UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Fall 2017**

Urs Hengartner
Stefanie Roos

ASSIGNMENT 1

Blog Task signup due date: **Tuesday, September 19th, 2017** 3:00pm (no extension)
Milestone due date: **Tuesday, September 26th, 2017 3:00pm**
Assignment due date: **Friday, October 6th, 2017 3:00 pm**

Total marks: 70
Written Response Questions TA: Bailey Kacsmar
Programming Question TA: Cecylia Bocovich & Erinn Atwater

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

Written Response Questions [30 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 the content.

1. For each of the following identify whether the scenario violates the property of confidentiality, integrity, availability, and/or privacy. For each scenario indicate one property that is being violated and explain how/why this property is violated in the scenario. (8 marks)

   (a) You sent an email to your friend Alice containing a number of cute cat photos to cheer them up. Mallory intercepts the message and thinks it would be funny to change all of the images to be of puppies before she sends the message back on its way to Alice.

(b) While in the airport you connect to what you believe to be a secure WiFi network and you head to your favorite web page for baby alpaca pictures. When attempting to login to your account on the site to see your favorite saved baby alpacas you find that the page will not load after you enter your credentials. Additionally, when you entered your information the data (password, username) was sent on to an additional party who had replaced the login screen on the alpaca site.

(c) You regularly access an online store to purchase purple spider posters and other neat items. Unknown to you the site sells any information you provide as well as information on your purchases to an advertising company.

(d) The May 2017 WannaCry attacks

2. For each of the following identify whether the attack violates the property of confidentiality, integrity, availability, and/or privacy **AND** whether the attack is an example of interception, interruption, modification, or fabrication. Your answer should include one of {confidentiality, integrity, availability, privacy } and one of {interception, interruption, modification, fabrication} for each scenario. (8 marks)

   Explain (4 marks) why each scenario is an example of one of {interception, interruption, modification, fabrication}.

   Alice likes to send written messages to Bob using a courier horse. Alice gives her message to a courier who rides a horse for the 6 hours it takes to get to Bob. The courier then gives the message to Bob.

   (a) Mallory has observed that Alice sends Messages to Bob. One day Mallory decides to stop the courier for a chat and offers him some delicious cookies. While he enjoys the cookies she slips a letter she wrote into his bag addressed to Bob from Alice.

   (b) Having enjoyed sending her own little message, Mallory decides she wants to learn more about what is being said between Alice and Bob. She again gives the courier cookies, and Mallory secretly acquires the letter from Alice to Bob and reads it before returning it to the courier's bag.

   (c) Since Alice continues to send messages to Bob, Mallory has decided it is time to change the meaning. She takes Alice's letters from the courier's bag and changes the contents to mean the opposite of what Alice wrote.

   (d) Mallory decides to give the courier Macaroons the next day as she has heard they are better then cookies. Having previously read messages from Alice to Bob, Mallory decides she does not approve of their communications and steals the letter from Alice from the courier's bag.

   (1 mark) Give an example in the digital world (ie: email) that corresponds to one of the above attacks. Indicate which attack and briefly describe the digital example.

3. You have been placed in charge of ensuring the security of the data for your company. In order to do so you take a number of measures. For each of the plans below, please indicate

what type of threat defence is being used. You may choose one of Preventing, Deterring, Deflecting, Detecting, and Recovering for each. Additionally, explain why it is the defense you indicate it to be. (8 marks)

(a) You enclose the company's data servers in a "fire-proof" room.

(b) You install sensors to indicate whether or not water has entered the server room which is located near a flood plane.

(c) All data on the servers in this location (near the flood plane) are backed up to another location twice a day.

(d) Among the main servers you have included an additional less secure server which stores no real or confidential information. It typically exists for testing new software without risk to any real information, but since it is no real loss were it to go offline or be re-set it is left open to some attacks.

4. (1 marks) Your company stores hashed versions of passwords. Since they are properly hashed there is no way to recover the original plaintext passwords. This is unlike when you encrypt a message, and can retrieve the original message later. What functionality does not exist for hash functions that is used to recover a message that has been encrypted?

Programming Question [40 marks]

Background

A custom-developed *backup application* has been circulating among small technical companies recently. 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*. It is well known that the application has *four or more vulnerabilities*! For that reason, many companies have decided not to run the program as root and have made attempts to patch the vulnerabilities in the original program. However, the employees making these patches are not experts and have called you in to make sure their patched programs are secure. You have been given access to five different patched programs from five different companies. After talking to the employees that made the patches, you are certain that each program is vulnerable to *at least one* of the original flaws. It is up to you to *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 with the purpose of backing up and restoring files. There are at least three ways to invoke it:

- `backup backup foo` : this will copy file *foo* from the current working directory into the backup space.
- `backup restore foo` : this will copy file *foo* from the backup space into the current working directory.
- `backup ls` : this will list the files stored in the backup space.

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 original application, `backup.c`, for further analysis.

The executable `backup` is *setuid*, meaning that whenever `backup` is executed (by any user), it will have the full privileges of the user that *owns the program* instead of the privileges of the user that invokes it. Therefore, if an outside user can exploit a vulnerability in a setuid target, he or she can cause the target to execute arbitrary code (such as shellcode) with the full permissions of the user that owns it. If you are successful, running your exploit program will execute the setuid backup, which will perform some privileged operations, which will result in a shell with the privileges of the user that patched it. You can verify that the resulting shell has a specific user's privileges by running the `whoami` command within that shell. The shell can be exited with `exit` command.

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 can retrieve your account credentials from the Infodist system.

Once you have logged into your ugster account, you can run `uml` to start your virtual linux environment. The following logins are useful to keep handy as reference:

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

You are testing the patched backup executables of five different users: alice, bob, carol, david, and eve. The executable `backup` application for each user has been installed to `/home/username/` in the virtual environment, where username corresponds to each of the five users. The original (unpatched source code can be found in `/usr/local/src/backup.c`. Conveniently, someone seems to have left some shellcode in `shellcode.h` in the same directory.

Each of the five users has also placed a file called `flag.txt` in their home directories. When you successfully exploit their patched backup program, you should be able to obtain the value of their flag to demonstrate that you have obtained access to their private files.

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 four exploit programs to be considered for full credit. Two of your submitted exploit programs must exploit specific vulnerabilities. Namely, one must target a buffer overflow vulnerability, and another must target a format string vulnerability. Your other submitted exploit programs can address 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. Each of the four exploit programs should exploit a backup executable corresponding to a different user.
- 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..4)
  - 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 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`. Also, please do not run any cpu-intensive processes for a long time on the ugster machines (see below). 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 a shell owned by the user of the vulnerable backup executable

- 4 marks for the description of the identified vulnerability/vulnerabilities, saying how your exploit program exploits it/them, and **describing how it/they could be repaired.**

A total of four exploits must be submitted to be considered for full credit. Marks will be docked if you submit no *buffer overflow* sploit or no *format string* sploit. **Note:** sploit1.c and sploit2.c are due by the milestone due date given above. You can but do not need to submit the buffer overflow sploit or format string sploit by the milestone due date.

What to hand in

It is very important that you follow the rules outlined in the Assignments section of the LEARN course site for submitting your assignment. Otherwise we may not be able to mark your assignment and you may lose partial or all marks.

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

sploit1.c, sploit2.c Two 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 exploit descriptions for sploit1 and 2 (including repairs, as explained above)

Note: You will not be able to submit sploit1.c, sploit2.c or a1-milestone.pdf after the milestone due date (plus 48 hours).

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

sploit3.c, sploit4.c: The two remaining exploit programs for the programming question.

a1.pdf: A PDF file containing your answers for the written-response questions, and the exploit descriptions for sploit3 and 4 (including repairs, as explained above). Do not put written answers pertaining to sploit1 and 2 into this file; they will be ignored. Also include a copy of the table below, filled in to show which sploit applies to which user, the type of sploit, and the contents of that user's `flag.txt` file.

user	sploit#	type	flag
alice			
bob			
carol			
david			
eve			

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

The first step in writing your exploit programs will be to identify vulnerabilities in the original `backup.c` source code, and then figuring out which of the patched `backup` executable binaries are still vulnerable. You should use techniques described in your readings to be sure that a binary still has that exploitable flaw before you begin writing and testing your exploit program.

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 passwd (man 5 passwd), execve (man 2 execve), su (man su), mkdir (man mkdir), and chdir (man chdir)
- SSH public key authentication (e.g., http://www.cs.uwaterloo.ca/cscf/howto/ssh/public_key/, ignore the PuTTY part for this assignment)
- Environment variables (e.g., http://en.wikipedia.org/wiki/Environment_variable).

Note that in the virtual machine you cannot create files that are owned by root in the `/share` directory. Similarly, you cannot run chown on files in this directory. (Think about why these limitations exist.)

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 `backup` application using the `execve()` (or a similar) function, the following statements will allow you to debug the `backup` application:

1. `gdb sploitX` (X=1..4)
2. `catch exec` (This will make the debugger stop as soon as the `execve()` function is reached)
3. `run` (Run the exploit program)
4. `symbol-file /home/username/backup` (We are now in the `backup` application, so we need to load its symbol table)
5. `break main` (Set a break-point in the `backup` application)

6. `cont` (Run to break-point)

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 `backup` will not run any program or command 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, 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` environment: `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 can retrieve your account credentials from the Infodist system.

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 is wisest to ssh twice into ugster. In one shell, start user-mode linux, and compile and execute your exploits. In the other account, log into ugster and edit your files directly in `~/uml/share/`, so as 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 your ugster environment should be asked on Piazza.