

Lab #2

Due date & Time 1:00pm on October 4th, 2007. The virtual machine for the lab will be shut down at the due time.

Late Policy: Late labs will not be accepted.

1 Overview

In this lab, you will get some experience of exploiting the software vulnerabilities, such as buffer overflow vulnerability and format string vulnerability.

1.1 The environment

You will work on a virtual machine with RedHat 6.2. To login, you first connect to the host machine (ssh forest.cs.purdue.edu) and then connect to the virtual machine (telnet 172.16.232.129).

You will work in teams of 2 people. Find your partner and send an email to TA with subject *cs426 team [lastname_lastname]*. In the body, it should include your names and emails, the account name and the initial password. The account name and the password will be used to login to both the host machine and the virtual machine.

1.2 The files

There are 5 target programs, target1, target2, target3, target4 and target5. These programs contain some bugs. Your task is to write programs to exploit these bugs. To help you start, the frames for the exploitation programs are provided.(c or shell script) The source code of the targets will be in directory targets of your home directory, and the exploits frames will be in directory exploits.

You are free to modify or add any files in the directory exploits. The files in the directory targets should remain unchanged. You may want to modify the target programs to do some tests. In that case, you should change them back to the original version when you do the real exploit and demo to TA.

Hint: Make a copy before you change to code.

1.3 Submission

You should modify the exploit code in the directory exploits to make them work. TA may ask you to demo the exploitation. Make the exploits stable and prepare for demo. You may be allowed to modify the exploits during demo, if it is due to the memory layout variant.

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising.

Hint: Exploit codes are short. Codes from others may not work.

2 Stack overflow attacks

Stack overflow is one of the most typical software vulnerabilities. The first three exploitations ask you to do basic stack overflow attacks. Read <http://reactor-core.org/stack-smashing.html> for detailed illustration of exploiting stack overflow.

2.1 Target 1 (10 pts)

Target1 takes a password as the input and checks the correctness of the password. The bug is in line 37, where strcpy is used to copy a string to another one without checking the length.

Your goal is to pass a password to target1 and make it accept the password. The successful exploitation will make the program print: Access granted to the document . . .

Hint: the function correct_passwd will always return 0!

2.2 Target 2 (20 pts)

Target2 takes a directory as the input and tells the user how to use the command ls to list the content of the directory. The bug is in line 12, where the length of variable arg is not checked.

Your goal is to pass an argument to the program to make it start a shell. Suppose this executable program is setuid root, you will login as a normal user and get a root shell!

Hint: Semicolons are used to separate consecutive shell commands.

2.3 Target 3 (20 pts)

Target3 takes a users password as the input and checks if the password is a strong one or a weak one. The bug is in line 18 where the length of variable passwd is not checked.

Your goal is to pass an argument to the program to start a shell.

The shellcode is provided in exploits/shellcode.h of your home directory.

Hint: Use *printf*("%x", &local_variable) to get the address of local variables in the stack frame, and then calculate the address of the return address.

3 Return-to-libc attacks

Return-to-libc attack is a type of buffer overflow attack, and it is generally used by attackers if the targetted system has a non-executable stack. The attack involves replacing the return

address on the stack with the address of another function (one of the libc functions in this case) and modifying other appropriate portions of the stack to provide arguments to the function. In this lab we will replace the return address with the address of `system()` and cause it to invoke a root shell.

Please read http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf before doing the lab.

3.1 Target 4 (25 pts)

Target4 takes a string as the input and copy it to a local buffer. The bug is in line 19 where the length of variable `argv[1]` is not checked.

Your goal is to pass an argument to the program to start a shell by using a return-to-libc attack.

Hint 1: Follow the instructions given in:

http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf.

Hint 2: The string `"/bin/sh"` is declared in the target program. Use *printf* to get its address.

4 Format string attacks

The format-string vulnerability is caused by code like `printf(user input)`, where the contents of variable of user input is provided by users. This vulnerability can lead to one of the following: (1) crash the program, (2) read from an arbitrary memory place, and (3) modify the values of in an arbitrary memory place.

4.1 Target 5 (25 pts)

The program has three secret values stored in its memory. They are `secret0`, `secret1` and `secret2`. The first secret is a integer value, and the second and third secrets are strings. For the sake of simplicity, we hardcode the secrets using constants, but you should pretend not to know those secrets. Although you do not know the secret values, in practice, it is not so difficult to find out the memory addresses (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:

1. (15 pts) Print out the three secret values, using the framework `exploit5.c`. 5 pts is given for each secret.
2. (10 pts) Modify the value of `secret0` to 11, using the framework `exploit5b.c`

Hint 1: Follow the instructions given in: <http://community.corest.com/juliano/tn-usfs.pdf>.

Hint 2: The address of `secret[1]` is just right after that of `secret[0]`.