UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Spring 2016**
Erinn Atwater and Doug Stinson

ASSIGNMENT 1

Assignment due date: **Thursday, May 26, 2016** **3:00 pm**

Total Marks: 60 (+5 bonus)
Written Response TA: Lalit Agarwal **Office hours:** Thursdays 12:00–1:00pm, DC 3332
Programming TA: Chenyu Yao **Office hours:** Tuesdays 12:30–2:30pm, DC 2582

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

Written Response Questions [30 marks]

Note: For written questions, please be sure to use complete, grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as the content.

1. (6 marks) Identify each of the following as a compromise of Confidentiality, Integrity, Availability, and/or Privacy, with a brief explanation.
   (a) Your doctor sells your medical information to a drug company in exchange for a discount on prescriptions.
   (b) Someone sets up a WiFi access point in a coffee shop, which replaces websites for its users with ones that deliver malware.
   (c) A hacker breaks into a dating site and steals user account information.
   (d) A child installs spying software on their parent's work phone.
   (e) Someone replaces the login program on your laptop with one that sends your password over the internet.
   (f) A herd of wild muffalo chew on a buried Internet cable.
2. (8 marks) Peg DHCP[1] is a method of distributing addresses via clothespins at hacker conventions. A central desk keeps all of the available addresses written on pegs attached to a

[1] `https://tools.ietf.org/html/rfc2322`

clothesline, and hands them out to attendees who request an address. Identify four threats that apply to the distribution of addresses to attendees, and categorize them as one of the major categories: *interception*, *interruption*, *modification*, and *fabrication*. You must include at least one threat from each category. Be creative.

Furthermore, suggest a potential defence for each of the four attacks, and classify the defence as prevention, deterrence, deflection, detection, or recovery. You may have duplicate types for this part.

3. (16 marks) The *Report on Business* is a monthly magazine that is distributed free with the *Globe and Mail*. In the May 16 issue of the magazine, there is an article entitled *Best-kept Secrets* that discusses several aspects of security/privacy, including the recent Apple-FBI case:

   `http://www.theglobeandmail.com/report-on-business/rob-magazine/best-kept-secrets-the-battle-to-safeguard-our-privacy/article29759575/`

   Unfortunately, this article is rife with inaccuracies and misunderstandings on the part of the author. The purpose of this question is to explore and critique some of the issues discussed in the article. You will be required to give written answers, using complete sentences formed into well-structured and coherent arguments. Each part should require no more than a couple of paragraphs to answer. You will probably have to do some research on the internet to prepare your answers. Remember to give appropriate attributions for information you use. Quoted material must be enclosed in quotation marks, and all material you use from an external source (whether quoted or paraphrased) must be appropriately cited.

   (a) The author uses the phrase *digital lock* several times, apparently meaning *encryption*. What does the phrase *digital lock* actually mean (especially in the context of copyright protection)?

   (b) Apple has often co-operated with law enforcement in the past, by supplying decryption keys when it was presented with a court order. What is different about the recent Apple-FBI case and why did Apple balk at the FBI's request?

   (c) The author thinks that "breaking encryption" means that keys will fall into the wrong hands and consequently any database that shares the same digital lock will be cracked. Critique this statement. In particular, what is the author saying, and does it make any sense whatsoever?

   (d) The author uses the phrase "being able to assure consumers that the data they often unwittingly hand over will be kept private is crucial to gaining their trust". Is the author using the term private in the way we understand it? Discuss briefly.

Programming Question [30 marks + 5 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. You will be e-mailed by the programming TA with your account credentials for your designated ugster machine.

Once you have logged into your ugster account with SSH, you can use the `uml` 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 three exploit programs to be considered for full credit. One of your exploit programs MUST target either a buffer overflow vulnerability or a format string vulnerability. The other two may target other vulnerabilities.
- 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 10 marks, divided up as follows:

- 6 marks for a successfully running exploit that gains root
- 4 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 three exploits must be submitted to be considered for full credit, with at least 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.

Bonus exploits

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

- 3 marks for a successfully running exploit that gains root
- 2 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 **assignment due date**, you are required to hand in:

sploit1.c, sploit2.c, sploit3.c: The three exploit programs for the programming question.

(optional) sploit4.c: Exploit program for the programming bonus question.

a1.pdf: A PDF file containing your answers for the written-response questions, and the exploit descriptions for all of your sploits (including the bonus, if applicable).

Note: The 24 hour 25%-penalty late policy, as described in the course syllabus, applies to the assignment due date. No late assignments will be accepted after 24 hours.

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. You will get an e-mail with your password and telling you which ugster you are to use. If you do not receive a password please check your spam folder.

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`” 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 the Programming Question TA.