UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Spring 2018**
Hassan Khan

ASSIGNMENT 1

Blog Task signup due date: **Mon, May 14, 2018 *3*:00pm (no extension)**
Milestone due date: **Fri, May 25, 2018 *3*:00pm**
Assignment due date: **Mon, June 4, 2018 *3*:00 pm**

Total Marks: 52 (+10 bonus)
Blog Task TA: Masoumeh Shafieinejad
Written Response TA: Pierfrancesco Cervellini
Programming TA: Sung-Shine Lee and Navid Esfahani

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are posted to Piazza.

Blog Task

0. [0 marks, but -2 if you do not sign up by the due date] Sign up for a blog task timeslot by the due date above. The 48 hour late policy, as described in the course syllabus, does not apply to this signup due date. Look at the blog task in the Course Materials, Content section of the course website to learn how to sign up.

   Please visit https://crysp.uwaterloo.ca/courses/cs458/infodist/blogtask.php to sign-up.

Written Response Questions [22 marks]

Note: Please ensure that written questions are answered using complete and grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as on the correctness of your answers.

Your mission, should you choose to accept it…

You are an employee of a well-known-for-being-shady political consulting firm Oxford Analytica (OA). Over the last month, things have become murkier than usual and allegations have surfaced that the company has been misappropriating people's data in an effort to influence the presidential election. OA is a large company so when by pure happenstance you catch an UberPool with two members of the data harvesting team, they don't recognize you. They are talking about their work, and what you hear makes it clear—it is time to blow the whistle.

1. For each of the following, please:

   - identify the scenario as a compromise of Confidentiality, Integrity, Availability, and/or Privacy,
   - and, briefly explain your choice of compromise.

   (a) (2 marks) You discover that a few years back Oxford Analytica used a mobile application to harvest a few hundred thousand Facebook profiles, but it turns out that due to loose restrictions by the Facebook's API they were able to download the full profiles of close to 52 million unsuspecting users who were friends of the mobile application users.

   (b) (2 marks) Fortunately, so many access attempts from the mobile application is detected by Facebook. However, the defense strategy of Facebook prevents all further incoming requests from the IP that is used by the mobile application.

   (c) (2 marks) Unfortunately, though 52 million data points are a lot, they are not enough to create an accurate model for the election that the company is trying to influence. One of them recommends adding a few million fake profiles to the data. After all, who would know.

   (d) (2 marks) The other seems to think this is a good idea. He adds that OA's Chief Security Officer has privately revealed to him the existence of a company owned, secret server farm that does not track any usage metadata. If they dumped the fake data there, truly no one would know they made it all up.

   The whistleblower

   You are unsettled by the conversation you overheard, but if you are going to blow the whistle, you need more information, and hard facts to prove Oxford Analytica is up to no good. You start doing some old fashioned detective work, and soon people start realizing your questions might be just a little too specific. You start getting clues that you may have overplayed your hand.

2. For each of the following, please:

- answer which of interception, modification, interruption, and/or fabrication, threats are represented in each of the following scenarios
- and, give a brief explanation for each of your answers.

(a) (2 marks) Looking through your company repository commit history you notice strange commits you do not recognize and do not remember doing.

(b) (2 marks) After taking a closer look you discover some of the commits are changes to an authentication protocol for access to secret client files.

(c) (2 marks) On a hunch you decide to check the internet packet routing from your work terminal and discover that your data is going through a hop that should not be there.

(d) (2 marks) The most obvious hint that something is very wrong is that you seem to have lost access to the more sensitive company files.

Running for cover

You made up your mind about exposing Oxford Analytica and are considering contingencies to minimize your own risks.

3. For each of the following, please:
   - explain how you could use the defense to your advantage
   - try to provide context that fits the narrative of your Oxford Analytica assignment.
   - The first one has been done for you.

(a) Deflecting.
Place some incriminating documents on the desk of a co-worker with a history of misbehaviour.

(b) (2 marks) Preventing.

(c) (2 marks) Detecting.

(d) (2 marks) Recovering.

Programming Question [30 marks + 10 bonus]

Background

You are tasked with testing the security of a custom-developed *file submission application* for your organization. It is known that the application was *very poorly written*, and that in the past, this application had been exploited by some users with the malicious intent of *gaining root privileges*. There is some talk of the application having *three or more vulnerabilities*! As you are the only person in your organization to have a background in computer security, only you can *demonstrate how these vulnerabilities can be exploited* and *document/describe your exploits* so a fix can be made in the future.

Application Description

The application is a very simple program to submit files. It is invoked in the following way:

- `submit <path to file> [message]` : this will copy the file from the current working directory into the submission directory, and append the string "message" to a file called `submit.log` in the user's home directory.

There may be other ways to invoke the program that you are unaware of. Luckily, you have been provided with the source code of the application, `submit.c`, for further analysis.

The executable `submit` is *setuid root*, meaning that whenever `submit` is executed (even by a normal user), it will have the full privileges of *root* instead of the privileges of the normal user. You can check which user you are running as with the command `whoami`.

Testing Environment

To help with your testing, you have been provided with a virtual *user-mode linux* (uml) environment where you can log in and test your exploits. These are located on one of the *ugster* machines. To obtain your ugster account information, go through the quest authentication and visit: https://crysp.uwaterloo.ca/courses/cs458/infodist/ugster.php

Once you have logged into your ugster account with SSH, you can use the `uml-a1` command to start your virtual Linux environment. The following logins can be used:

- `user` (no password): main login for virtual environment
- `halt` (no password): halts the virtual environment, and returns you to the ugster prompt

The executable `submit` application has been installed to `/usr/local/bin` in the virtual environment, while `/usr/local/src` in the same environment contains `submit.c`. Conveniently, someone seems to have left some shellcode in `shellcode.h` in the same directory.

It is important to note all changes made to the virtual environment will be lost when you halt it. Thus it is important to remember to keep your working files in `/share` on the virtual environment, which maps to `~/uml/share` on the ugster environment.

Rules for exploit execution

- You have to submit two exploit programs to be considered for full credit. Only one of your exploit programs MUST target either a buffer overflow vulnerability or a format string vulnerability. The other may target any other vulnerability except a buffer overflow vulnerability or a format string vulnerability.
- Each vulnerability can be exploited only in a single exploit program. A single exploit program can exploit more than one vulnerability. If unsure whether two vulnerabilities are different, please ask a private question on Piazza.
- There is a specific execution procedure for your exploit programs ("*sploits*") when they are tested (i.e. graded) in the virtual environment:
  - Sploits will be run in a **pristine** virtual environment, i.e. you should not expect the presence of any additional files that are not already available
  - Execution will be from a clean `/share` directory on the virtual environment as follows: `./sploitX` (where X=1..3)
  - Sploits must not require any command line parameters
  - Sploits must not expect any user input
  - If your sploit requires additional files, it has to create them itself
- For marking, we will compile your exploit programs in the `/share` directory in a virtual machine in the following way: `gcc -Wall -ggdb sploitX.c -o sploitX`. You can assume that `shellcode.h` is available in the `/share` directory.
- Be polite. After ending up in a root shell, the user invoking your exploit program must still be able to exit the shell, log out, and terminate the virtual machine by logging in as user `halt`. None of the exploits should take more than about a minute to finish.
- Give feedback. In case your exploit program might not succeed instantly, keep the user informed of what is going on.

Deliverables

Each sploit is worth 15 marks, divided up as follows:

- 10 marks for a successfully running exploit that gains root
- 5 marks for a description of the vulnerability used, an explanation of how your sploit program exploits the vulnerability, and a description of how the vulnerability could be fixed

A total of two exploits must be submitted to be considered for full credit, with only one being a *buffer overflow* or *format string* exploit. Marks may be docked if you do not submit a buffer overflow or format string exploit or submit both.

Bonus exploits

You may submit *at most one* extra vulnerability for bonus points. It must exploit a distinct vulnerability from your first two exploits. Bonus marks will be given as follows:

- 6 marks for a successfully running exploit that gains root
- 4 marks for a description, as above. Note that bonus points *will not be awarded* for descriptions without accompanying working exploit code.

What to hand in

All assignment submission takes place on the `student.cs` machines (not ugster or the virtual environments), using the `submit` utility. In particular, log in to the Linux student environment (`linux.student.cs.uwaterloo.ca`), go to the directory that contains your solution, and submit using the following command: `submit cs458 1 .` (dot included). CS 658 students should also use this command and ignore the warning message.

By the **milestone due date**, you are required to hand in:

sploit1.c One completed exploit programs for the programming question. Note that we will build your sploit programs **on the uml virtual machine**

a1-milestone.pdf: A PDF file containing your answers for the written-response questions, and the exploit descriptions for sploit1.

Note: You will not be able to submit sploit1.c, or a1-milestone.pdf after the milestone due date (plus 48 hours).

By the **assignment due date**, you are required to hand in:

sploit2.c: one exploit programs for the programming question.

(optional) sploit3.c: Exploit program for the programming bonus question.

a1.pdf: A PDF file containing your exploit descriptions for sploit2, and optional sploit3. Do not put written answers pertaining to sploit1 and written-response questions into this file; they will be ignored.

The 48 hour late policy, as described in the course syllabus, applies to the milestone due date and the assignment due date. It does not apply to the blog task signup due date.

Useful Information For Programming Sploits

Most of the exploit programs do not require much code to be written. Nonetheless, we advise you to start early since you will likely have to read additional information to acquire the necessary knowledge for finding and exploiting a vulnerability. Namely, we suggest that you take a closer look at the following items:

- Module 2
- Smashing the Stack for Fun and Profit (http://insecure.org/stf/smashstack.html)
- Exploiting Format String Vulnerabilities (v1.2) (http://julianor.tripod.com/bc/formatstring-1.2.pdf) (Sections 1–3 only)
- The manpages for execve (man execve), pipe (man pipe), popen (man popen), getenv (man getenv), setenv (man setenv), passwd (man 5 passwd), shadow (man 5 shadow), symlink (man symlink), expect (man expect).

GDB

The gdb debugger will be useful for writing some of the exploit programs. It is available in the virtual machine. In case you have never used gdb, you are encouraged to look at a tutorial (e.g.,http://www.unknownroad.com/rtfm/gdbtut/).

Assuming your exploit program invokes the `submit` application using the `execve()` (or a similar) function, the following statements will allow you to debug the `submit` application:

1. `gdb sploitX` (X=1..3)
2. `catch exec` (This will make the debugger stop as soon as the `execve()` function is reached)
3. `run` (Run the exploit program, which will stop when the exec of `submit` happens)
4. `symbol-file /usr/local/bin/submit` (We are now in the `submit` application, so we need to load its symbol table)
5. `break main` (Set a breakpoint in the `submit` application)
6. `cont` (Run to breakpoint)

You can store commands 2–6 in a file and use the "`source`" command to execute them. Some other useful gdb commands are:

- "`info frame`" displays information about the current stackframe. Namely, "saved eip" gives you the current return address, as stored on the stack. Under saved registers, eip tells you where on the stack the return address is stored.
- "`info reg esp`" gives you the current value of the stack pointer.
- "`x <address>`" can be used to examine a memory location.
- "`print <variable>`" and "`print &<variable>`" will give you the value and address of a variable, respectively.
- See one of the various gdb cheat sheets (e.g., http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf) for the various formatting options for the print and x command and for other commands.

Note that `submit` will not run with root privileges while you are debugging it with gdb. (Think about why this limitation exists.)

The Ugster Course Computing Environment

In order to responsibly let students learn about security flaws that can be exploited in order to become "root", we have set up a virtual "user-mode linux" (uml) environment where you can log in and mount your attacks. The gcc version for this environment is the same as described in the article "Smashing the Stack for Fun and Profit"; we have also disabled the stack randomization feature of the 2.6 Linux kernel so as to make your life easier. (But if you'd like an extra challenge, ask us how to turn it back on!)

To access this system, you will need to use ssh to log into your account on one of the `ugster` machines: `ugsterXX.student.cs.uwaterloo.ca`. There are a number of ugster machines, and each student will have an account for one of these machines. To obtain your ugster account information, visit: https://crysp.uwaterloo.ca/courses/cs458/infodist/ugster.php. Authentication is done via UW CAS and uses your quest student information.

The ugster machines are located behind the university's firewall. While on campus you should be able to ssh directly to your ugster machine. When off campus, you have the option of using the university's VPN (see these instructions), or you can first ssh into `linux.student.cs.uwaterloo.ca` and then ssh into your ugster machine from there.

When logged into your ugster account, you can run "`uml-a1`" to start the user-mode linux to boot up a virtual machine.

The gcc compiler installed in the uml environment may be very old and may not fully implement the ANSI C99 standard. You might need to declare variables at the beginning of a function, before any other code. You may also be unable to use single-line comments ("//"). If you encounter compile errors, check for these cases before asking on Piazza.

Any changes that you make in the uml environment are lost when you exit (or upon a crash of user-mode linux). **Lost Forever**. Anything you want to keep must be put in `/share` in the virtual machine. This directory maps to `~/uml/share` on the ugster machines, which is how you can copy files in and out of the virtual machine. It can be helpful to ssh twice into the ugster. In one shell, start user-mode linux, and compile and execute your exploits. In the other shell, edit your files directly in `~/uml/share/`, to ensure you do not lose any work. The ugster machines are not backed up. You should copy all your work over to your student.cs account regularly.

When you want to exit the virtual machine, use exit. Then at the login prompt, login as user "halt" and no password to halt the machine.

Any questions about the ugster environment should be directed to Navid Esfahani.