Merhabalar,

NOT: GDB kullanımı ile ilgili komutların her detayına inenemiş olabilirim. Bu yüzden GDB Cheat sheet'e bakarak takip edebilirsiniz , takıldığınız yerleri. GDB Cheat Sheet bu adrestem bakabilirsiniz.

Reverse2 sorusunun çözümü için GDB aracını kullanarak ilerlemeye başlayalım.

Programı çalıştırıyoruz:

```
root@kali:~/pwn# ./reverse2
Enter the my hash :1234
Try harder !
root@kali:~/pwn#
```

Bir önceki writeup'dan yola çıkarak tekrar Gdb aracı ile programın içine girerek incelemeye başlayalım:

```
root@kali:~/pwn# gdb -q ./reverse2
Reading symbols from ./reverse2...(no debugging symbols found)...done.
gdb-peda$ b main
Breakpoint 1 at 0x80485c0
gdb-peda$ run
```

Main fonksiyonundaki assembly kodlarını görmek için "**disas-main**" kodunu çalıştırtıktan sonra kodlara dikkatlice bakarak yapılan işi anlamaya başlayalım:

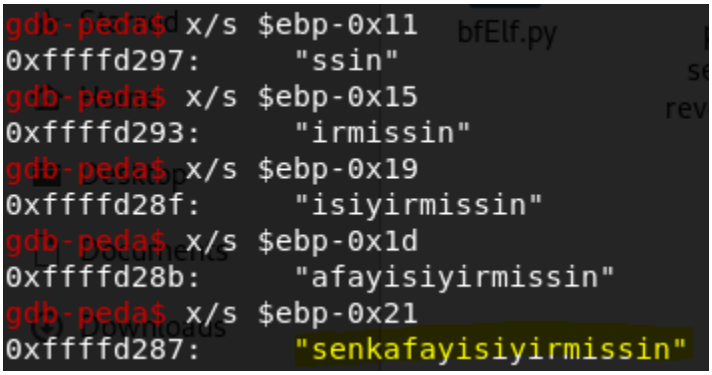
```
0x080485e1 <+47>:    lea    eax,[ebp-0x2c]
0x080485e4 <+50>:    push   eax
0x080485e5 <+51>:    push   0x804871e
0x080485ea <+56>:    call   0x8048400 <__isoc99_scanf@plt>
0x080485ef <+61>:    add    esp,0x10
0x080485f2 <+64>:    mov    DWORD PTR [ebp-0x21],0x6b6e6573
0x080485f9 <+71>:    mov    DWORD PTR [ebp-0x1d],0x79616661
0x08048600 <+78>:    mov    DWORD PTR [ebp-0x19],0x79697369
0x08048607 <+85>:    mov    DWORD PTR [ebp-0x15],0x696d7269
0x0804860e <+92>:    mov    DWORD PTR [ebp-0x11],0x6e697373
0x08048615 <+99>:    mov    BYTE PTR [ebp-0xd],0x0
0x08048619 <+103>:   sub    esp,0xc
0x0804861c <+106>:   lea    eax,[ebp-0x21]
0x0804861f <+109>:   push   eax
0x08048620 <+110>:   call   0x80483e0 <strlen@plt>
0x08048625 <+115>:   add    esp,0x10
0x08048628 <+118>:   sub    esp,0x8
0x0804862b <+121>:   push   eax
0x0804862c <+122>:   lea    eax,[ebp-0x21]
0x0804862f <+125>:   push   eax
0x08048630 <+126>:   call   0x804851b <hash_hesapla>
0x08048635 <+131>:   add    esp,0x10
0x08048638 <+134>:   mov    DWORD PTR [ebp-0x28],eax
0x0804863b <+137>:   mov    eax,DWORD PTR [ebp-0x2c]
0x0804863e <+140>:   sub    esp,0x8
0x08048641 <+143>:   push   DWORD PTR [ebp-0x28]
0x08048644 <+146>:   push   eax
0x08048645 <+147>:   call   0x804857f <coqhoj>
```

Scanf ile bizden girdi aldıktan sonra Hard-coded şekilde ascii karakterlerinin hex halinin stack'e yazıldığını görebilirsiniz. İsterseniz hex değerleri dönüştürüp de bulabilirsiniz. Ama artık debug aracını biraz kullanabildiğimize göre bu String'in son halini kendimiz keşfedelim.

Bunun için main+99 adresinde, yani DWORD Pointerler'a taşınma işlemlerinin bittiği adıma geçiyoruz.(

```
End of assembler dump.
gdb-peda$ b *0x08048615
Breakpoint 2 at 0x8048615
gdb-peda$ c
Continuing.
Enter the my hash :12345
```

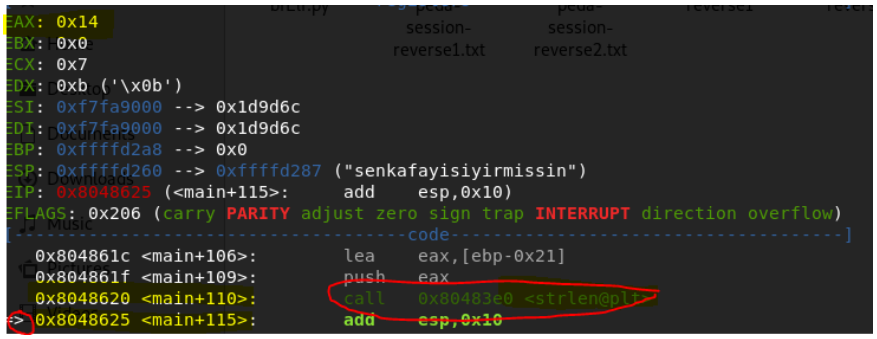
Bu adımda bizi tam istediğimiz yerde durdurdu ve bizde stackte nasıl bir String girilmiş hemen “examine” stack’e sorgulama işlemine başladık.



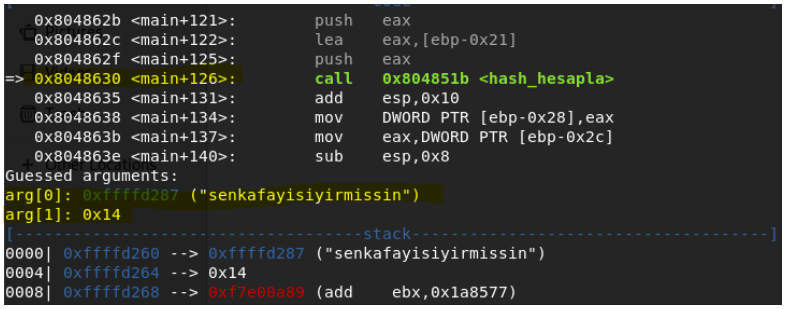
Bu adresleri yukardaki resimleri incelerseniz neye göre “examine” edildiğini anlamanız rahat olacaktır.

“**x/s**“ x = examine , s=String değer okuma belirteçleridir.

Hemen ardından gelen call fonksiyonu strlen yani eax’in tuttuğu adresteki verinin uzunluğunu hesaplayacak. Bunuda hemen strlen altına bp atarak kontrol edelim:



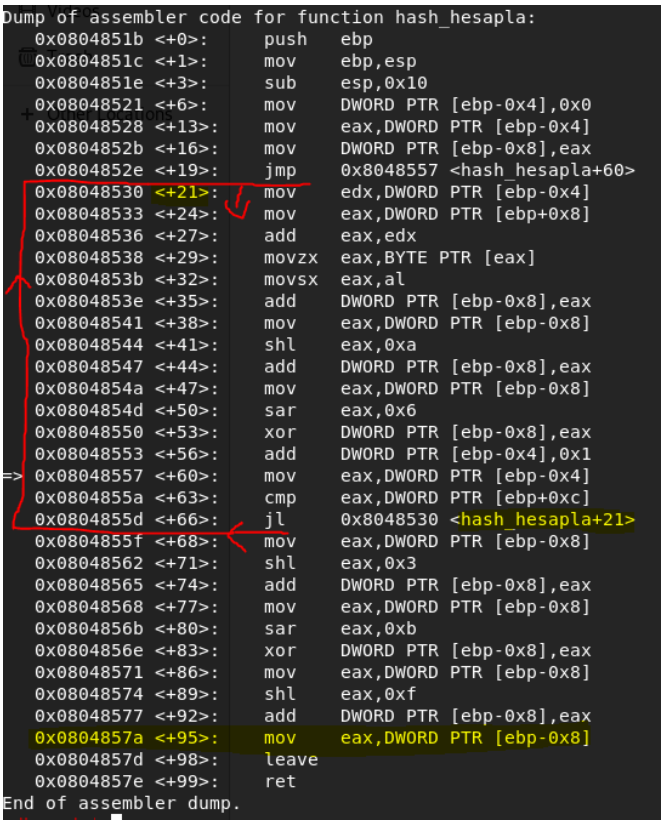
0x14= 20 karakter uzunluğunda ! doğru. Pedanın Güzelliğinden faydalanarak bir sonraki fonks. olan hash_hesapla ‘ya bakalım.



Burada tahmini olarakda olsa çok güzel yardımı dokunuyor PEDA’nın ☺ stringimizi ve uzunluğunu parametre olarak alıyor.

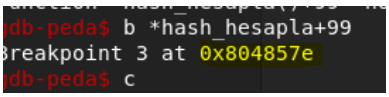
Şimdi asıl olay olan hash_hesapla’da neler dönüyor? ☺ :

Fonksiyon bizden aldığı Girdi değerini sadece karşılaştırma olarak en son kullandığı için aslında çok kafa karıştıracak bir şey yokta diyebiliriz. Sadece kendi Stringi ve onun uzunluğuyla sabit işlemler döndürüyor. Hash döngüsünün neler yapıtğını inceleyebilirsiniz adım adım. Ben ise burda artık daha fazla kafa ağrıtmadan döngü bitiminde karşılaştırma için ürettiği hash değerini yakalamaya çalışayım:

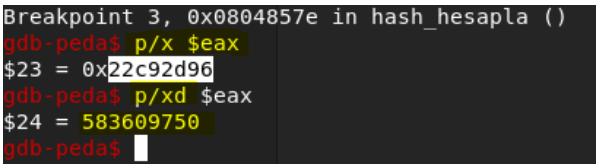


Kırmızı ile çizdiğim alanda hash üretme işlemleri “senkafayisiyirmissin” string için çeşitli işlemlerden geçerek üretilmekte. İsterseniz ne işlemler döndüğünü gün yüzüne çıkartabilirsiniz ama şimdilik ben bu adımı geçiyorum.

Son olarak +95 adımına bp atacağım ve eax değerine atılan son değere bakalım , belki bizim merak ettiğimiz hash bu olabilir ☺

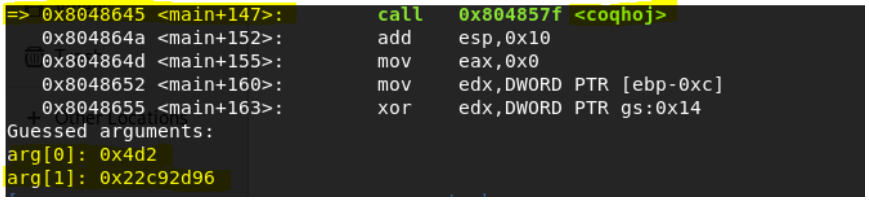


“c” ile devam ederek “p/x $eax” ‘ile return edilen değere bakalım.



Son olarak decimal olarak çıktı aldık ve bunu unutmadan artık bitirelim bu işi ☺

Son Fonksiyon olan “coqhoj” a doğru giderken gönderilen parametrelere bakalım ve şimşekler çaksın ^^



Bundan sonrasını size bırakıyorum , iyi eğlenceler ☺