CAP6135: Programming Project 1 (Spring 2010)

University of Central Florida
Cliff C. Zou

Delivery:

Submit your source code (exploit.c) and a project report document (in word or PDF format) through Webcourse homework submission page. Your project report should contain:

1. Explain how you design the overflowed buffer and why you can cause buffer overflow.
2. Draw the stack memory allocation graph to show the stack memory of the function foo() in executing target code (before running the line : `strncpy(buf, arg, len);`) You should show the addresses of (1). return address, (2) calling stack pointer, (3) the four local variables buf, i, maxlen, len. Please refer to the stack memory graph I drew in Jan. 28th lecture.
3. Show how you use Gdb to find out the stack information. You can either copy the Gdb running procedure's texts in your report, or put the screen shot of the SSH shell showing the Gdb running procedure in your report.

Objective:

You need write an attack program (exploit.c) to obtain a shell by exploiting stack overflow vulnerability of a given target application (target.c). In reality, if this target application is running on a server accepting remote user input, you can use your exploit code to obtain a command shell (most likely to have root privilege) on the remote server without any password.

This project is modified from the programming project 1 in U. Berkley's Dr. Dawn Song's course "CS161: computer security" in Fall 2008:
http://inst.eecs.berkeley.edu/~cs161/fa08/

Target

The target (target.c) is a simple application which copies the command-line parameter into its own internal buffer stored on the stack. The source code is provided for your reference. Of course you can build your own target while working on the assignment, but the submitted code MUST use the prebuilt target.c.

Exploit

The exploit is the application (exploit.c) which calls the target (executable program) and passes in an artfully crafted character buffer which causes the target program to give you a shell.

The compressed file in the assignment, "assignment1.tar", contains two directories: exploits and targets. You can extract it under Eustis Unix machine using this command:

```
tar -xvf assignment1.tar
```

You will need to generate the executable "target" first. Then you will need to modify "exploit.c" to add your overflow code. Noted that you will need to modify:

```
#define TARGET "/home/czou/buffer-code/targets/target"
```

to use your own “target” directory.

You can download the compressed file “assignment1.tgz” from webcourse@UCF at the first assignment place.

The following are the suggested steps to complete this assignment:
1. Read the “Smashing The Stack for Fun and Profit” tutorial at http://insecure.org/stf/smashstack.html
2. Make sure you understand it.
3. Familiarize yourself with gdb. Be sure to watch my lecture given on Jan. 28th on how I used gdb. You will need gdb to debug your target/exploit and display memory/stack information. With it help, you can decide the where is the return address to overwrite and how to jump to execute the provided shellcode. Look up commands: break, run, continue, info f, p/x, backtrace, attach…
4. Draw the stack (locations of parameters, local variables and registers) of the target application once execution reaches the first statement of the foo function.
5. Figure out the length of the exploit buffer.
6. Figure out where in the exploit buffer the new RET address will go.
7. Compute the new RET address.
The code to spawn a shell is provided in the shellcode.h file. You can use testshellcode to test it.

Important notes:
Eustis.eecs.ucf.edu (the server you’ll be working on) employs a technique called “Stack address randomization”, which randomizes the beginning of the stack for each new process. Therefore your SP register will always be different when you start a new process. This behavior is not desirable for this assignment, because we want the SP to be the same every time we execute the exploit/target (see the aforementioned tutorial for why this is so important).

The way around this is by using the wrapper **setarch** in the following way:
czou@eustis:~/buffer-code/exploits$ **setarch i686 -R ./exploit**
The –R parameter disables stack address randomization and creates a new process by executing ./exploit.

In addition, please use the “Makefile” under each directory to generate your executable code. The Makefile contains option that will disable “StackGuard” defense used by our Unix server.

Grading:
[+10%] if your program compiles and runs
[+10%] if your program can successfully overflow the buffer
[+10%] for correctly showing how you use gdb to obtain the stack information.
[+10%] for correct drawing of the stack (in the document) with the addresses.
[+60%] if execution of target gives the user a shell
[-20%] if you submitted your project late within the late submission deadline.