

Merhabalar ,

NOT: Bu çözüm yolu uzun yol olarak debug etme ve fonksiyonları inceleme ve araç kullanmayı gösterme amaçlıdır. Pratik yollar ile çözüme gidilebilir.

Rev50 linux çalıştırılabilir dosyasını çözmeye bakalım: Bu çözümü isterseniz sizde GDB aracını kullanarak deneyebilirsiniz. (Peda arayüzünü kullanıyorum. İsteyenler [burdan](#) yükleyebilir)

Öncelikle programımızı çalıştıralım ve istediği şeyleri yapalım ☺

```
root@kali: ~  
File Edit View Search Terminal Help  
root@kali:~# ./reverse1  
Enter the pincode : 1234  
Try harder :/root@kali:~#
```

Pincode kontrolünü incelemek için programın içine detaylıca bakalım:

```
root@kali:~# gdb ./reverse1  
GNU gdb (Debian 8.1-4+b1) 8.1  
Copyright (C) 2018 Free Software Founda  
License GPLv3+: GNU GPL version 3 or la  
This is free software: you are free to  
There is NO WARRANTY, to the extent per  
and "show warranty" for details.  
This GDB was configured as "x86_64-linu  
Type "show configuration" for configura  
For bug reporting instructions, please <  
<http://www.gnu.org/software/gdb/bugs/>  
Find the GDB manual and other documenta  
<http://www.gnu.org/software/gdb/docume  
For help, type "help".  
Type "apropos word" to search for comman  
Reading symbols from ./reverse1...(no d  
gdb-peda> b main  
Breakpoint 1 at 0x00484c9  
gdb-peda> run
```

Main fonksiyonuna breakpoint attıktan sonra “run” komutu ile programı çalıştırıyoruz. Programımız main fonksiyonunda durunca bulunduğu prosedürü disassemble ederek assembly kodlarına bakalım.

Peda kullanmıyorsanız disassemble için Intel ve AT&T yazım şekilleri okunma kolaylığı için size uygun olanını seçebilirsiniz. Peda intel yazımına çevirdiği için benimde kolayıma geliyor. GDB için intel yazımında okumak isterseniz:

“set disas-flavor intel”

Komutuyla isterseniz bu yazıma geçebilirsiniz.

Bu adımdan sonra main fonk. Kodlarını ,

“disas main” komutuyla açalım:

```
gdb-peda> disas main  
Dump of assembler code for function main:  
0x00484bb <+0>: lea ecx,[esp+0x4]  
0x00484bd <+2>: and esp,0xffffffff  
0x00484c2 <+7>: push DWORD PTR [ecx-0x4]  
0x00484c5 <+10>: push ebp  
0x00484c6 <+11>: mov ebp,esp  
0x00484c8 <+13>: push ecx  
0x00484c9 <+14>: sub esp,0x4  
0x00484cc <+17>: sub esp,0xc  
0x00484cf <+20>: push 0x00485b9  
0x00484d4 <+25>: call 0x0048330 <printf@plt>  
0x00484d9 <+30>: add esp,0x10  
0x00484dc <+33>: sub esp,0x8  
0x00484df <+36>: push 0x004a028  
0x00484e4 <+41>: push 0x00485ce  
0x00484e9 <+46>: call 0x0048350 <__isoc99_scanf@plt>  
0x00484ee <+51>: add esp,0x10  
0x00484f1 <+54>: call 0x004846b <xyz>  
0x00484f6 <+59>: mov ds:0x004a024,eax  
0x00484fb <+64>: mov eax,ds:0x004a028  
0x0048500 <+69>: sub esp,0xc  
0x0048503 <+72>: push eax  
0x0048504 <+73>: call 0x0048487 <zxy>  
0x0048509 <+78>: add esp,0x10  
0x004850c <+81>: mov eax,0x0  
0x0048511 <+86>: mov ecx,DWORD PTR [ebp-0x4]  
0x0048514 <+89>: leave  
0x0048515 <+90>: lea esp,[ecx-0x4]  
0x0048518 <+93>: ret  
End of assembler dump.
```

Burada scanf fonksiyonundan hemen sonra xyz prosedürüne git-

tiğini görüyoruz. O zaman bu adrese gidip bakalım:

Bunun için “b *0x00484F1” adresine bu şekilde bir breakpoint atalım.

Şuana kadar 2 bp koyduk. “continue” veya “c” yazarak programı başlatalım

Çalıştırdığım zaman bizden pincode istemekte:

```
Breakpoint 1, 0x080484c9 in main ()
gdb-peda> c
Continuing.
Enter the pincode : 1234
```

Girdinin hemen ardından xyz() fonk. 'da durdu; Şimdi bu fonksiyonun içine "si" ya da "step 1" komutuyla girelim. Ve girmiş olduğumuz fonk. Disassemble edelim "disas":

```
0x0804846b in xyz ()
gdb-peda> disas
Dump of assembler code for function xyz:
=> 0x0804846b <+0>:    push    ebp
0x0804846c <+1>:    mov     ebp,esp
0x0804846e <+3>:    sub     esp,0x10
0x08048471 <+6>:    mov     DWORD PTR [ebp-0x4],0xf4240
0x08048478 <+13>:   add     DWORD PTR [ebp-0x4],0x29a
0x0804847f <+20>:   shl     DWORD PTR [ebp-0x4],1
0x08048482 <+23>:   mov     eax,DWORD PTR [ebp-0x4]
0x08048485 <+26>:   leave  eax,DWORD PTR [ebp-0x4]
0x08048486 <+27>:   ret
End of assembler dump.
```

Burada yapılan işlemi şöyle yazmak isterek:

Ebp-0x4= (0xf4240+0x29a) , son olarakta bu sonucu bir shif-left yaparak EAX değerine atıyor.

Biz bu işlemlerin direkt sonucuna bakalım; bunun için leave komutuna bp atarak prosedürden return edilen EAX değerinin son halini öğrenmiş olalım. Anladığımız kadarıyla burdan return edilen EAX değeri bizim girdimizle karşılaştırılacak olan doğru PINCODE değeri.

```
gdb-peda> disas
Dump of assembler code for function xyz:
=> 0x0804846b <+0>:    push    ebp
0x0804846c <+1>:    mov     ebp,esp
0x0804846e <+3>:    sub     esp,0x10
0x08048471 <+6>:    mov     DWORD PTR [ebp-0x4],0xf4240
0x08048478 <+13>:   add     DWORD PTR [ebp-0x4],0x29a
0x0804847f <+20>:   shl     DWORD PTR [ebp-0x4],1
0x08048482 <+23>:   mov     eax,DWORD PTR [ebp-0x4]
=> 0x08048485 <+26>:   leave  eax,DWORD PTR [ebp-0x4]
0x08048486 <+27>:   ret
End of assembler dump.
gdb-peda> display $eax
2: $eax = 0x1e89b4
```

Eax decimal = **2001332** olarak çıkıyor.

Şimdi Main fonksiyonumuzda bir sonraki call zyx prosedürü çağrılmakta. Sanırım bu fonksiyonda Eax'in değeri ve Girdimizin karşılaştırıldığı fonksiyon olması lazım eğer iç içe girmiş prosedürlerle karmaşılaştırılmadıysa. Main disas. haline yukarı çıkarak tekrar bakabilirsiniz.

Sarı çizilmiş yerlere dikkatlice bakarsanız. ZXY fonksiyonunu çağırmadan hemen önce EAX değerini(PIN CODE) depoluyor. Ve girdimizi EAX üzerine yazarak argument olacak şekilde ZXY(girdi) yolluyor.

Zxy adresine bp atalım .

"b *0x8048487" sonrasında **"c"** ile devam ederek bp'anında **"si"** ile fonk. İçerisine girip **"disas"** yapalım.

```
0x08048487 in zxy ()
1: $eax = 0x4d2
2: $eax = 0x4d2
gdb-peda> disas
Dump of assembler code for function zxy:
=> 0x08048487 <+0>:    push    ebp
0x08048488 <+1>:    mov     ebp,esp
0x0804848a <+3>:    sub     esp,0x8
0x0804848d <+6>:    cmp     DWORD PTR [ebp+0x8],0x1e89b4
0x08048494 <+13>:   je      0x080484a8 <zxy+33>
0x08048496 <+15>:   sub     esp,0xc
0x08048499 <+18>:   push    0x80485a0
0x0804849e <+23>:   call    0x08048330 <printf@plt>
0x080484a3 <+28>:   add     esp,0x10
0x080484a6 <+31>:   jmp     0x080484b8 <zxy+49>
0x080484a8 <+33>:   sub     esp,0xc
0x080484ab <+36>:   push    0x80485ae
0x080484b0 <+41>:   call    0x08048330 <printf@plt>
0x080484b5 <+46>:   add     esp,0x10
0x080484b8 <+49>:   nop
0x080484b9 <+50>:   leave
0x080484ba <+51>:   ret
End of assembler dump.
```

Bakın ☺ 0x1e89b4 değeri ile input'umuz cmp edilmiş , yani bir eşitlik kontrolü var;

Eğer eşitse zxy+33 adresine dallanarak "Yuppi" datasının bulunduğu adresi printf fonk. için hazırlayıp yazdırıyor.