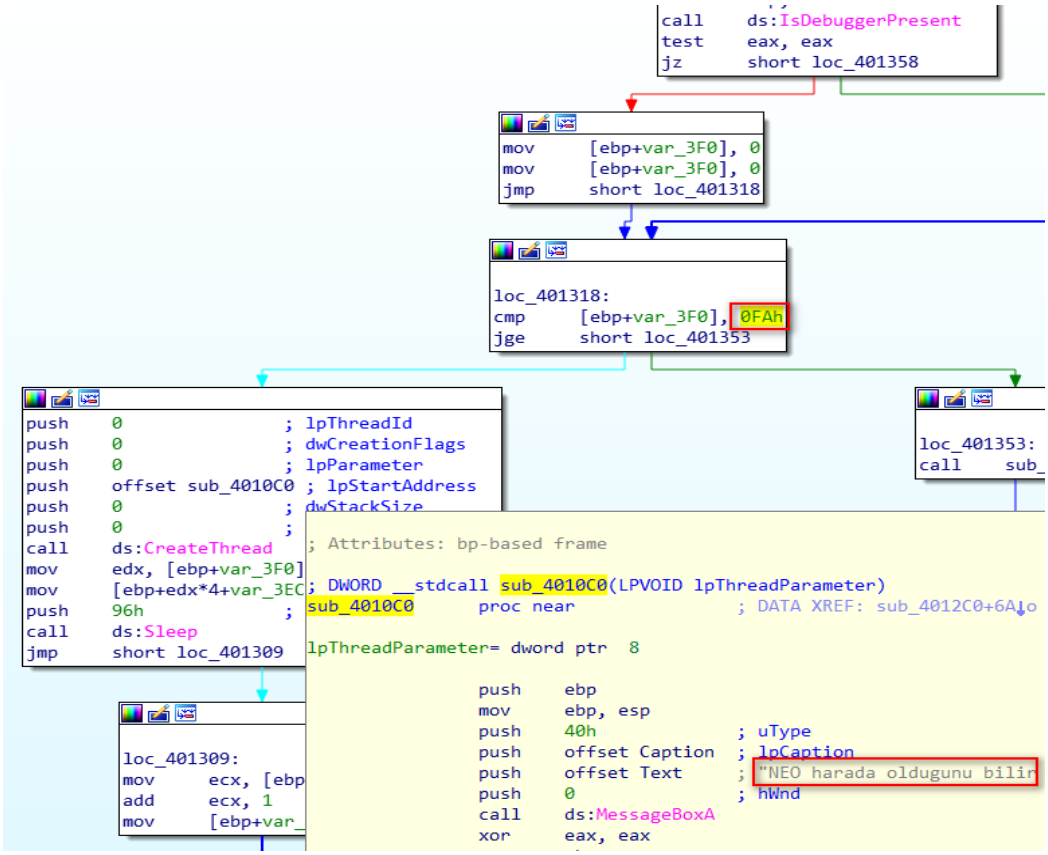
```
loc_401383:
call    sub_401060
call    sub_4012C0
call    sub_401280
```

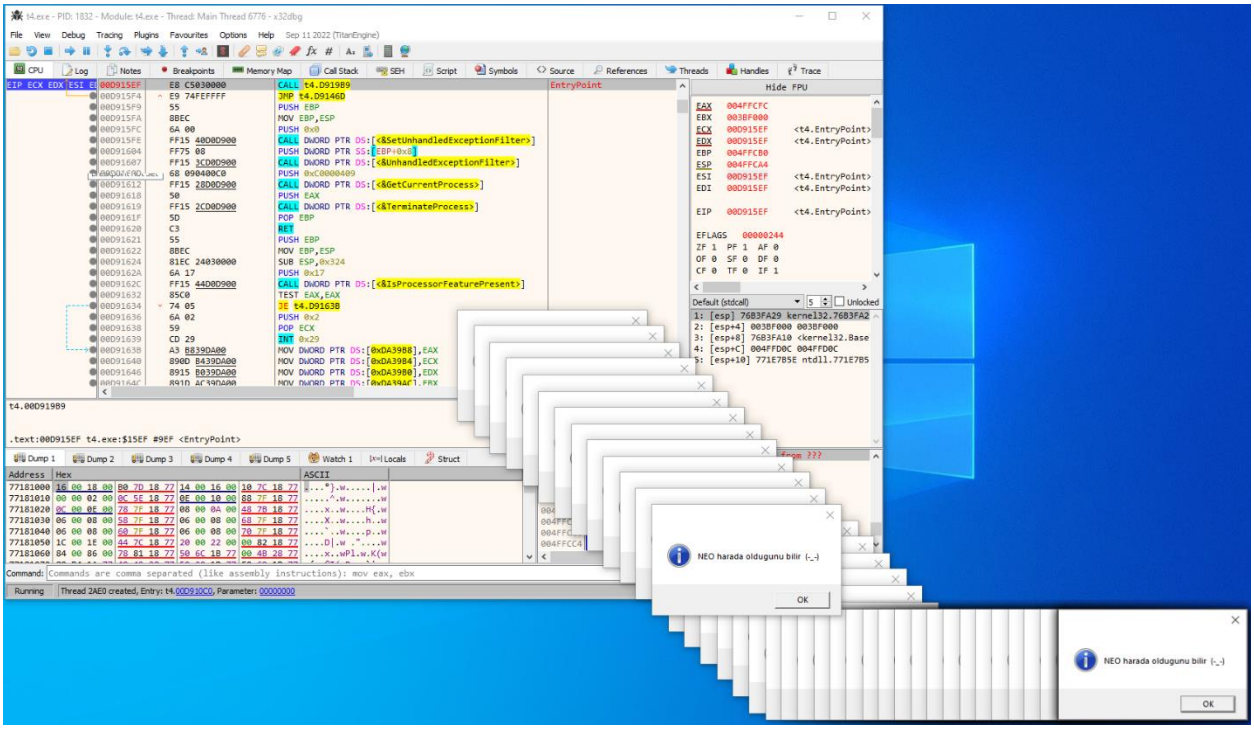
Sub_401060 funksiyası çağrıldığı zaman yeni thread yaradır. Bu thread məhz icra edilən proqram pəncərəsinin gizlədilməsi üçün istifadə edilir.



Pəncərə gizlədildikdən dərhal sonra sub_4012C0 funksiyası çağrılır. Bu funksiya isə kodların debug edilib-edilmədiyini test edir. Əgər proqram debugger tərəfindən işə salınıbsa 250 ədəd yeni thread çağrılaraq ekrana iştirakçı üçün mesaj verilir.

Neo harada oldugunu bilir





Nəhayət sonuncu və iştirakçını bayrağa aparan funksiya - sub_401280. İştirakçı bu funksiyaya gəlmək üçün yuxarıda ki, əməliyyatlardan yayınmalı və ya kodları disasm edərək lazımı bayrağı

tapamalıdır. Bu funksiyada öz növbəsində yeni bir thread yaradır və sub_4010E0 funksiyasını çağırır. Funksiya ilk olaraq komanda sətirindən istifadəçidən 12 baytlıq bir məlumat daxil etməsini istəyir.

```
push    0FFFFFFF6h      ; nStdHandle
call    ds:GetStdHandle
mov     [ebp+hConsoleInput], eax
push    0               ; pInputControl
lea     ecx, [ebp+NumberOfCharsRead]
push    ecx             ; lpNumberOfCharsRead
push    0Ch             ; nNumberOfCharsToRead
lea     edx, [ebp+Buffer]
push    edx             ; lpBuffer
mov     eax, [ebp+hConsoleInput]
push    eax             ; hConsoleInput
call    ds:ReadConsoleA
```

Daha sonra daxil edilən məlumat (ascii kodlar) üzərində sadə riyazi əməliyyatlar aparır.

Əməliyyat 1:

```
mov     ecx, 1
shl     ecx, 0
movzx   edx, byte ptr [ebp+ecx+Buffer]
add     edx, 2
cmp     edx, 7Ah
```

Burada daxil edilən məlumatın 2-ci (index 1) xarakterinin üzərinə 2 əlavə edir və nəticənin 0x7A olub olmadığını test edir. Bayrağın 1. indexində hansı xarakterin olduğunu öyrənmək üçün 0x7A(z) dəyərindən 2-ni çıxaraq 2-ci xarakterin nə olduğunu öyrənmək olar. 0x7A – 2 = ‘x’

flag[1] = ‘x’

Əməliyyat 2

```
mov     eax, 1
imul    ecx, eax, 3
movzx   edx, byte ptr [ebp+ecx+Buffer]
add     edx, 13h
cmp     edx, 78h
```

Eyni şəkildə daxil edilən məlumatın 3-cü indexini yoxlayır.

Burada 3. index içərisində hansı ascii xarakterin olduğunu öyrənmək üçün ‘x’ (0x78) dəyərindən 0x13 (19) çıxılır.

0x78 – 19 = 101 (e).

İştirakçı bu şəkildə bütün hesablamaları aparıb sonda bayrağı əldə edə bilər.

expelliarmus : 99f2a77ff9c6967e8304f6c635a3ed67