Computer Security
CS 4351 / CS 5352

Spring 2017
Professor L. Longpré

Assignment 2

Introduction

The purpose of this assignment is to gain first-hand experience with exploiting a buffer-overflow vulnerability and to protect against this vulnerability.

General description

This assignment can be done individually, or in a team of 2. If it is done in a team, each one should perform the tasks on their own computer and submit their own report. The report should mention the name of the other team member. This lab was developed and tested on a Kali-linux image (light version) downloaded from:
https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download.
Also consult the user manual available on the same page for configuration, account and password information. The image's compressed size is about 1.2G, and between 3 and 4G when uncompressed and running. You can run the image using either VMware or VirtualBox. I have tested this lab using VirtualBox.
If you are not familiar with Unix commands,

`http://infohost.nmt.edu/tcc/help/unix/unix_cmd.html`

contains a list of those commonly used. You can probably find other resources on the internet.

Task 0: Setting up your virtual machine

First, you probably want to update your image to the latest version. While your virtual machine is running and connected to the internet,

```
sudo apt-get update
sudo apt-get upgrade
```

Doing the update only updates the source of packages. The upgrade is not necessary for this project and may take a lot of time, so don't do the upgrade unless you can let it run for a few hours.
The provided program uses the SHA256 function, which is not available by default. Get the `libssl` library:

```
sudo apt-get install libssl-dev
```

To easily transfer files from your host to your guest virtual machine, you may want to enable a shared folder. These instructions are for VirtualBox. Instructions for VMware are proabaly similar. In the VirtualBox menu of the running guest, select Devices, Shared Folders, Shared Folders Settings. Add the desired host shared folder. Then, in a Linux terminal window, create a folder under `/media` and mount the shared folder:

`sudo mount -t vboxsf foldername /media/directoryname`

where foldername is the name you selected when you set up the shared folder, and directoryname is a name of the folder you created. The shared folder has some restrictions in terms of permissions. You can compile and read or write files easily. But in later tasks, when you create users other than root and you want to test permission, you should not do so in the shared folder.
Compile the `main.c` program using the command: `gcc -l crypto -o fname main.c`. This will create the executable file `fname`, or the default `a.out` if you did not specify the output file name with the `-o` option.

Task 1: Running the sample vulnerable program

The provided program `main.c` has buffer overflow vulnerabilities. First, compile and run the vulnerable program (`./executableFileName`) the way it was meant to run. You can get the expected user id and passwords in the source code.
Then create a text file containing the user id and passwords. Run the program redirecting the standard input to your created text file with the command:
`./executableFileName <inputFileName`.

Report Section 1

Include a screen shot of running the vulnerable program with typed inputs and with the redirected input file.

Task 2: Studying the vulnerable program

The Vulnerable Program

The `main.c` program provided has buffer overflow vulnerabilities. The following statements cause a buffer overflow:

```
fgets(buffer, 1024, stdin);
strcpy(pw1,buffer);
```

The program asks the user for a password. It reads the password with the function `fgets()`, which is the safe version of the `gets()` function. It reads the password in a general buffer of size 1024. The passwords should be of length at most 16. With the `strcpy` function, the program copies the password from the buffer to char array of size 16. The `strcpy` function does not check boundary, so if the copy goes beyond the boundaries, it modifies other program variables.
If this program is a set-user-id program owned by root or another user and a normal user can exploit this buffer overflow vulnerability, the normal user might be able to gain unintended privileges.
Ultimately, we want exploit this program as a set-uid program owned by another user while we run it from a normal user account. But since it is inconvenient to toggle between accounts

and it is hard to debug a set-uid program, especially from a debugger, we will first study the execution of this program as a regular program from a normal user account.
We want to figure out where variables are stored when the compiled program runs.
You can do this by printing the address and contents variables. Alternatively, you can inspect this in a debugger.
In C, you can print the address and contents of a variable in hexadecimal as follows:

```
printf("i address is %p\n", (void *) &i);
printf("i contents is %08x\n",  i);
```

if `buffer` is an array, printing `buffer` is equivalent to printing `&buffer[0]`.
To print the contents of 8 words (32 bytes) of contiguous memory starting at a specific address (here where buffer starts):

```
long *ptr= (long *) buffer;
for (i=0; i<8; i++){
  printf("address %p is %08x\n",
             (void *) (ptr+i),
             (unsigned int) (*(ptr+i)));
}
```

Modify `main.c` to figure out how the memory is laid out.

Report Section 2

(a) Explain what you figured out the layout of the program's memory. Include a diagram of that layout.

It is also possible to use a debugger to investigate the memory organization If you follow that path, instead of the above, answer items above, explain the steps you took to reach your conclusion, with some screen shots as evidence.

Task 3: Exploiting the vulnerability

The purpose now is create a bad input file that will exploit the vulnerability. Construct an input file that can actually launch the shell without knowing any of the passwords. You should note that in the C language, the end of a string is indicated by a 00 byte. When you use `strcpy()`, it copies the string from a buffer to another buffer and stops copying when it reaches the 00 byte. Also, the `fgets()` function reads until the new line character is read, or until the limit is reached, whichever comes first.
Details on what to include in your report will be communicated later.

Task 4: Gain unintended privileges

Create two new users in your virtual machine. Put a compiled version of `main.c` in one user's directory. Change the permissions to make the program setusrid, with execute permission to

the other user. Place a file in the first user's directory only accessible by the owner. Then, when logged in as the second user, run the program successfully entering the shell. Verify that you now have access to the first user's restricted file.

Task 5: Exploit the vulnerability in your individualized program

An individual program will be sent by e-mail to each student. Perform Task 3 on your individual program.

Task 6: Exploiting the vulnerability on the instructor's server

Once it works on your computer, you should be able to access some secret file on the instructor's server.