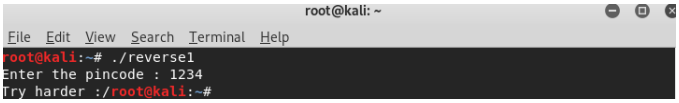
Merhabalar ,

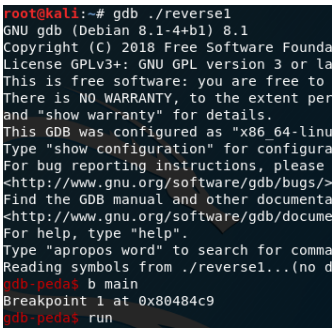
NOT: *Bu çözüm yolu uzun yol olarak debug etme ve fonksiyonları inceleme ve araç kullanmayı gösterme amaçlıdır. Pratik yollar ile çözüme gidilebilir.*

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 ☺



Pincode kontrolünü incelemek için programın içine detaylıca bakalım:



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

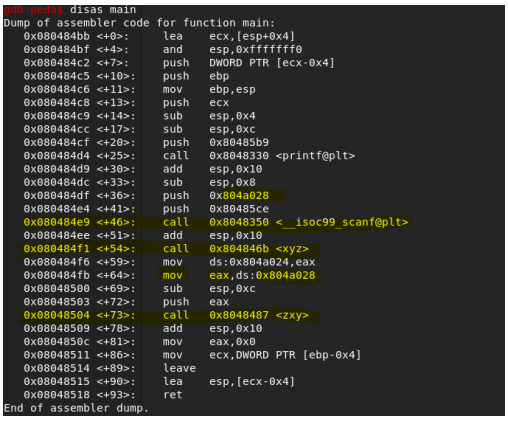
Peda kullanmıyorsanız disassemble için Intel ve AT&T yazım şekilleri okunma kolaylığı için size uygun olanını seçebilirsiniz. Peda intel yazımına çevirdiği için benimde kolayıma geliyor. GDB için intel yazımında okumak isterseniz:

"set disas-flavor intel"

Komutuyla isterseniz bu yazıma geçebilirsiniz.

Bu adımdan sonra main fonk. Kodlarını ,

"**disas main**" komutuyla açalım:



Burada scanf fonksiyonundan hemen sonra xyz prosedürüne git-tiğini görüyoruz. O zaman bu adrese gidip bakalım:

*Bunun için "b *0x080484F1" 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
   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
   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 karmaşıklaştırılmadıysa. 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     0x80484a8 <zxy+33>
   0x08048496 <+15>:    sub    esp,0xc
   0x08048499 <+18>:    push   0x80485a0
   0x0804849e <+23>:    call   0x8048330 <printf@plt>
   0x080484a3 <+28>:    add    esp,0x10
   0x080484a6 <+31>:    jmp    0x80484b8 <zxy+49>
   0x080484a8 <+33>:    sub    esp,0xc
   0x080484ab <+36>:    push   0x80485ae
   0x080484b0 <+41>:    call   0x8048330 <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. İçin hazırlayıp yazdırıyor.