

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.

```
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
```

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

```
>>> data = [97, 68,  
77,120,77,84,66,102,86,122,66,121,98,69,82,65,81,122,78,121,100,67,53,110,77,72,89,117,89,8  
8,111,61]
```

```
>>>
```

```
>>> ba = bytearray(data)
```

```
>>> ba
```

```
bytearray(b'aDMxMTBfVzBybERAQzNydC5nMHYuYXo=')
```

```
>>> import base64
```

```
>>> base64.b64decode(ba)
```

```
h3110\_W0rID@C3rt.g0v.az : 027cde3cb5ea8935e84ee82728d412d0
```


-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 dəyişəninə 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

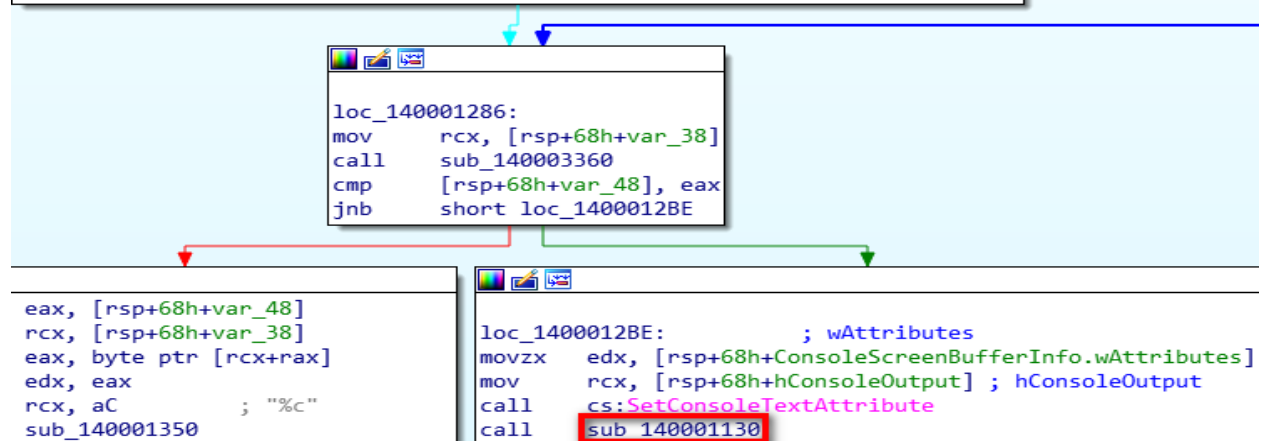
Programı komanda xətti (command line) üzərindən icra edərkən iştirakçılar aşağıda ki, mətn ilə qarşılaşacaqlar.

```
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_\(film\)](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.

```
lea    rax, aWelcomeToBridg ; "WELCOME TO BRIDGE OF SPIES Mr. James Do"...
mov    [rsp+68h+var_38], rax
mov    [rsp+68h+var_48], 0
lea    rax, [rsp+68h+ConsoleScreenBufferInfo]
mov    rdi, rax
xor    eax, eax
mov    ecx, 16h
rep    stosb
mov    ecx, 0FFFFFFF5h ; nStdHandle
call   cs:GetStdHandle
mov    [rsp+68h+hConsoleOutput], rax
lea    rdx, [rsp+68h+ConsoleScreenBufferInfo] ; lpConsoleScreenBufferInfo
mov    rcx, [rsp+68h+hConsoleOutput] ; hConsoleOutput
call   cs:GetConsoleScreenBufferInfo
mov    dx, 840Ch ; wAttributes
mov    rcx, [rsp+68h+hConsoleOutput] ; hConsoleOutput
call   cs:SetConsoleTextAttribute
mov    [rsp+68h+var_48], 0
jmp    short loc_140001286
```



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

```

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], 0FFFFFFFFFFFFFFFh
jnz     short loc_1400010CB
jmp     short loc_14000110B

```

```

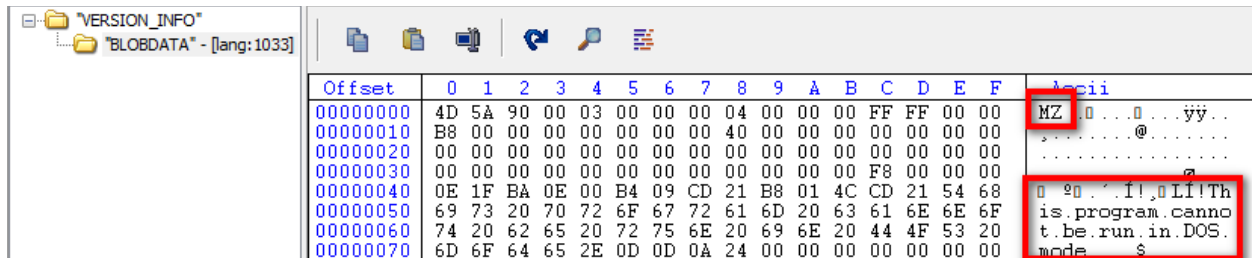
; CODE XREF: sub_140001000+C7↑j
mov     qword ptr [rsp+78h+dwCreationDisposition], 0 ; lpOverlapped
lea     r9, [rsp+78h+NumberOfBytesWritten] ; lpNumberOfBytesWritten
mov     r8d, 14h          ; nNumberOfBytesToWrite
lea     rdx, [rsp+78h+Buffer] ; lpBuffer
mov     rcx, [rsp+78h+hFile] ; hFile
call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



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	.
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	@
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F8	00	00	00	.
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	!
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program cannot
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode

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 obyektini 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
00000000E1AFEFDC0	74 68 40 6E 6B 5F 79 30 75 5F 6D 72 5F 64 30 6E	th@nk_y0u_mr_d0n
00000000E1AFEFDD0	30 76 40 6E 00 00 00 00 00 00 00 00 00 00 00	0v@n.....

[th@nk_y0u_mr_d0n0v@n](https://github.com/0x00sec/prosol) : 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: 00000000000000008
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.

```
000007EE02FF868 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]

əmaliyyətdir.

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. Məsələn tam olaraq anlamaq üçün GetProcAddress funksiyasına breakpoint təyin edib nələr olduğuna baxaq. Funksiya çağırılarkən parametrləri diqqət ilə analiz etsək burada kiçik bir problemin (anomaliya) olduğunu təyin edə bilərik.

```
CALL QWORD PTR DS:[<&GetProcAddress>]
MOV QWORD PTR SS:[RSP+0x38],RAX
CALL QWORD PTR SS:[RSP+0x38]
MOV QWORD PTR SS:[RSP+0x30],RAX
MOV RBD,0x13
LEA RDX,QWORD PTR DS:[0x7FF65D16D600]
MOV RCX,QWORD PTR SS:[RSP+0x30]
CALL cert.gov.az.7FF65D141E90
LEA RDX,QWORD PTR DS:[0x7FF65D16D63C]
MOV RCX,QWORD PTR SS:[RSP+0x28]
CALL QWORD PTR DS:[<&GetProcAddress>]
MOV QWORD PTR SS:[RSP+0x40],RAX
CALL QWORD PTR SS:[RSP+0x40]
XOR EAX,EAX
ADD RSP,0x58
RET
INT3
INT3
INT3
PUSH RBX
SUB RSP,0x20
MOV ECX,0x1
CALL cert.gov.az.7FF65D143184
CALL cert.gov.az.7FF65D1419AC
MOV ECX,EAX
CALL cert.gov.az.7FF65D143CD0
CALL cert.gov.az.7FF65D1419A0
MOV EBX,EAX
CALL cert.gov.az.7FF65D143E94
MOV FCX,0x1
```

00007FF65D16D63C: "Call0"

Register	Value	Comment
RAX	0000000000000000	
RBX	000001C582EC3110	&"C:\\Users\\user1\\Desktop\\CTF\\PROSOL\\t2
RCX	0000000000000000	
RDX	00007FF65D16D634	"Pool0"
RBP	0000000000000000	
RSP	0000003E65FFFA60	
RSI	0000000000000000	
RDI	000001C582EC8290	&"ALLUSERSPROFILE=C:\\ProgramData"
R8	00000000FFFFFFFF	
R9	0000000000000001	
R10	000001C582EC0000	
R11	0000003E65FF9950	
R12	0000000000000000	
R13	0000000000000000	
R14	0000000000000000	
R15	0000000000000000	
RIP	00007FF65D1413E0	cert.gov.az.00007FF65D1413E0

RFLAGS 0000000000000304
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0

Default (x64 fastcall) 5

```
1: rcx 0000000000000000 0000000000000000
2: rdx 00007FF65D16D634 cert.gov.az.00007FF65D16D634 "Pool0"
3: r8 00000000FFFFFFFF 00000000FFFFFFFF
4: r9 0000000000000001 0000000000000001
5: [rsp+20] 00007FF65D16D628 cert.gov.az.00007FF65D16D628 "prosol.dll"
```

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.

CALL QWORD PTR DS:[<&LoadLibraryA>]			Hide FPU
MOV QWORD PTR SS:[RSP+0x28],RAX			R14 0000000000000000
LEA RDX,QWORD PTR DS:[0x7FF65D16D634]	00007FF65D16D634:"Pool0"		R15 0000000000000000
MOV RCX,QWORD PTR SS:[RSP+0x28]			RIP 00007FF65D1413C9 cert.gov.az.00007FF65D1413C9
CALL QWORD PTR DS:[<&GetProcAddress>]			RFLAGS 000000000000304
MOV QWORD PTR SS:[RSP+0x38],RAX			ZF 0 PF 1 AF 0
CALL QWORD PTR SS:[RSP+0x38]			OF 0 SF 0 DF 0
MOV QWORD PTR SS:[RSP+0x30],RAX			CF 0 TF 1 IF 1
MOV R8D,0x13			LastError 00000000 (ERROR_SUCCESS)
LEA RDX,QWORD PTR DS:[0x7FF65D16D600]			LastStatus C0000034 (STATUS_OBJECT_NAME_NOT_FOUND)
MOV RCX,QWORD PTR SS:[RSP+0x30]			GS 002B FS 0053
CALL cert.gov.az.7FF65D141E90	00007FF65D16D63C:"Call0"		ES 002B DS 002B
LEA RDX,QWORD PTR DS:[0x7FF65D16D63C]			CS 0033 SS 002B
MOV RCX,QWORD PTR SS:[RSP+0x28]			ST(0) 0000000000000000 x87r0 Empty 0.0000000000000000
CALL QWORD PTR DS:[<&GetProcAddress>]			ST(1) 0000000000000000 x87r1 Empty 0.0000000000000000
MOV QWORD PTR SS:[RSP+0x40],RAX			ST(2) 0000000000000000 x87r2 Empty 0.0000000000000000
CALL QWORD PTR SS:[RSP+0x40]			ST(3) 0000000000000000 x87r3 Empty 0.0000000000000000
XOR EAX,EAX			ST(4) 0000000000000000 x87r4 Empty 0.0000000000000000
ADD RSP,0x58			ST(5) 0000000000000000 x87r5 Empty 0.0000000000000000
RET			
INT3			
INT3			
INT3			
PUSH RBX			
SUB RSP,0x20			
MOV ECX,0x1			
CALL cert.gov.az.7FF65D143184			
CALL cert.gov.az.7FF65D1419AC			
MOV ECX,EAX			
CALL cert.gov.az.7FF65D143C00			

=<kerne!32.LoadLibraryA>

Default (x64 fastcall)

- rcx 00007FF65D16D628 cert.gov.az.00007FF65D16D628 "prosol.dll"
- pdx 00007FF65D16D628 cert.gov.az.00007FF65D16D628 "prosol.dll"
- r8 0000021DEF6C8290 0000021DEF6C8290 &"ALLUSERSPROFILE=C:\ProgramData"
- r9 0000021DEF6C9470 0000021DEF6C9470
- [rsp+20] 00007FF65D16D628 cert.gov.az.00007FF65D16D628 "prosol.dll"

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.

FF15 51CC0000	CALL QWORD PTR DS:[<&LoadLibraryA>]		Hide FPU
48:894424 28	MOV QWORD PTR SS:[RSP+0x28],RAX		R14 0000000000000000
48:8D15 59C20200	LEA RDX,QWORD PTR DS:[0x7FF65D16D634]	00007FF65D16D634:"Pool0"	R15 0000000000000000
48:8B4C24 28	MOV RCX,QWORD PTR SS:[RSP+0x28]		RIP 00007FF65D1413CF cert.gov.az.00007FF65D1413CF
FF15 32CC0000	CALL QWORD PTR DS:[<&GetProcAddress>]		RFLAGS 000000000000204
48:894424 38	MOV QWORD PTR SS:[RSP+0x38],RAX		ZF 0 PF 1 AF 0
FF5424 38	CALL QWORD PTR SS:[RSP+0x38]		OF 0 SF 0 DF 0
48:894424 30	MOV QWORD PTR SS:[RSP+0x30],RAX		CF 0 TF 0 IF 1
41:80 13000000	MOV R8D,0x13		LastError 0000007E (ERROR_MOD_NOT_FOUND)
48:8D15 FFC10200	LEA RDX,QWORD PTR DS:[0x7FF65D16D600]		LastStatus C0000135 (STATUS_DLL_NOT_FOUND)
48:8B4C24 30	MOV RCX,QWORD PTR SS:[RSP+0x30]		
E8 850A0000	CALL cert.gov.az.7FF65D141E90	00007FF65D16D63C:"Call0"	
48:8D15 2AC20200	LEA RDX,QWORD PTR DS:[0x7FF65D16D63C]		
48:8B4C24 28	MOV RCX,QWORD PTR SS:[RSP+0x28]		
FF15 FBC00000	CALL QWORD PTR DS:[<&GetProcAddress>]		

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.

MOV QWORD PTR SS:[RSP+0x28],0x0		
MOV QWORD PTR SS:[RSP+0x30],0x0		
LEA RAX,QWORD PTR DS:[0x7FF65D16D628]	00007FF65D16D628:"qsptpm/emm"	
MOV QWORD PTR SS:[RSP+0x38],RAX		

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 geyrəndikdən sonra iştirakçılar prosol.dll faylını tapmalıdır. Bunun üçün iştirakçı proqram içerisində “qsptpm/emm” mətnini axtara bilər. Mətinlər bölməsində “qsptpm/emm” mətninə 2 yerdə referans göstərildiyini görürük.

```
.data:00007FF6... 0000000B C qsptpm/emm
.data:00007FF6... 0000000B C qsptpm/emm

aQsptpmEmm db 'qsptpm/emm',0 ; DATA XREF: sub_7FF65D141000+15↑o
align 8
aQsptpmEmm_0 db 'qsptpm/emm',0 ; DATA XREF: sub_7FF65D141230+16↑o
align 4
```

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

Funksiya var olmasına baxmayaraq proqram içerisində 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) yazılır.

```

call cs:CreateFileA
mov [rsp+68h+hFile], rax
cmp [rsp+68h+hFile], 0FFFFFFFFFFFFFFFh
jnz short loc_7FF65D1411D4

```

```

loc_7FF65D1411D4: ; lpOverlapped
mov qword ptr [rsp+68h+dwCreationDisposition], 0
lea r9, [rsp+68h+NumberOfBytesWritten] ; lpNumberOfBytesWritten
mov r8d, 16600h ; nNumberOfBytesToWrite
lea rdx, unk_7FF65D157000 ; lpBuffer
mov rcx, [rsp+68h+hFile] ; hFile
call cs:WriteFile ; Alignment : default
mov [rsp+68h+var_24], eax ; -----
cmp [rsp+68h+var_24], 0
jnz short loc_7FF65D141212

```

```

mp short loc_7FF65D14121D

```

```

mov rcx, [rsp+68h+hFile] ; hFile
call cs:CloseHandle
jmp short loc_7FF65D14121D

```

```

; Segment type: Pure data
; Segment permissions: Read/Write
data segment para public 'DATA'
assume cs: data
;org 7FF65D157000h
unk_7FF65D157000 db 4Dh ; M ;
db 5Ah ; Z ;
db 90h ; É ;
db 0 ;
dh 3 ;

```

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 mənimşədir və dll içərisində 7FFE4BDC32D8 funksiyası çağrılır. Funksiyanı analiz etsək RtlAllocateHeap funksiyasını çağırıldığını görə bilərik. DLL proqram yaddaşında 1024 baytlıq yer ayırır və yeni yaradılan yaddaş bölməsinin adresini RAX registeri üzərindən proqrama geri qaytarır.

Bundan sonra isə program

00007FF65D141406 E8 850A0000 CALL cert.gov.az.7FF65D141E90

əmrini 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 çağırıldıqdan sonra isə ekrana aşağıda ki, mesaj yazdırılır.

The screenshot shows a debugger window with assembly code on the left and a 'Congratulation' dialog box on the right. The assembly code includes instructions like MOV, CALL, LEA, XOR, ADD, RET, INT3, and TINT3. A comment next to the assembly code reads: 00007FF61BFC63C: "Call0". The 'Congratulation' dialog box has a title bar with 'Congratulation' and a close button (X). It contains an information icon (i) and the text '1t3rum t3nt@'. There is an 'OK' button at the bottom of the dialog box. Below the assembly code, there is a 'Locals' window showing a variable 'ASCII' with a value of '&v*..?%(\$\$.((.4>' and a memory dump window showing hex values like 000000901E0FF8C0, 000000901E0FF8C8, etc.

“**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 olmadığını yoxlanılır.

The screenshot shows assembly code in a window. The code is: lea rdx, [rsp+488h+nSize] ; nSize; lea rcx, [rsp+488h+Buffer] ; lpBuffer; call cs:GetComputerNameA; test eax, eax; jnz short loc_1800010BC. A red arrow points from the jnz instruction to a box containing the assembly code for loc_1800010BC: cmp [rsp+488h+nSize], 8; jz short loc_1800010E4.

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 bərabədirsə 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.

70	33	72	5F	40	73	70	65	72	61	5F	61	64	5F	61	73	p3r_@spera_ad_as
74	72	40	00	00	00	00	00	00	00	00	00	00	00	00	00	tr@.....

[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 tərəfindən 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_\(file_format\)](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
.  75 08 JNE damager.E21091
.  83C8 FF OR EAX,0xFFFFFFFF
.  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 mənimsəyib 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 orjinal məlumat ilə dəyişdirir. Bundan sonra yenidən 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** dəyərindən 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.

qr.zip			
0000	0000:	53 41 03 04 14 00 00 00 06 00 28 1A 2F 55 95 5A	SA..... ..(./UðZ
0000	0010:	25 91 E6 B0 00 00 00 99 01 00 06 00 00 00 71 72	%æµ...Öqr
0000	0020:	2E 70 6E 67 8A 7B 6D 3E F5 21 94 69 42 76 43 7E	.pngè{m> !öiBvC~
0000	0030:	CA 46 3E D3 C6 04 A9 EC EC 53 34 23 22 A7 EC 86	½F> .r∞ ∞S4#"∞å
0000	0040:	3F 4B 72 6F 46 74 DA 3C E7 20 ED 72 94 E5 C4 EE	?KroFt r< τ φröσ-ε
0000	0050:	92 F1 B0 9E 7D 93 88 3C 9D B4 D0 B0 90 0C 33 A2	±±R₀}ôê< ¥ ½É.3ó
0000	0060:	51 C6 04 27 4C E9 EC E6 3C 27 4D 4C 30 A3 79 24	Q .'L0∞µ <'ML0úy\$
0000	0070:	67 23 82 47 94 B9 AA 98 6F E9 B8 9D 1B 91 3B B4	g#éGö -ÿ oθγ¥.æ;
0000	0080:	D4 89 FB FB FF BB FB FB BF BF 22 CB 99 64 39 8D	↳èVV γVγ γγ "πÖd9i
0000	0090:	98 29 91 C1 4E AC 1C C8 B9 8A 31 4E AC FA A1 59	ÿ)æ¹N%. ll è1N%.îY
0000	00A0:	CF 63 F2 0A 51 3D CD C2 4E 41 0C 88 0A 89 E3 CD	↳cz.Q=→T NA.ê.ëπ=
0000	00B0:	10 B2 31 78 5C 52 58 09 AD 84 6D 91 F1 EA 4A 7E	.µ1x\RX. jãmæ±ΩJ~
0000	00C0:	9D BB B7 F2 5D EB 70 C9 19 8D 77 58 A2 18 2B D9	¥γ ≥ δprr .îwXó.+
0000	00D0:	1F EB 85 CE 46 7B B9 97 4F B9 B3 AE 3D 24 B1 CC	.δâ±F{ ü O «= \$
0000	00E0:	F0 6E DB FA E4 A9 8F 03 8B F6 39 19 1D 7A 69 FF	èñ.Σ-Á. ÿ÷9..zi
0000	00F0:	BE 7B 11 7B 47 BC 14 C2 B5 FA C4 14 B9 BA F8 64	↓{. {G .T .~. °d
0000	0100:	FF CF C3 69 D6 07 07 0E F1 EE C4 CC C1 F8 FD 6F	↳i r... ±ε- °z0
qr-original.zip			
0000	0000:	50 48 03 04 14 00 00 00 08 00 28 1A 2F 55 D6 1F	PK..... ..(./U r.
0000	0010:	77 C5 4A B1 00 00 00 99 01 00 06 00 00 00 71 72	w†J ...Öqr
0000	0020:	2E 70 6E 67 EC 1D 0B 58 93 47 F2 0F 24 10 25 18	.png∞..X ôGz\$.%.
0000	0030:	AC 20 58 B5 A0 62 CF 8A 8A 35 52 45 44 C1 8A E0	% X†áb±è è5RED¹èα
0000	0040:	59 2D 14 09 20 12 BC 5A 81 46 8B 14 F2 83 A2 88	Y-.. . Z üFÿ.≥âóè
0000	0050:	F4 97 D6 F8 1B F5 EE 5A FB D2 B6 D6 F6 6A 55 C4	[ü °. εZ vπ r†jU-
0000	0060:	37 A0 62 41 2A 8F 8A 80 5A 41 2B 2A 56 C5 1F 42	7ábA*Áèç ZA+*V†.B
0000	0070:	01 45 E4 21 F2 DF CC FE 09 8F DE FB 7D F7 5D D2	.EΣ!≥ . Å V ≈ π
0000	0080:	B2 EF 9D 9D 99 DD 9D 9D D9 D9 44 AD FF 02 5F EB	n¥¥Ö ¥¥ J†Dj . δ
0000	0090:	FE 4F F7 A7 28 CA 7A AE DF EC 57 28 CA 9C C7 3F	■0≈°(½z« ■∞w(½f ?
0000	00A0:	A9 05 94 6C 37 5B AB A4 28 27 6A EE 6C EF 85 AB	~.ö17[¥ñ ('jε1nà%
0000	00B0:	76 D4 57 1E 3A 34 3E 6F CB E2 0B F7 97 8C 2C 18	v½w.:4>o πΓ.≈ùî,.
0000	00C0:	FB DD D1 94 3B 8D 16 AF 7F EB 11 3E C4 7E 4D BF	√ πö;i.» ∆δ.>~Mγ
0000	00D0:	79 8D E3 A8 20 1D DF F1 29 DF D5 C8 5B 42 D7 AA	yipz .±) fll[B ~
0000	00E0:	96 08 BD 9C 82 CF E9 65 ED 90 5F 7F 7B 1C 0F 99	ü.½fé¹øe φÉ_∆f..Ö
0000	00F0:	D8 1D 77 1D 21 DA 72 A4 D3 9C A2 72 DF DC 9E 02	†.w.!rñr llÉör■Rₛ.
0000	0100:	99 A9 A5 0F B0 61 61 68 97 88 A2 AA A7 9E 9B 09	Ö-Ñ.∞aah ûêó~°Rₛ¢.

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 alətlərindənin 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.

The screenshot shows a hex editor interface. The main window displays hex data for a file named 'qr.png'. A red circle highlights the 'MZ' signature at address 6900h. The ASCII column shows the text 'is program cannot be run in DOS mode'. The right sidebar shows the 'Inspector' window with various data types and values. The bottom status bar shows 'Start: 26880 [6900h]'.

Address	Value
6900h	MZ
7562h	MZ
78B2h	MZ

```

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.

cmd.exe	2264
conhost.exe	9584
t4.exe	2924

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

-----

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

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

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



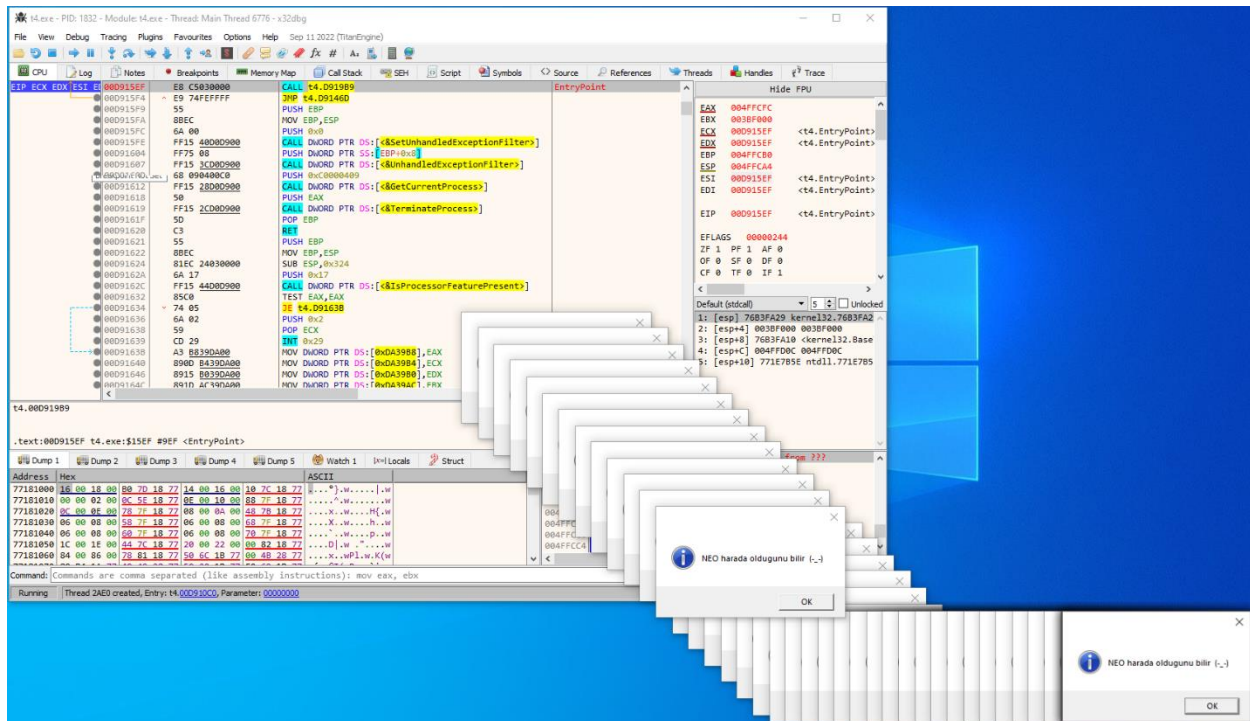
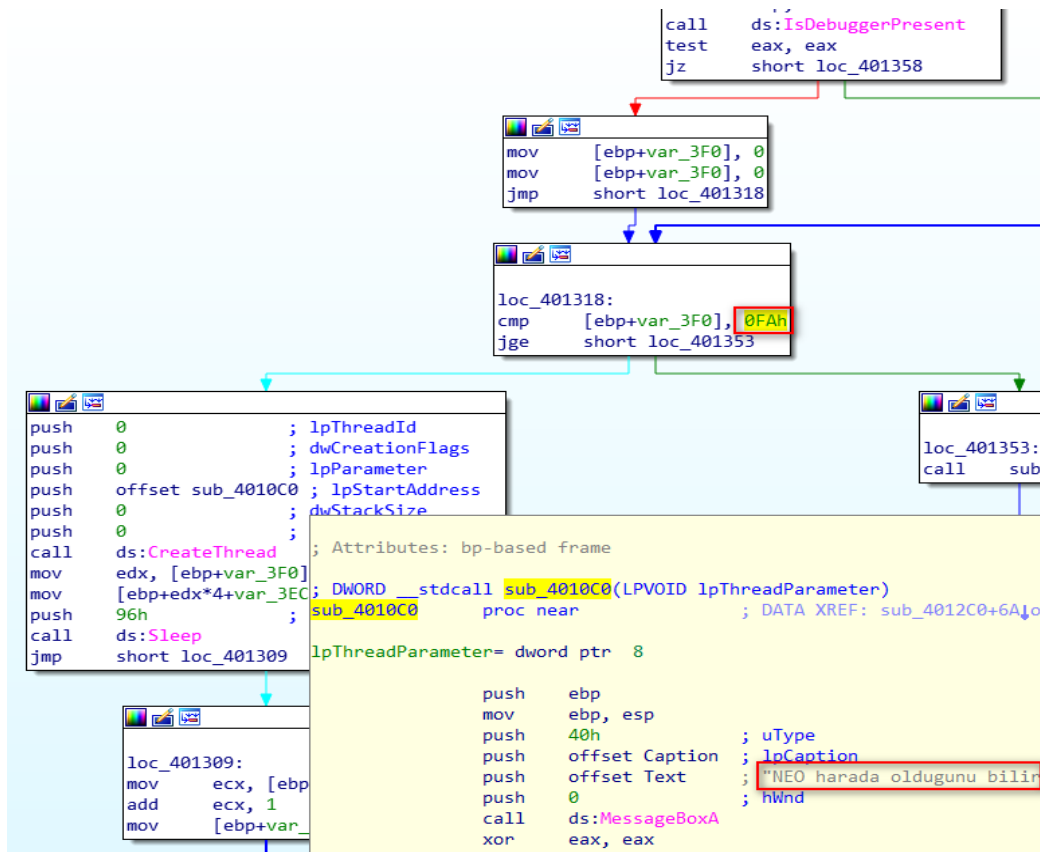
```
loc_401383:  
call sub_401060  
call sub_4012C0  
call sub_401280
```

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

```
push offset StartAddress ; lpStartAddress  
push 0 ; dwStackSize  
push 0 ; lpThreadParameter  
call ds:CreateThread ; Attributes: bp-based frame  
mov [ebp+hHandle], eax  
push 0FFFFFFFh ; dwFlags  
mov eax, [ebp+hHandle] ; StartAddress proc near ; DATA XREF: sub_  
push eax ; hThread  
call ds:WaitForSingleObject ; lpThreadParameter= dword ptr 8  
mov ecx, [ebp+hHandle]  
push ecx ; hThread  
call ds:CloseHandle  
mov esp, ebp  
pop ebp  
retn  
sub_401060 endp  
  
push ebp  
mov ebp, esp  
push 0 ; nCmdShow  
call ds:GetConsoleWindow  
push eax ; hWnd  
call ds:ShowWindow  
mov eax, 1  
pop ebp  
retn 4
```

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

Neo harada olduğunu bilir



Nəhayət sonuncu və iştirakçını bayrağa aparən funksiya - sub_401280. İştirakçı bu funksiya 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](https://expelliarmus.org/99f2a77ff9c6967e8304f6c635a3ed67) : 99f2a77ff9c6967e8304f6c635a3ed67