**PWN**

**Eksta kaynaklar:**

**Nightmare: https://guyinatuxedo.github.io/**

**https://sec-fortress.github.io/posts/thm/posts/pwn101.html**

**https://7rocky.github.io/en/ctf/htb-challenges/pwn/**

**Kurulumlar**

GDB zaten var varsayılan olarak.

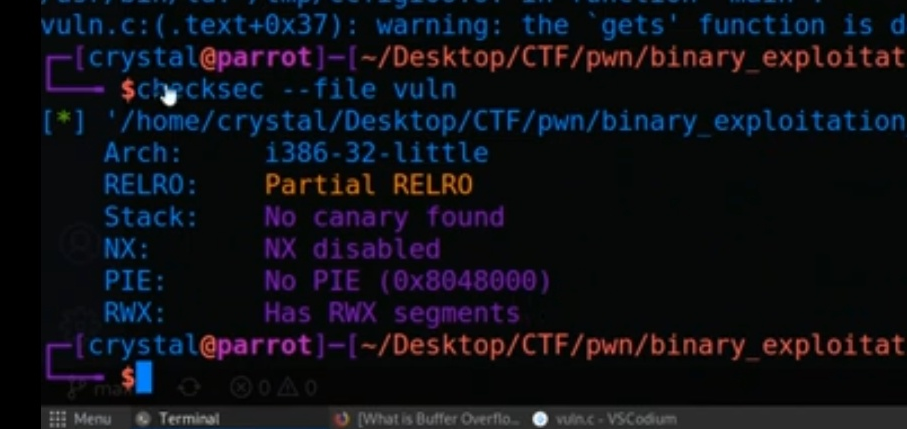
PWNDBG KUR!

Python scriptleri yazabilmek için:

pip install pwn

pip3 install pwntools

**checksec:**



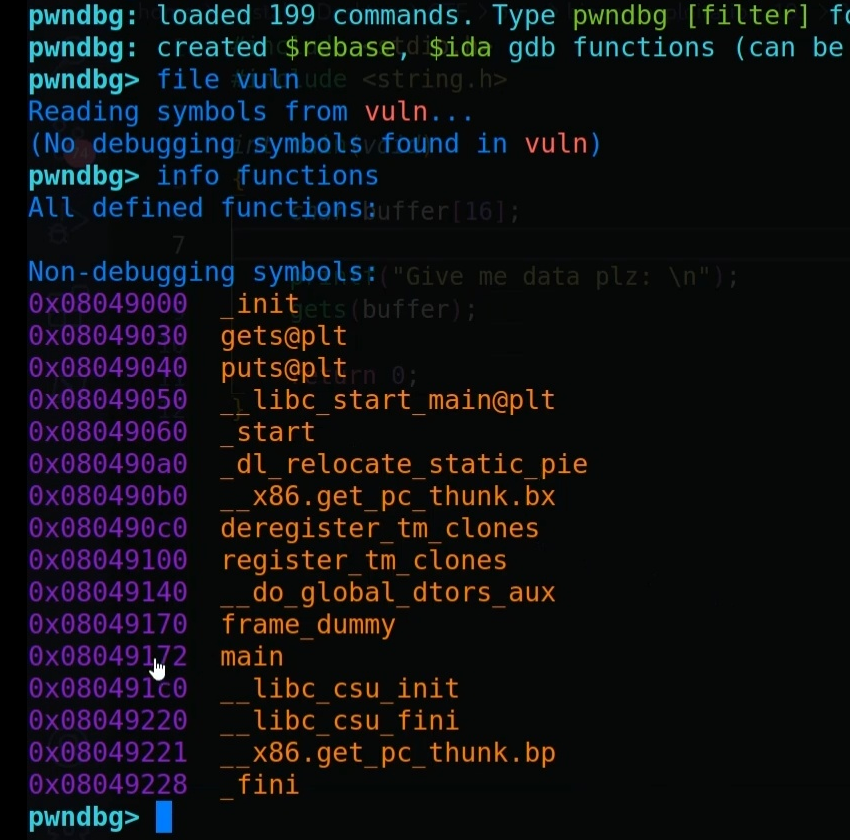
* Canary (Kanarya): Canary, bir programın bellek taşmasına (buffer overflow) karşı korunmasına yardımcı olan bir güvenlik mekanizmasıdır. Bir işlevin başına eklenen rastgele bir değeri içerir ve bu değer, işlevin sonunda kontrol edilir. Eğer no Canary found yazıyorsa, bir buffer overflow saldırısı olabilir.
* PIE (Position Independent Executable): PIE, yürütülebilir dosyaların (executable) rastgele bir bellek konumunda çalıştırılabilmesini sağlayan bir özelliktir. Bu, saldırganların daha zor hedefler seçmesini gerektirir çünkü dosyanın bellek konumu
* önceden tahmin edilemez.
* NX (No Execute): Bu, veri bölgesinin (data segment) yürütülebilir olmadığını belirtir. Bu, saldırganların veri bölgesini kötü amaçlı kod yürütmek için kullanmasını engeller.
* ASLR (Address Space Layout Randomization): ASLR, işletim sistemi tarafından belirli bileşenlerin bellek konumlarının rastgele olarak değiştirilmesini sağlayan bir güvenlik önlemidir. Bu, saldırganların programların bellek yapısını tahmin etmelerini zorlaştırır.
* Stripped(file ile bakılır): fonksiyon adları ghidrada çıkmaz. Fun3737379282 yazar.

**\*\*\*\*\*\*\*\*PWNDBG\*\*\*\*\*\*\*\***

₺gdb-pwndb

<pwndb>₺file hacme //hedef dosya seçilir

<pwndb>₺info functions //fonksiyonları listele



<pwndb>₺disassemble main //main fonksiyonun assemble kodlarını gösterir

<pwndb>₺break main //main e breakpoint koy

<pwndb>delete breakpoints //breakpointleri kaldır

<pwndb>₺run //programı çalıştır. Önceden breakpoint koyulmuşsa orada bekler.

<pwndb>n //breakpoint varken adım adım

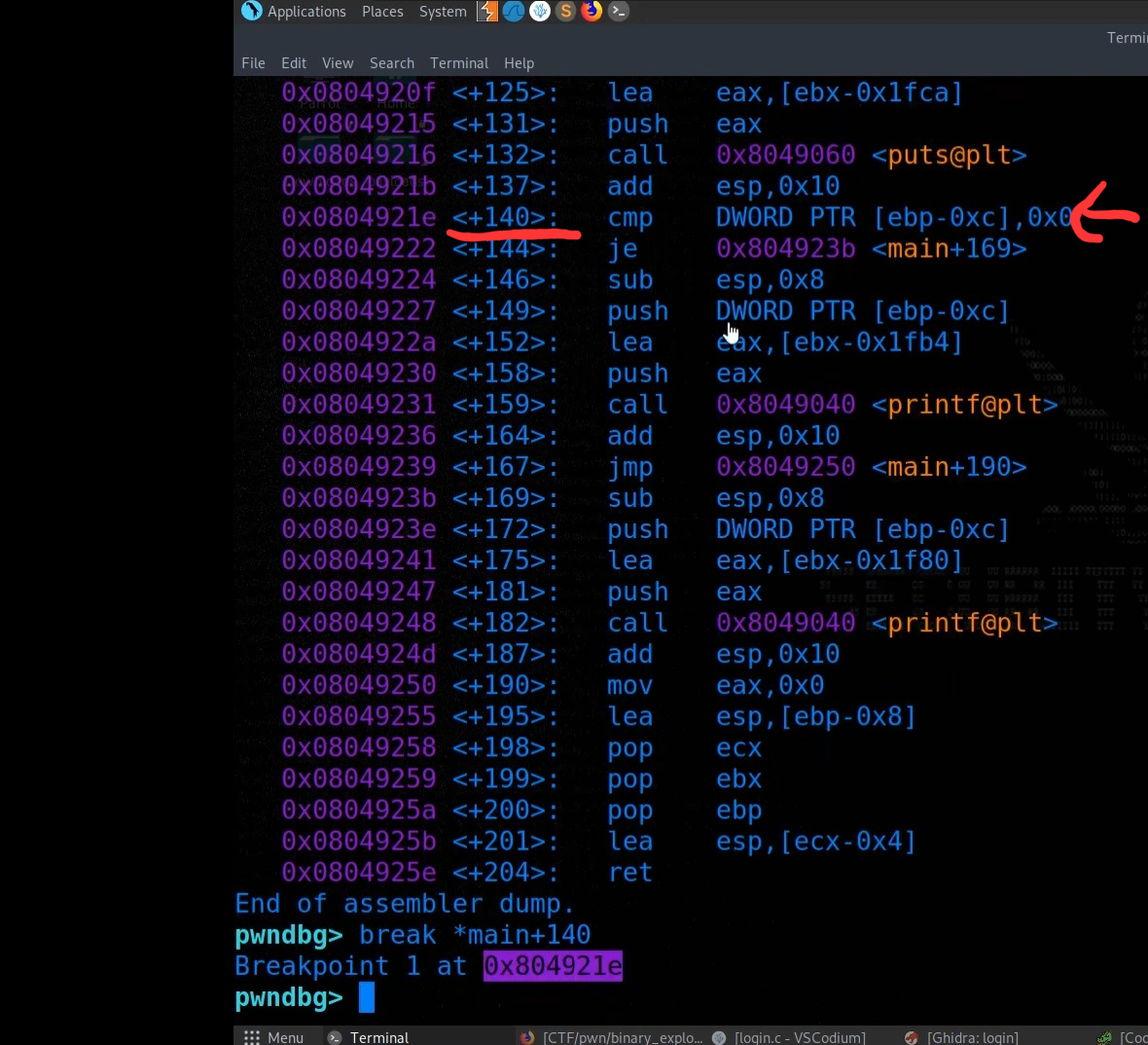
çalıştırmak için. Next

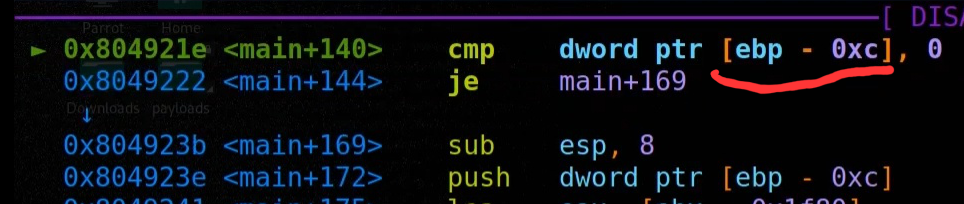
<pwndb>c //breakpoint varken adım adım çalıştırmayı sonlandırıp devam et komple çalıştır demek.

<pwndb>x $eax //registerin içindekini ekrana yaz

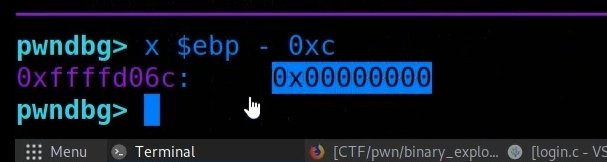
**Örnek1(overwrite stack variable):** Karşılaştırmanın olduğu adresi değiştirme:

<pwndb>break \*main+140 //özel bir koda breakpoint koy

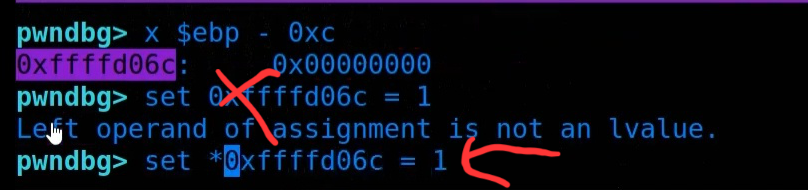




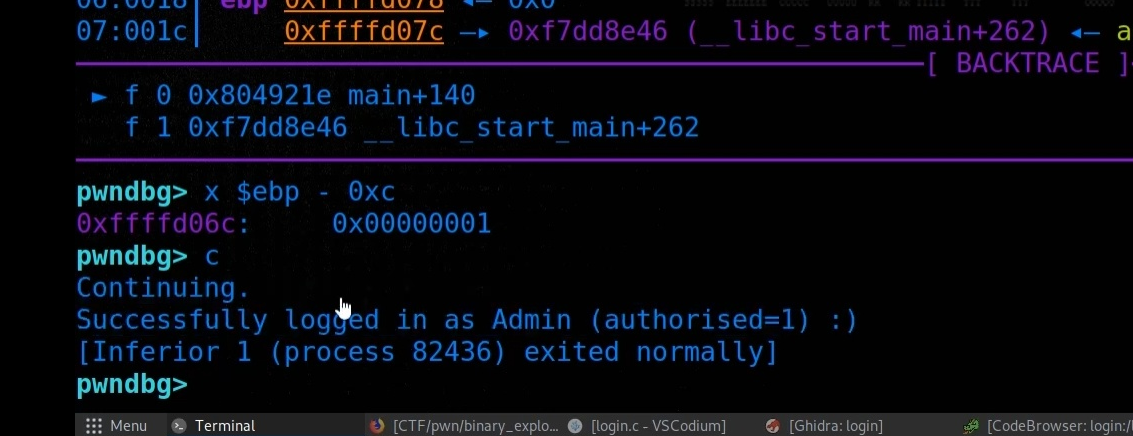
İşaretli satırda ebp-0xc'den gelen değer 0 a eşitmi diye kontrol ediyor.



ebp-0xc'den gelen değeri ekrana yazdırdık sıfırmış.



<pwndb>set \*0xadres =1 //ile adrsteki 0 ı 1 yaptık. Adresi bir önceki resimde görebilirsin x ile yazdırmıştık.



Devam ettiğimizde başarılı olduğumuzu gördük.

**Örnek2 (Overwrite stack variables):** Buffer owerflow

Kaynak kod:



32 karakterli buffer değişkenine dışarıdan 32 karakterden uzun bir değer girrilerek key değişkenine etki edilebilir.

Yol1: python2 ile overflow

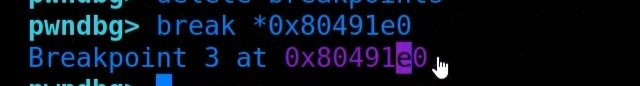


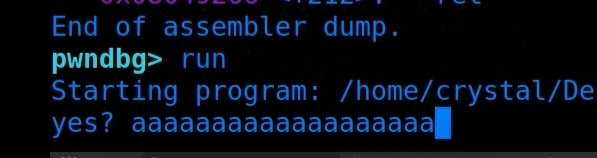


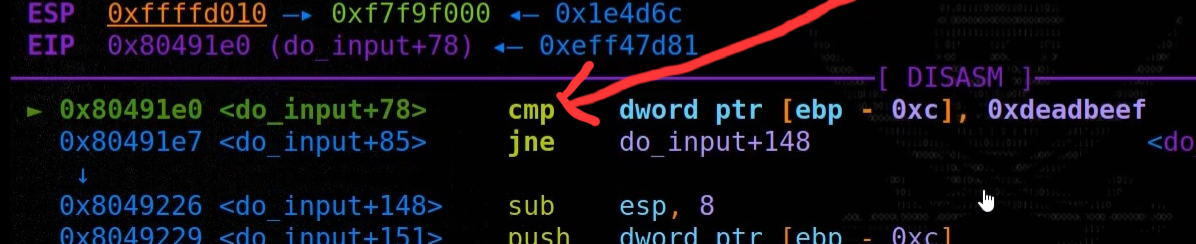
Yol2: adresteki değeri değiştirerek

<pwndbg>₺disassemble do\_input

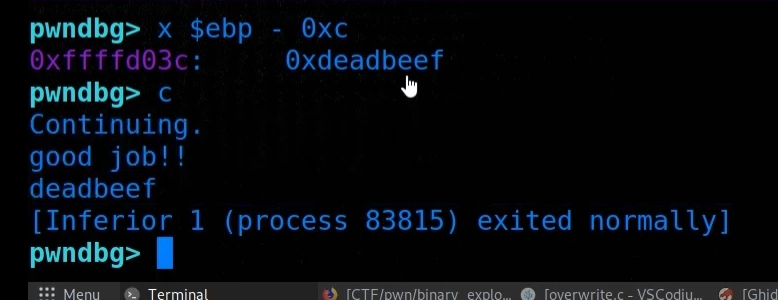












Yol3: python3 pwntool ile overflow

İşaretli yer eklenirse adım adım neler olduğunu göstererek çalışır.

**Örnek3 (RET2WİN)** Buffer owerflow ile kullanılmayan bir fonksiyonu çağırma.(checksec yaparsan no pie olduğunu görebilirsin. Adresler sabit depişmiyor)

Kaynak kod:

Yol1: python2 ile overflow

Ulaşmak istediğimiz fonksiyon hakced. Adresini kopyalayabiliriz. Lazım olacak.

Sonraki komutun adres bilgisini tutan EIP registerına overflow yapıp yukarıda kopyaladığımız adresi oraya eklemek istiyoruz. Bu sayede hacked fonksiyonu

çalıştırılacak.

Bunun gerçekleşmesi için EIP registerina kaçıncı bitte ulaşılıyor bulmamız lazım. Bunun için aşağıdaki işlemleri yap.

0x08049182 -> tersten ikili yazılır.

(Bunu farklı terminalde, elf dosyası ile aynı klasörde yap. Payload kullanılacak)

Yol2: python3 ile hazır script.

Github adresinde var.

**Örnek4 (RET2WIN WITH PARAMS 32bit)**Buffer owerflow ile kullanılmayan parametreli bir fonksiyonu çağırma.(checksec yaparsan no pie olduğunu görebilirsin. Adresler sabit depişmiyor)

Kaynak kod:

Yol1: python2 ile overflow

Hacked fonksiyonun başlangıç adresini kopyalıyoruz. Sonraki python kodunda EIP için kullanılacak.

Register\_name fonksiyonun son adresine yani ret komutuna breakpoint bırakıyoruz. Çünkü hacked fonksiyonunu adım adım çalıştırıp parametreleri göreceğiz.

(Bunu farklı terminalde, elf dosyası ile aynı klasörde yap. Payload kullanılacak)

Breakpoint koyduğumuz yerden itibaren başladı. Bir sonraki adım hacked fonksiyonuna geçecek.

n ile adım adım ilerleyerek ilk CMP koduna geliyoruz. Burada [ebp+8] adresindeki değer 0xdeadbeef 'e eşit mi diye kontrol ediyor. İkinci cmp kodunu ilkini doğru girmeden göremiyoruz o yüzden önce ilkini doğru girip sonrakini bulmaya çalışcaz.

[ebp+8] adresinde BBBB varmış.

Bizim payloaddaki BBBB yerine deadbeef ekliyoruz.

n ile adım adım ilerleyerek ikinci CMP koduna geliyoruz. Burada [ebp+0cx] adresindeki değer 0xc0debabe 'e eşit mi diye kontrol ediyor.

[ebp+0xc] adresinde CCCC varmış.

Bizim payloaddaki CCCC yerine c0debabe ekliyoruz.

İşlem tmm.

**Örnek4 (RET2WIN WITH PARAMS 64bit)**Buffer owerflow ile kullanılmayan parametreli bir fonksiyonu çağırma.(checksec yaparsan no pie olduğunu görebilirsin. Adresler sabit depişmiyor)

Kaynak kod:

64 bit olunca registerlar ve parametrelerin çekilme mantığı değişiyor. Eax EIP gibi registerlar artık RAX ve RIP oluyor. Registerlar 32 bitteyken 8 bit uzunluktayken 64 bitteyken 16 bit oluyor.

Burada başlangıç noktamız RIP'te yazmıyor. RSP'nin ilk 4 bitinden alınması gerek.

24 bitlik padding yapılcak

Sonrasını aşağıdaki payload gibi yapacağız. Burada pop\_rdi ve pop\_rsi ları sonraki adımlarda bulup oraya ekleyelim.

32bitteyken bu poplar yoktu. 64 bitte parametreler pop ile çekiliyormuş. İlk 2

parametrede rdi ve rsi kullanılıyor. Daha fazla parametre olduğunda

RDI, RSI, RDX, RCX, R8 ve R9 gibi kaydedicilere parametreler yığından çıkarılır. 6dan fazla parametre olduğunda, yığına yerleştirme işlemi yapılır ve pop talimatları bu kaydedicilere uygulanır.

Payload:

Ropper aracı ile pop rdi adresini al. Yukarıdaki payloada tersten /x ile yazıldı

Ropper aracı ile pop rsi adresini alıcaz. Fakat rsi ret olmadan r15 de çağırılmış. R15 içinde boş bir yazı göndermeknlazım. Yukarıdaki payloada ikisi de tersten /x ile yazıldı

İşlem tmm.

**Örnek5(INJECTING CUSTOM SHELLCODE):**

Yukarıdali secret fonksiyonda assembly

komutu yazılmış. Ve çalıştırıldıpında bu kod bizi Esp registerına zıplatıyor.biz de bh register' a shelcode yüklüyoruz ve onu çalıştırtmış olıyoruz. Aşağıdaki script githubında var:

**Örnek6:(32bit)** Return to Lib C

Kaynak kodlar yukarıdaki ile aynı. Sadece koruma eklenmiş. Adresler sabit değil değişiyor.

ldd kodu ile her baktığımızda adresin değiştiğini görüyoruz

Adresleri sabitlememiz lazım. Bunun için:

Yukarıdaki sabitlediğimiz altı çizili adres pwn scriptinde libc\_base değişkeninde kullanılacak.

Yukarıdaki readelf -s komutu ile systemin adresini aldık. pwn scriptinde system değişkeninde kullanılacak.

Yukarıdaki strings kodu ile de bin/sh

komutunun adresini aldık. pwn scriptinde system değişkeninde kullanılacak.

**Örnek6:(64bit)** Return to Lib C

Kaynak kod yukarıdaki ile aynı.

64bit olduğu için ve tek parametre istediği için pop rdi komutunun adresini alıyoruz.

Bu adresi aşağıdaki pwn scriptimize ekliyoruz. Diğer yerleri yine 32bitteki yöntemlerle ekliyorsun.

**Örnek6:(32-64bit):**

**https://github.com/david942j/one\_gadget**

Bu örnekteki işleri otomatik yapıyormuş. Gün gelir belki lazım olur.

**Örnek7(Format String Vuln):**

**Önce mantığını anla:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Yukarıdaki kodda %s %d %f yer tutucuları ile değişkenlerin içeriği bellekten çağırılıyor.

Yukarıdaki %s kullanılmazsa str değişkeni dışarıdan alınıyor saldırgan kendisi %s %x %n gibi şeyler kullanarak hem bellekten gizli veriler okuyabilir hem de değiştirebilir. Okumak için %s %p %x kullanılıyor, değiştirmek için %n kullanılıyor.

Örneğin: printf("hello %n", target)

Yukarıdaki örnekte target değişkenine 6 yazılacak. Çünkü "%n" o ana kadar yazdırılan karakter sayısını bir değişkene yazdırıyor. hello + boşluk = 6 **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Şimdi soruya gelelim:

CryptoCat hazır pwn scripti ile yaptı. Çünkü burada flag direkt bellek içinde. Fuzz yaparak %s ile okuyor. 1.sıraya bak 2.ye...100.ye bak şeklinde. Çoğunda veri yok segmentetion hatası verir.

--------------------------

Bu örnekten farklı olarak Format string vuln konusunu burası da güzel anlatmış->

https://suleymanergen.com/ctf/protostar/format-string

Format1:

Bu örnekte parametre olarak alınan değer ekrana yazdırılıyor. Buradan yürüyerek target değişkenini 1 yapmamız lazım.

Format2:

**Örnek8(Leak Pie bypass):**

**Checksec:**

1.Önce ASLR kapatılır.

2.programı pwndbg ile aç. Maine breakpoint koy.

3. piebase yaz.

4.Cyclic ile offset bulunur.(RSP den)

5.fuzz.py ile değişmeyen leak pie adres bulunur. Bizdeki 15. Sırada mainin adresi. Çıkarma işleminde 3.adımda bulunan pieabase adresi kullanıldı.

6.ropper ile pop rdi adresi alınır.

7. Bulunan tüm bilgiler exploit.py ' ye eklenir. Çalıştırılıp shell alınır.

**Örnek9(Overwrite GOT with format string):**

**Checksec:**

Relro full değil, GOT üzerine yazılabilir.

Printf kısmında format string vuln var.

Bellekte yazdığımız yeri bulmamıznlazım. Yukarıda yazdığımız AAAA değerini 4. Sırada bulduk. Bunun için fuzz.py lazım. Onu THMPWN101DEN al. Daha iyi.

Ldd ile libc adresini bulıyoruz. Exploitte lazım.

Exploit.py ye gerekli bilgiler eklendikten sonra aşağıdaki kod sayesinde printf ile system değiştirilir. Printf çağırılacağı zaman sistem gelir shell alırısın.

**Örnek10(Bypass canary with format string):**

**Checksec:**

Gets kullanılmış ama canary koruması aktif. Buffer overflow olmuyor.

Fuzz.py ile yukarıda 23. Satırın canary leak olduğu bulunur. Başı f7 veya ff ile başlamıyor. Sonu 00 ile bitiyor. Bu özellikler her çalıştırmada random hep 23te denk geliyor.

Offset buffer 64, ret değeri 12.

Bu bilgilerle exploit.py yaz ve çalıştır.

11.**Null Byte ile Buffer Overflow(ctftime)**

**Tüm korumalar varken.**

**Tanım:** Null byte ile buffer overflow, bir değişkenin taşma yoluyla başka bir değişkenin değerini değiştirmek için null byte (\x00) kullanılmasıdır. Bu teknik, özellikle strcmp gibi null byte ile sonlanan stringleri karşılaştıran fonksiyonları kandırmak için kullanılır.

**Temel Adımlar:**

* **Hedef Değişken Belirleme:**
* Bellek düzenini analiz edin ve hedef değişkeni belirleyin.
* Genellikle sabit bir string ile karşılaştırma yapılan bir fonksiyon bulunur (strcmp gibi).
* **Taşma Noktasını Bulma:**
* Kullanıcı girdisinin hangi noktada taşma yaparak hedef değişkenin başlangıç byte'ını etkilediğini belirleyin.
* **Null Byte Gönderme:**
* Taşma yapacak şekilde null byte (\x00) içeren bir payload oluşturun.
* Bu payload, hedef değişkenin ilk byte'ını null yaparak strcmp gibi fonksiyonları kandırır.

**Örnek Durum:** Bir program, kullanıcıdan okunan 7 byte'lık girdiyi 6 byte'lık bir buffer'a yazar ve ardından bu girdiyi sabit bir string ile karşılaştırır:

**Payload Oluşturma:**

* Kullanıcı girdisi olarak 7 null byte (\x00\x00\x00\x00\x00\x00\x00) gönderilir.
* Bu girdinin taşması local\_18 değişkeninin ilk byte'ını null yapar (0x0073736170743377).

**Sonuç:**

* strcmp(local\_1e, local\_18) fonksiyonu iki null stringi karşılaştırır ve eşit bulur, böylece program doğru şifreyi girmişsiniz gibi davranır.

**Önemli Noktalar:**

* Null byte taşması, sadece hedef değişkenin ilk byte'ını değiştirmek için kullanılabilir.
* Bu teknik, özellikle strcmp gibi null byte ile biten stringleri karşılaştıran fonksiyonlarda etkilidir.

**Kısa Kod Örneği (pwntools):**

import pwn

p = pwn.remote('target\_address', target\_port)

p.recv() p.sendline(b'\x00\x00\x00\x00\x00\x00\x00') # 7 null byte gönderir

flag = p.recv()

print('Flag:', flag.decode())

**CTF CRYPTOCAT pwn örnekleri:**

**Jeeves:** Hackthebox

Canary hariç tüm korumalar var. Kaynak koda bakarsan da local\_c değişkenini buffer overflow ile değiştirirsen flagi

alıyorsun.

Yol1:manuel

Karşılaştırmanın olduğu yere breakpoin koy.

Cyclic yap

Cyclicteki değeri gir.

Karşılaştırmanın olduğu adres yani local\_c

değişkenine nerede etki ettiğini bul. paaa

Payloadı oluştur.

Run

Yol2:oto

https://github.com/cihangungor/CTF/blob/main/HackTheBox/pwn/jeeves/jeeves.py

**Ekstra pwn örnekler:**

**Ex1:** Strcpy ile basit overflow

$ ./stack1 `python -c 'print "a"\*64 + "dcba"'`

you have correctly got the variable to the right value

**Ex2:** getenv ile basit overflow

Bu sorudaki farklılık girdiyi doğrudan almak yerine environment veriables denilen ortam değişkenlerinden alıyor.

Önce çevresel değişkeni ekliyoruz. Sonra güm.

**Ex3:** (\*fp)() ile basit overflow

₺pwndbg> info funcions

İle win fonksiyonunun adresi alınır.

$ ./stack3 < <(python -c 'print "a"\*64 + "$\x84\x04\x08"')

calling function pointer, jumping to 0x08048424 code flow successfully changed

Extra bilgi: python2 yerine python3 ile payload yazma:

ÇIKABİLECEK KARIŞIK ÖRNEKLER

ÖRN1: pwn içinde bir yerde uzun bir sleep yani bekleme olabilir. Bu durumda pwndbg ile sleepin olduğu yere breakpoint koyulur, çalışırken jump ile bir sonraki adıma atlatılır.

ÖRN2: pwn içinde tam cmp kontrolü yapıldığı sırada stack içeriğinde search yaparak flag bulma:

Uh... Magically, get what seems to be the flag.

Then, using pwndbg search command to get the entire string :

ÖRN Shebang yöntemi ile bellekten dosya çalıştırma:

* **Bellekte "golf" adında bir dosya oluşturuluyor.**
* **memfd\_create** fonksiyonu, bu geçici dosyayı RAM’de oluşturur ve fd tanımlayıcısıyla temsil eder.
* **Kullanıcıdan gelen veri bu dosyaya yazılıyor.**
* read fonksiyonuyla 32 baytlık bir veri tampon olarak ayrılan buffer değişkenine alınıyor, ardından write fonksiyonuyla bu veri bellekteki dosyaya kaydediliyor.
* **fexecve** fonksiyonu, bellekte oluşturulan dosyayı çalıştırır. Ancak bir çalıştırılabilir dosya yapısına ihtiyaç vardır, bu yüzden içine **bash script tarzı bir shebang (#!) ile bir komut** koymak gerekiyor.
* #!/bin/cat flag.txt gibi bir komut koyarsan, dosya çalıştırıldığında cat flag.txt komutunu çalıştırır ve flag.txt içeriğini ekrana yazıyor.