

Computer System Security (INGI 2347)

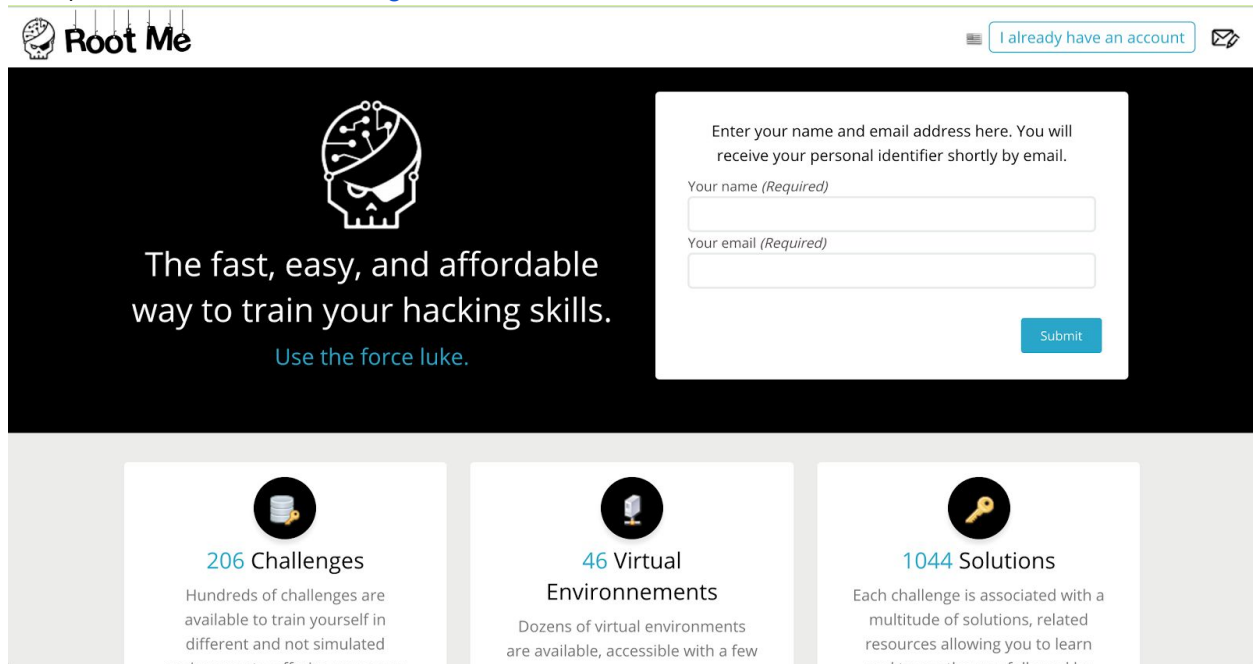
Lab Session 1: Buffer Overflow

Xiao Chen, Marco Canini

February 15, 2016

Based on the last year students' feedback, this year we decide to give you more practical exercises rather than analyze and solve theoretical questions in lab sessions. To make the lab sessions more practical, we use www.root-me.org as our exercise testing platform. Before we start our lab session, you need to sign up for an account. Unless you have done so already, please follow these steps:

- 1) Go to www.root-me.org -> Create an account;



The screenshot shows the Root Me website's registration page. At the top, the 'Root Me' logo is on the left, and a link 'I already have an account' with an envelope icon is on the right. The main content area has a dark background. On the left, there's a stylized skull icon with a brain inside, and the text 'The fast, easy, and affordable way to train your hacking skills. Use the force luke.' On the right, there's a white registration form. The form has the text 'Enter your name and email address here. You will receive your personal identifier shortly by email.' followed by two input fields: 'Your name (Required)' and 'Your email (Required)'. A blue 'Submit' button is at the bottom right of the form. Below the main content area, there are three white boxes with icons and text: '206 Challenges' (with a document icon), '46 Virtual Environnements' (with a server icon), and '1044 Solutions' (with a key icon). Each box has a brief description of the content.

- 2) Enter your name and your email;
- 3) You will receive your personal identifier shortly by email;
- 4) After you have received the email, click Login;
- 5) Enter your login or email address and password;
- 6) Change your password if you prefer;
- 7) Start hacking!

Login credentials are provided for different challenge, the goal is to obtain additional rights by exploiting program's weaknesses and get a password to validate challenges on the portal.

Exercise 1: Stack buffer overflow basic 1

Memory safety is always a concern for software developers. They try their best to avoid such type of vulnerabilities on their products such as websites, backend systems. Unfortunately, some buggy code written by careless software developers can be found on the Internet.

Suppose you happen to find the code shown below. This code prints out the value of variables `check` and `buf[40]` on the screen. Wait, what if `check` is replaced by `0xdeadbeef`... Can you describe the flaw you find in this code and suggest an exploit?

Ultimately, to validate the challenge, you must obtain the password stored in a file called `“.passwd”`. Of course, if you type `'cat .passwd'` in the console, it won't work as you don't have sufficient perms! But when you exploit the buffer overflow, you may enter a privileged shell wherein you can read the file. To do so, your exploit will need to concatenate `'cat .passwd'` at the end of the exploit string and pass it to the stdin of the program (called `ch13`).

Go to [this webpage](http://www.root-me.org/en/Challenges/App-System/ELF32-Stack-buffer-overflow-basic-1)

(<http://www.root-me.org/en/Challenges/App-System/ELF32-Stack-buffer-overflow-basic-1>) to perform the challenge and validate your exploit and solution.

```
1. #include <stdlib.h>
2. #include <stdio.h>
3.
4. /*
5. gcc -m32 -o ch13 ch13.c -fno-stack-protector
6. */
7.
8.
9. int main()
10. {
11.     int var;
12.     int check = 0x04030201;
13.     char buf[40];
14.
15.     fgets(buf, 45, stdin);
16.
17.     printf("\n[buf]: %s\n", buf);
18.     printf("[check] %p\n", check);
19.
20.     if ((check != 0x04030201) && (check != 0xdeadbeef))
21.         printf ("\nYou are on the right way !\n");
22.
23.     if (check == 0xdeadbeef)
24.     {
25.         printf("Yeah dude ! You win !\n");
26.         system("/bin/dash");
27.     }
28.     return 0;
29. }
```

Exercise 2: Stack buffer overflow basic 2

Now let's consider the code below. Can you see the flaw now? Write an exploit for it such that the program runs the privileged shell dash.

```
1. /*
2. gcc -m32 -fno-stack-protector -o ch15 ch15.c
3. */
4.
5. #include <stdio.h>
6. #include <stdlib.h>
7.
8. void shell() {
9.     system("/bin/dash");
10. }
11.
12. void sup() {
13.     printf("Hey dude ! Waaaaazzzaaaaaaaaaa ?!\n");
14. }
15.
16. main()
17. {
18.     int var;
19.     void (*func) ()=sup;
20.     char buf[128];
21.     fgets(buf,133,stdin);
22.     func();
23. }
```

Go to [this webpage](http://www.root-me.org/en/Challenges/App-System/ELF32-Stack-buffer-overflow-basic-2)

(<http://www.root-me.org/en/Challenges/App-System/ELF32-Stack-buffer-overflow-basic-2>) to perform the challenge and validate your exploit and solution.

Do you know that gdb allows you to see the address of different symbols like variables and functions?

For instance, if you run `gdb ch15` and then type `print XXX`, you will see the information of symbol XXX.

Exercise 3: Format string bug

Now let's consider the code below. Can you see the flaw now? Write an exploit for it such that the program finally shows the content of `.passwd` file. Unlike previous problems whose solution is to exploit the buffer overflow and enter a privileged shell wherein you can read the file, you may notice that the file is already open. The only thing you can control is the content of `'printf'`. How will you utilize such a chance?

```
1. #include <stdio.h>
2. #include <unistd.h>
3.
4. int main(int argc, char *argv[]){
5.     FILE *secret = fopen("/challenge/app-system/ch5/.passwd",
6.         "rt");
7.     char buffer[32];
8.     fgets(buffer, sizeof(buffer), secret);
9.     printf(argv[1]);
10.    fclose(secret);
11.    return 0;
12. }
```

Go to [this webpage](#)

(<http://www.root-me.org/en/Challenges/App-System/ELF32-Format-string-bug-basic-1>) to perform the challenge and validate your exploit and solution.

You may read [some related resource](#) to understand what is 'Format String Vulnerabilities'.