UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2017**
Florian Kerschbaum

ASSIGNMENT 2

Assignment due date: **Friday, March 3, 2017 3:00 pm**

Total marks: 70
Written Response Questions TA: Kshitij Jain (Office hours: Monday 1:30-2:30pm in DC2568)
Programming Question TA: Suman Nakshatri (Office hours: Friday (starting Feb 17th) 12-1pm in DC3332)

Please use Piazza for all communication. Ask a private question if necessary. You are expected to follow the expected Academic Integrity requirements for Assignments; you can find them here: `https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial` Strict penalties will be enforced on students for any Academic Integrity violations.

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. [Total: 8 marks] **Behavioural Authentication**
   Alice usually forgets to lock her tablet, so she has developed an application for her tablet that locks it if a finger swipe of the user does not match Alice's finger swipe pattern.[1] The false acceptance rate (FAR) of the application is 4% (i.e., the tablet incorrectly identifies a stranger's finger swipes as Alice's swipes 4% of the time). The false rejection rate (FRR) of the application is 8% (i.e., the tablet incorrectly identifies Alice's swipes as a stranger's swipes 8% of the time). Being rejected frequently, Alice decides to add a *window* of seven swipes. A rejected swipe increases the rejection counter by one. Her application locks the tablet as soon as the rejection counter reaches three, and the counter is set to zero after every seven swipes (the window is non-overlapping). Suppose Alice knows that her tablet is being used by others 9% of the time. (Give all answers in at least 3 decimal places)

   (a) [4 marks] What are the new values for the FAR and FRR of the application for each non-overlapping window (seven swipes)?

[1] This technology is called *implicit authentication*, and is currently an active topic of research.

New FAR: At most 2 rejections in 7 swipes by a stranger

$$\binom{7}{0}(0.96)^0(0.04)^7 + \binom{7}{1}(0.96)^1(0.04)^6 + \binom{7}{2}(0.96)^2(0.04)^5 \approx 2.01 \cdot 10^{-6}$$

New FRR: At least 3 rejections by Alice

$$1 - (\binom{7}{0}(0.08)^0(0.92)^7 + \binom{7}{1}(0.08)^1(0.92)^6 + \binom{7}{2}(0.08)^2(0.92)^5) = 0.01401$$

(b) [4 marks] If the tablet locks within the first 7 swipes, what is the probability that it is locked correctly (i.e., a stranger was using the tablet)?

Hint: You should use Bayes' Rule for conditional probabilities to solve this question: `http://en.wikipedia.org/wiki/Bayes'_theorem`

From the Bayes' rule we have: $Pr(Stranger|Reject) = \frac{Pr(Reject|Stranger)Pr(Stranger)}{Pr(Reject)}$. We already know that $Pr(Reject|Stranger) = 1 - Pr(Accept|Stranger) \approx 1$, $Pr(Stranger) = 0.09$, and $Pr(Alice) = 0.91$. We need to compute $Pr(Reject)$:

$$Pr(Reject) = Pr(Reject|Stranger)Pr(Stranger) + Pr(Reject|Alice)Pr(Alice)$$

$$\approx 1 \times 0.09 + 0.01401 \times 0.91 \approx 0.10275$$

Therefore $Pr(Stranger|Reject) \approx \frac{0.09}{0.10275} \approx 0.87591$.

2. [Total 10 Marks] **Bell-La Padula Confidentiality Model / Biba Integrity Model.**

Consider the following hierarchy of students and staff requiring access to course materials:

Student $\leq$ TA $\leq$ Professor $\leq$ Course Coordinator

(a) [5 marks] Let Alice be a faculty member with the following clearance: (Professor, {assignments, exams, marks}). Using the Bell-La Padula Confidentiality Model, state if Alice has read access, write access, both, or neither, for each of the following files:

- file1: (Professor, {slides, exams, marks})
- file2: (TA, {marks})
- file3: (Student, {slides, marks, evaluations})
- file4: (Course Coordinator, {assignments, exams, marks, evaluations})
- file5: (Professor, {assignments})

no read up
no write down

- file1: no access
- file2: read access
- file3: no access

- file4: write access
- file5: read access

(b) [5 marks] Bob has integrity level (Professor, {assignments, exams, marks}). Under the Dynamic Biba Integrity Model using the low watermark property described in the course notes, describe how the integrity level of Bob and each file changes after each operation if he performs the following in sequence:

i. Bob reads from record1 having integrity: (Professor, {feedback, exams, marks})
ii. Bob writes to record2 having integrity: (TA, {marks})
iii. Bob reads from record3 having integrity: (Student, {feedback, marks, evaluations})
iv. Bob writes to record4 having integrity: (Course Coordinator, {assignments, exams, marks, evaluations})
v. Bob writes to record5 having integrity: (Professor, {exams})

i. Bob's integrity becomes (Professor, {exams, marks})
ii. record2's integrity is unchanged
iii. Bob's integrity becomes (Student, {marks})
iv. record4's integrity becomes (Student, {marks})
v. record5's integrity becomes (Student, $\emptyset$)

3. [6 marks] **Password Hashing**
The task of designing a secure account database is assigned to Alice. You are provided with the following option. Find at least three problems with the design, explain them and give solution(s) to fix them.

i. For each password $\mathcal{P}$ entry, choose a 6-bit salt $\mathcal{S}$ at random.
ii. Compute $\mathcal{H}$ =SHA-512$(\mathcal{P}) + \mathcal{S}$.
iii. Store the user ID and $\mathcal{H}$ and $\mathcal{S}$ in the database.
iv. Whenever a user enters his/her ID and password $\tilde{\mathcal{P}}$, look for the ID in the database, and find the corresponding value $\mathcal{S}$. Verify user's authenticity if and only if $\mathcal{H}$ =SHA-512$(\tilde{\mathcal{P}}) + \mathcal{S}$

(a) Salt is too short, Mallory can compute the hash value for all the common passwords with different values of salts.

(b) Salt should be added to the password before hashing. In the suggested method of hashing, it is possible to get the hash value just by subtracting the salt.

(c) SHA-512 is too fast. They need to use a time and memory consuming hash, e.g. scrypt.

4. [6 marks] **Firewall**
There are different kinds of spoofing attacks, depending on who is spoofing what addresses to whom.

   (a) Suppose that in order to defend against spoofing attacks, UW sets up a packet filtering gateway that blocks all packets originating from *within* UW whose source address is not of the form 129.97.x.y. What kind of traffic spoofing attack does this rule protect against (2 marks)?
   >(2 marks) This prevents an attacker *within the network* from launching IP *spoofing attacks against external hosts*.

   (b) Suppose UW also creates a rule to block traffic originating from *outside* of UW whose source address is of the form 129.97.x.y. What kind of packet spoofing attack does this rule protect against (2 marks)?
   >(2 marks) This prevents an *outside attacker* from *spoofing the address of an internal host to another internal host*.

   (c) Give an example of spoofed attack that is still possible (2 marks).
   >(2 marks) An *outside attacker* can *spoof the address of a different external host*; an *inside attacker* can *spoof the address of a different internal host*.

Programming Question [40 marks]

Background

You are tasked with performing a security audit of a custom-developed *web application* for your organization. The web application is a content sharing portal, where registered and confirmed users can post, comment, and vote on articles. It is known that the content sharing portal was *very poorly written*, and that in the past, the web server has been compromised to allow users to log in without a password, or allowed clients to perform restricted actions such as voting as another user. As you are the only person in your organization to have a background in web security, you are tasked with examining the web application and *identifying any violations of certain security principles*, and *demonstrating that these are exploitable*, so that your organization may properly fix the vulnerabilities. Instead of having the full source code however, you are given nothing except outside access to the server. You are given the userid `alice` with the password `passw0rd` to try out the system.

Application Description

In *The Protection of Information in Computer Systems*, Section I.A., Saltzer and Schroeder present eight design principles for trusted systems. Unfortunately, many real systems still fail to follow the principles faithfully. The OWASP Top 10 (`https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`) lists the 10 most common/damaging web vulnerabilities, according to a survey of organizations and researchers in the field. The latest survey was conducted in 2013, while the 2016 call for data is currently open. Most of the top 10 are due to failing to follow the design principles. For this question, you are to examine a simple web application in order to identify violations of Saltzer and Schroeder's principles and/or examples of common vulnerabilities from the top 10.

You are asked to find at least five exploits violating the security principles. **Each exploit must perform a unique persistent[2] action by a different (non-`alice`) user of the portal.** For each exploit, you should do the following:

1. Write a program to exploit each vulnerability that you uncover (4 marks each).
2. Explain how each exploit works and how you were able to find it (2 marks each).
3. State exactly which one of Saltzer and Schroeder's eight design principles was violated, **or** which OWASP Top 10 vulnerability it represents and explain why (2 marks each).

[2]Here, "persistent" means that the effect of the exploit should be visible by a guest user visiting the portal.

There are at least six vulnerabilities in the web application:

1. SQL Injection : Can be used to impersonate any user.
2. Easy to guess password : nnasresf user had password 123456 which is commonly used.
3. Easy to crack password : Once you get the data.db file, you could then crack bob's password using rainbow tables.
4. Incomplete mediation : An attacker supplies a user ID of their choosing as a post param (uid) to comment action. Use it to impersonate nvolodin.
5. Old password file : This gives you sdnaksha's password.
6. Confirm hash : The URL for confirming accounts (confirm.php), takes one parameter - the confirm ?hash? value. User mmsabri is inactive and could be impersonated by passing an empty hash value.

Clues:

1. An HTML comment left by a developer with a TODO to remove the annoying files ending in the tilde character (as some text editors do) leads you to many tilde ending php files.
2. A post on the forum asks a question about whether using a robots.txt file is a good idea? Navigating to robots.txt from the web app root will lead you to a directory called ?docs?, which has the database file and oldpasswords file.

Rules for sploit execution

For your sploits, you should submit five tarballs (sploit[1-5].tar) each containing the following:

- **sploit.sh** - shell script to run your sploit
- any other files needed to run your sploits

Each script must accept **exactly one command-line argument**, which will be the **HTTP path** of the web application, e.g.

```
./sploit.sh ugsterXX.student.cs.uwaterloo.ca/userid
```

The scripts should *not* be within a directory in the tar file.

When run, each script should do the following:

1. compile the exploit code (if necessary)
2. execute the exploit code
3. output a message saying what it is doing

The purpose of your sploits should be to perform some action without the privilege to do so. In general you will need to impersonate a user in order to do this. For each vulnerability, there is a particular user account to be exploited, and a particular action that the user is capable of doing.

If we can't figure out how your sploit works, you will not earn any marks for it. If in doubt about how to present a sploit (or whether an exploit is acceptable), please send a message on Piazza to the programming TA or check during office hours.

What to hand in

Using the "submit" facility on the student.cs machines, hand in the following files:

a2.pdf A PDF file that contains your answers to all written response questions, plus parts 2 and 3 for each of your exploits.

sploit[1-5].tar (uncompressed) tar files containing your completed exploits for the programming question.

Note 1: You must include your name, your uWaterloo userid, and your student number at the top of the first page of a2.pdf. Failure to do so will result in a deduction of 5 marks from your final score on this assignment! Also, be sure to "embed all fonts" into your PDF files. Some students' files were unreadable in the past; if we can't read it, we can't mark it.

Note 2: We have installed a script on the ugsters that will let you check your submission format. Simply run the command `check` on the ugster from a directory with your prepared submission files. It will do some basic checks on the file names and formats, and output a detailed success or failure message.

Useful Information For Programming Sploits

A web server is running on each of the ugster machines, and a directory has been created using your ugster userid. Your copy of the web application can be found at:

`http://ugsterXX.student.cs.uwaterloo.ca/userid`

...where `ugsterXX` and `userid` are the machine and credential assigned to you previously (see Piazza).

If you need to reset the webserver, simply access the following URL:

`http://ugsterXX.student.cs.uwaterloo.ca/reset/userid`

This will delete all changes made to your copy of the web application, and restore it to its original state.

As with the previous assignment, the ugster machines are only accessible to other machines on campus. If you are working from home, you will need to first connect through another machine (such as `linux.student` or the campus VPN).

Note 1: If one of your exploits forces the application into an unusable state, you can restore the default state by accessing the URL above. If the web server itself becomes unusable, please report this on Piazza, along with a description of what you were doing when the problem occurred. *Do not perform denial of service attacks* on either the ugster machine or the web server.

Note 2: **Do not** connect to the web server with a userid different from the one that is assigned to you. Any student that is caught messing around with another student's web application by connecting to the wrong URL will receive an **automatic zero** on the assignment, and further penalties may be imposed.

Writing your exploits

You may use any language of your choosing (within reason) to exploit the server; the only restriction is that your exploits run correctly from the ugster machines. You may use external tools to aid you if they are not directly required for exploit execution (i.e. they help you gather information, etc.). If your programming language of choice is not currently supported on the ugsters, you can request that the appropriate packages be installed (as long as you do it well in advance of the assignment due date).

The tools **netcat** and **telnet** may be useful in writing and testing your exploits. These tools enable you to make network connections from a command line. There are plenty of resources on the Internet that explain how these tools can be used to interact with a web server via HTML GET commands; Google is your friend. A good starting point might be:

`http://www.stearns.org/doc/nc-intro.v0.9.html`

A basic understanding of HTML forms will also be helpful for completing this assignment; a good starting point for learning about forms might be:

`http://www.cs.tut.fi/~jkorpela/HTML3.2/5.25.html`

Here are some other links that you may find useful:

- cURL: `http://curl.haxx.se/docs/manpage.html` (information about headers, GET, POST)
- cURL Scripting: `http://curl.haxx.se/docs/httpscripting.html` (if you don't know how HTML forms work)
- Shebang: `http://en.wikipedia.org/wiki/Shebang_(Unix)` (if you are new to scripting)
- HTTP: `http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol` (if you are new to how HTTP/web works)
- Web sessions: `http://en.wikipedia.org/wiki/Session_(computer_science)` (go to the section on Web Server Session Management)