F14 18-487 Introduction to Computer Security, Network Security, and Applied Cryptography
Homework #1

David Brumley, Peter Chapman, Zachary Weinberg
Problem Design and Implementation by Jiyong Jang

Due: **2:30 pm on September 22, 2014**

September 15, 2014

1 Introduction

As a special agent from the `18487` hacking group, you have been assigned a secret mission. In this mission (a.k.a. *homework*), you need to break into the `DJB` company's systems and steal secret documents. Fortunately, an important executable file has been stolen from the `DJB` server and is stored in `recoverpw.tar.gz` for your mission. The `DJB` company runs three servers: the CREDENTIALS system, the SECURE system and the SUPERSECURE system. Tasks 1 is to be run on the CREDENTIALS system where ASLR is turned *off*. Tasks 2 and 3 are to be run on the SECURE system where ASLR is also turned *off*. Tasks 4 and 5 are to be run on the SUPERSECURE system where ASLR is turned *on*.

1.1 Grading

1.1.1 Rules

- You can discuss with others, but you must create *your own* exploits.
- Use Piazza to ask questions.
- **DO NOT** try to obtain root access on the systems.
- **DO NOT** brute-force the systems.

1.1.2 Criteria

Each task will be graded based upon the following 3 criteria. Partial credit will be given as deemed appropriate by the graders.

- **Reversing (10 pts):** Explain what the program does, especially how the *user input* is processed (feel free to present a pseudocode version of the program). Then, identify the *vulnerability* in the program, i.e., which kind of vulnerability it is and where the vulnerability is. Be sure to explicitly name the functions relevant to the vulnerability (*strcpy*, *printf*, etc.).
- **Exploiting (10 pts):** Create *your own exploit* and provide a full description of how your exploit works with a *stack diagram*. Be sure to explain how your exploit gets around any countermeasures in the target program.
- **Pwning (5 pts):** Obtain a *secret key* using your exploit.

1.1.3 Submission

You need to submit

- a **writeup** describing the 3 criteria above for every task. Your writeup must be **UNDERSTANDABLE**!
- working **exploits** which you need to put under your home folder on the SECURE or SUPERSECURE system. Your exploit will be tested from the command line, *not* in the `gdb`.

1.2 Scoreboard

When you complete a task, you will get a secret key (a.k.a. flag) in the form of an MD5 hash. A valid secret key is the evidence showing you solved the problem. Submit your secret key promptly to the submission server with your Andrew ID. Keep in mind that you must put **your correct Andrew ID** because we also use the scoreboard to verify that your exploit really works. **DO NOT** share the secret key with your friends.

Submission:	`http://debian.ece.cmu.edu:7397/submit`
Scoreboard:	`http://debian.ece.cmu.edu:7397/score`

NOTE: There will be *breakthrough* points. For each task, the **first** student to complete the task and submit the valid secret key will be given an **extra 5** points. A student can only receive breakthrough points for one problem allowing for a total of five different students to receive breakthrough points on the assignment.

1.3 Servers

There are three servers you will use to for this homework.

CREDENTIALS system:	`debian.ece.cmu.edu:7321`
SECURE system:	`debian.ece.cmu.edu:8952`
SUPERSECURE system:	`debian.ece.cmu.edu:35989`

2 Task 1: `recoverpw` (25 pts)

Your first task is to infiltrate the `DJB` company's network. By eavesdropping conversations between employees of the `DJB` company, you got to know a password recovery service is running on the CREDENTIALS system, which is accessible via port `7321` on `debian.ece.cmu.edu`. You can connect to the service using the `nc` program in the following way:

```
nc debian.ece.cmu.edu 7321
```

Your goal is to retrieve the SSH login credentials for the other systems by exploiting the password recovery service (or maybe by guessing a correct PIN...). You should analyze and understand the binary `recoverpw.bin` first, then find the vulnerability, and make an exploit for it. A successful attack will give you the login information. (*Hint:* you might be interested in the `recover_passwd` function.)

Tips:

1. You can print hexadecimals using `perl`. Example: `perl -e 'print "\xFF\xFF"x4'`
2. You can send a message by: `echo "hello" | nc debian.ece.cmu.edu 7321`
3. Or, you can use the given `client.pl` to communicate with the server.

3 Task 2: `cfo.snote` (25 pts)

Now you are in the `DJB` company's SECURE system. The Chief Financial Officer (CFO) of `DJB` uses a program called `snote` (secure note) to record confidential messages. You can find the program `snote` and a confidential document `secret` at `/home/cfo`.

Your goal is to read `secret` by exploiting `snote`. (*Hint:* you might be interested in `note` function.)

Tips:

1. Due to the environment variables, the stack addresses when a program runs in `gdb` may be different from those when the program is run directly. To mitigate this problem, run the following commands in `gdb` before running the program:

   ```
   unset environment COLUMNS
   unset environment LINES
   unset environment OLDPWD
   set environment _ "/home/cfo/snote"
   ```

   Then, use *the absolute path* when running at the command line.

2. You can find various shellcode for Linux at `http://www.shell-storm.org/shellcode/shellcode-linux.php`.
   For example, in order to spawn a shell, you can use this shellcode: `http://www.shell-storm.org/shellcode/files/shellcode-752.php`.

3. Environment variables related to the name of the user or even the path to the current working directly can cause an exploit to be unreliable. One technique for building a robust exploit is to use a series of NOP instructions (`0x90`) to form a NOP slide: `http://en.wikipedia.org/wiki/NOP_slide`.

4. Running a program under `gdb` disables its `setuid` privileges. In order to read the various 'secret' files you must come up with an exploit that works when the program is run directly from the shell.

5. The `whoami` command is helpful if you are not sure whether you have successfully launched a shell with extra privileges.

4 Task 3: `cto.snote` (25 pts)

Go to the `/home/cto` directory. The Chief Technology Officer (CTO) of `DJB` developed a more secure version of `snote` to make secret notes by adding a buffer-overflow checking routine.

Your goal is to, again, read `secret` by exploiting this more secure version of `snote`. (*Hint:* you might be interested in `note` function.)

5 Task 4: `ceo.snote` (25 pts)

Great job! You have completed all the required tasks on the SECURE system. Let's move on to the SUPERSECURE system.

The Chief Executive Officer (CEO) of `DJB` is worried about hacking and moved CTO's `snote` to the SUPERSECURE system where ASLR is turned on. The CEO also improved the buffer-overflow checking routine. Your goal is to read `secret` by exploiting `snote` located at the `/home/ceo` directory.

NOTE 1: Regarding the **Reversing** criterion, instead of copy-and-paste, describe why your exploit for `cto.snote` does not work for `ceo.snote`. For example, compare the output of the command
`cat /proc/sys/kernel/randomize_va_space` on SECURE and SUPERSECURE systems.

NOTE 2: Your exploits for `cto.snote` and `ceo.snote` should be *different*. The main difference would be figuring out stack addresses by using `gdb` vs. bypassing ASLR altogether. **DO NOT** brute-force the system.

6 Task 5: `vp.snote` (25 pts)

Your last task is to read a Vice President's (VP) `secret` message by exploiting `snote` located in the `/home/vp` directory. VP is also concerned about hacking and so VP copied CFO's `snote` to the SUPERSECURE system, made a few modifications, and further made the stack of `vp.snote` *non-executable*. Good luck on your final task!

NOTE: Regarding the **Reversing** criterion, instead of copy-and-paste, describe why your exploit for `cfo.snote` does not work for `vp.snote`.

Tips:

1. The Global Offset Table, which contains the addresses of dynamically linked functions, is not placed in a randomized location by ASLR and can be viewed with `readelf -r`.
2. The short write variation of a format string attack drastically reduces the amount printed to standard output.