UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Fall 2017**
Urs Hengartner, Stefanie Roos

ASSIGNMENT 2

Assignment due date: **Monday, November 6th, 2017 3:00 pm**

Total Marks: 62
Written Response Questions TA: Jiayi Chen jiayi.chen@uwaterloo.ca
(Office hours: Thursday 15:00 pm–16:00 pm in DC 3333)
Programming Questions TA: Nikita Volodin nikita.volodin@uwaterloo.ca
(Office hours: Tuesday 11:00 am–12:00 pm in DC 3333)

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are also posted to Piazza for reference. 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. **Authentication** [Total: 10 marks]

   Alice's smartphone supports five different authentication methods to unlock the screen by default, namely PIN, password, unlock pattern (see course slide 3-59), fingerprint, and wearable device unlock[1]. To enhance security, Alice has additionally installed an application to enable touch-based implicit authentication that automatically locks the phone if a finger swipe of the user does not match Alice's finger swipe pattern. Please answer the following questions about authentication:

   (a) Which authentication factors are being used by these six authentication methods? Please refer to the course slides about authentication factors (3-33) and classify them into the correct categories. [3 marks]

   (b) Suppose that Alice chooses PIN, unlock pattern, and wearable device unlock for the screen lock. Any of these methods can be chosen to unlock the screen. Please list one potential threat for each method. [3 marks]

   (c) Suppose that Alice adopts touch-based implicit authentication. The false acceptance rate (FAR) of the application is 5% (i.e., the phone incorrectly identifies a stranger's finger swipes as Alice's swipes 5% of the time). The false rejection rate (FRR) of the application is 8% (i.e., the phone incorrectly identifies Alice's swipes as a stranger's swipes 8% of the time). Suppose Alice knows that her smartphone is being used by others 10% of the time. If the smartphone locks after only one swipe, what is the probability that the lock is a result of someone other than Alice using the phone? [2 marks] How can we (possibly) reduce the false rejection rate? [2 marks] (**Hint:** please refer to the Bayes' Theorem: `https://en.wikipedia.org/wiki/Bayes%27_theorem`)

2. **Security Policies** [Total: 10 marks]

   In a certain organization, all the documents and members are classified into three sensitivity levels satisfying the following hierarchy:

   $$\textbf{Confidential (C)} \leq \textbf{Secret (S)} \leq \textbf{Top Secret (TS)},$$

   where "≤" means "less sensitive than or as sensitive as". Each document/member is assigned to one or more of four compartments: Alpha, Beta, Gamma, Delta.

[1] As long as the wearable device is in proximity of Alice's smartphone, authentication is done automatically.

(a) Suppose that the organization follows the **Bell-La Padula confidentiality model**, and Alice is a member with clearance (Secret, {Alpha, Delta}). Please indicate whether she has read access, write access, or both to each of the following documents. [5 marks]

- D001: (Confidential, {Alpha, Gamma})
- D002: (Top Secret, {Alpha, Gamma, Delta})
- D003: (Secret, {Alpha, Delta})
- D004: (Secret, {Beta, Gamma, Delta})
- D005: (Confidential, {Delta})

(b) Suppose that the organization follows the **dynamic Biba integrity model** (i.e., satisfying both Subject and Object Low Watermark Properties), and there are four members with the following integrity classification:

- Alice: (Secret, {Alpha, Delta})
- Bob: (Confidential, {Alpha, Gamma, Delta})
- Carol: (Secret, {Alpha, Gamma})
- Eve: (Confidential, {Beta, Delta})

For the following operations in the given order, please state the change in the integrity classification of the subject and the object one by one. [5 marks]

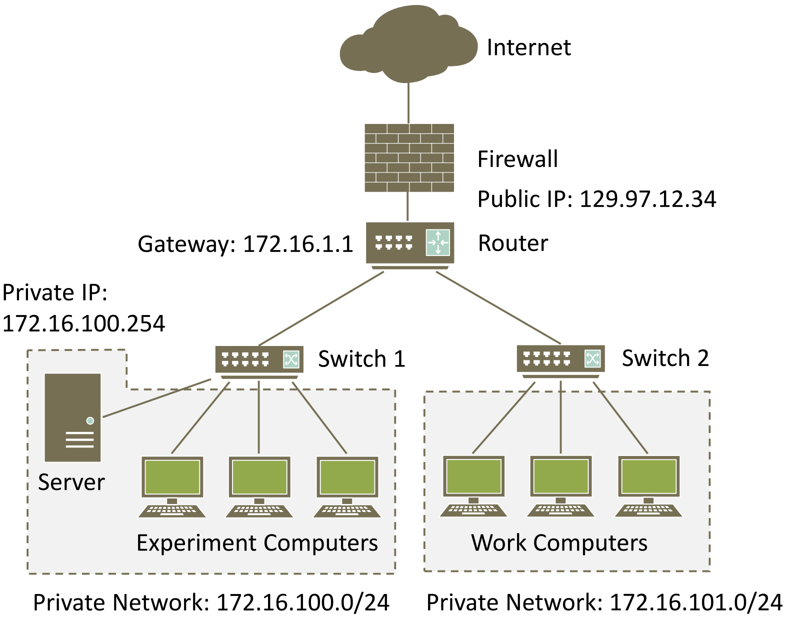
i. Alice writes to D101 (Top Secret, {Alpha, Gamma, Delta}).
ii. Alice reads from D102 (Confidential, {Alpha, Beta, Gamma}).
iii. Bob writes to D103 (Secret, {Beta, Delta}).
iv. Eve writes to D104 (Secret, {Alpha, Beta, Delta, Gamma})
v. Carol reads from D104 (after Eve has written to it).

3. **Firewall** [Total: 10 marks]

A uWaterloo research group has recently moved to a new laboratory with several computers and one server. Some of these computers are used to collect experimental data and upload it to the server. The other computers are used by researchers for work. You are the network security engineer who ensures the lab network security.

(a) Suppose that uWaterloo sets up a packet filtering gateway. What kind of spoofing attacks can it defend against? [2 marks]

(b) Suppose that you need to satisfy the following requirements: 1) to prevent hosts in external networks from initiating a connection to the experiment computers and the server, 2) to prevent all the devices in the lab from browsing malicious websites, and 3) to record all remote login operations from the external networks to the work computers. In addition to a packet filtering gateway, what kind of firewall do you need to deploy and how do they work together to satisfy these requirements? [2 marks]



(c) Suppose that the lab network topology is organized as shown in the figure.

Can the firewall control access from the work computers to the experiment computers and the server? Please justify your answer. [2 marks] Suppose that the packet filtering functionality is now part of the router, which is enabled by maintaining extended access control lists (ACLs)[2]. Use the table below to define a set of packet filtering rules on the interface connecting Router and Switch 1 that enable only the work computer users to connect to the server via SSH. Note that you can consider the following access control list as a white list. PROTOCOL field only accepts transport layer protocols. [4 marks] (**Hint:** please make rules with the given directions[3]. You can use "ANY" to represent all possible options. For example, "OUT,TCP,1.1.1.1,ANY,2.2.2.0/24,80" means permitting all outgoing TCP traffic from host 1.1.1.1 (any port) to subnet 2.2.2.0/24 (port 80).)

DIRECTION	PROTOCOL	SOURCE	PORT	DESTINATION	PORT
OUT					
IN					

[2] Extended ACLs can control traffic according to both source and destination addresses of the IP packets as well as more Layer-3 and Layer-4 information.

[3] OUT - from Router to Switch 1; IN - from Switch 1 to Router

Programming Response Questions [32 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 any user can view content, which includes articles, links, images, and comments. Registered users can log in and then post content. Note, users in the process of using the website can inadvertently leak information that may assist you in solving your tasks. You are provided black-box access to this application, that is, you can access the website in the same manner as a user.

The website uses PHP to generate HTML content dynamically on the server side. The server stores the application data, which includes usernames, passwords, comments, articles, links and images, in a SQLite3 database on the server. In this manner, we consider two systems, a relational database and HTML forms, that by default do not validate user inputs.

Questions

1. [20 marks] **User impersonation**

   (a) [4 marks] **Confirmation hash**

   One of the users on the website has been inactive for some time and now must confirm the username in order to be able to use the website.

   i. [3 marks] Find the user, their confirmation hash value and obtain access as that user. Save the server response from the hash confirmation request into the file `response.txt`.

   ii. [1 mark] Create a post on behalf of the user you just impersonated.

   What to submit: `exploit1_a.tar` file which contains the following files (**not** inside a folder):

   - `exploit.sh` — shell script performing the confirmation and logging in as the user.
   - `response.txt` — file with the response from the server after successful log in. It should contain the hash that you just sent in the URL.

   (b) [4 marks] **SQL Injection**

   User-generated data is directly passed into database queries in several contexts. For example, to insert or modify user-generated content in a database, or to query a database that contains user credentials for authentication. A user can craft their input to include malicious database queries, such as to alter queries or insert/modify records.

i. [3 marks] Use an SQL injection attack on the log-in form to obtain access as another user. Save the response from the log-in request into the file `response.txt`.

ii. [1 mark] Create a post on behalf of the user you just impersonated.

What to submit: `exploit1_b.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script with exploit.
- `response.txt` — file with the response from the server after successful log in, containing username and password.

(c) [4 marks] **Easily guessable password**

One of the users on the website has a password that is very easy to guess.

i. [3 marks] Guess the password and save the response from the log-in request into the file `response.txt`.

ii. [1 mark] Create a post on behalf of the user you just breached.

What to submit: `exploit1_c.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script that performs the login as the user with the weak password.
- `response.txt` — file with the response from the server after successful log in, containing username and password.

(d) [8 marks] **Incorrect server configuration**

As mentioned above, the website uses PHP to generate HTML content dynamically on the server side. This implies that the client browser is sending requests for the PHP scripts and the web server executes these requests on the fly. A common mistake of developers is to not restrict file access during the execution of these requests. As a consequence, any user of the website might access confidential files. For instance, the attacker might learn sensitive information about the content and structure of the database. Based on this information, the attacker might be able to mask the occurrence of the attack by manipulating the database.

i. [3 marks] Your task is to come up with an exploit script that obtains a dump of the website's database and saves it as `data.db` alongside the exploit file.

ii. [4 marks] By examining the structure of the database file from part 1(d)i, come up with an exploit that injects the new user into the database of the website. The new user should be indistinguishable from a real user registered on the website. Save the username and the password of the new user into the file `credentials.txt` and save the response from the server after issuing the request into the file `response.txt`.

Note, the solutions to parts 1a and 1b might give you hints necessary to solve this question.

iii. [1 mark] Create the post on behalf of the user you just created.

What to submit: `exploit1_d.tar` file which contains the following files (**not** inside a folder):

- `exploit.sh` — shell script injecting user into the database.
- `data.db` — database file downloaded from the server.
- `credentials.txt` — file containing the username and password of a new user. Include username on the first line and password on the second line, followed by the new line character and nothing else.
- `response.txt` — file with the response from the server after injecting the user into the database.

For the rest of the assignment, please use the following credentials:

```
username:  "alice"
password:  "passw0rd"
```

2. [6 marks] **Cross-site scripting attack**

Cross-site scripting attacks involve injecting malicious web script code (including HTML, Javascript) into a webpage via a form. As a result, the document object model (DOM) of the HTML webpage changes whenever the code is executed. Depending on whether the change persists across reloads of the webpage, XSS attacks are classified as stored (persistent) or non-persistent attacks.

As a tester, who wants to detect changes in the DOM of the HTML webpage, you may want to check the source code of the page and monitor the HTTP requests and responses. This can be done using standard 'Developer tools' menus in browsers and/or using adequate flags for command-line utilities like `curl`.

(a) [3 marks] Write a JavaScript function that first finds the ID of a post given the post content as parameter and upvotes the post.

(b) [3 marks] Create a new post, which contains the script we created in part 2a (it might be slightly modified), so that the post is automatically upvoted by every person who sees the post.

Refer to Section Notes about JavaScript for resources about JavaScript.

What to submit: `exploit2.tar` file with the following files (**not** inside a folder):

- `exploit.sh` — shell script that creates new post.
- `response.txt` — file with the response from the server after creating the post, containing the type, title, and post content.
- `upvoteScript.js` — file with the script as described in part 2a.

3. [6 marks] **Cross-site request forgery attack**

   Cross-site request forgery attacks manipulate the browser to send state information maintained by the client to the website without the knowledge of the user. Hence, they use the fact that there is no way to identify whether an HTTP request is 'genuinely' sent by the user or if the browser has been 'duped' into sending it.

   (a) [3 marks] URLs on the website are created using the html tag `<a>`, as follows: `<a href="[LINK]">[LINK TITLE]</a>` with [LINK] indicating the URL and the [LINK TITLE] indicating the title of the URL. Substitute the [LINK] in such a manner that the comment is created on the post with id "2" on behalf of the user who follows the [LINK].

   Note that clicks on `<a>` links result in GET requests by the browser, which might not be something that we would like in order to create a new post. There are some tricks how to override the behaviour of the `<a>` tag, for example, as in this post on stackoverflow.

   (b) [3 marks] Create a new post, which contains the link with the malicious URL as described in part 3a. Whenever a user clicks on that link, a comment is created for the post containing the malicious link in the name of the user.

   NOTE: The attack performed in part 3b is not a pure CSRF attack. Strictly speaking, the attack has elements of both XSS (including malicious code in the post) and CSRF (having the user's browser execute code without the user's intention). Normally, a CSRF attack requires the link to be embedded in a different website. However, to demonstrate the general idea of the attack within the limited scope of the assignment, we use the web portal as both the attack, i.e., the website responsible for the malicious behaviour, and the target website, i.e., the website that is affected by the user's unintended behaviour.

   Refer to Section Notes about JavaScript for resources about JavaScript.

   What to submit: `exploit3.tar` file with the following files (**not** inside a folder):

   - `exploit.sh` — shell script which creates the new post and follows the newly created link in a post.
   - `response.txt` — response from the server after creating the post, containing type, title and content of the post.
   - `url.txt` — URL as described in the part 3a.

Rules about exploit script execution

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 from the Infodist system. 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).

You need to submit exploit scripts. These are the tarball files with the name and contents specified in the text of the assignment. Tarballs may also contain other files that are required for `exploit.sh`. Submitted exploits have to be standalone, as the environment will be reset for each exploit execution. The exploit script `exploit.sh` must accept exactly **one** command-line argument, which will be the **URL address** of the web server (e.g. `'./exploit.sh http://ugsterXX.student.cs.uwaterloo.ca/userid'`). Failure to follow this rule will result in 1 mark being deducted for every time `exploit.sh` script accepts incorrect parameters.

Obviously, these are exploit files and can potentially steal confidential information. Therefore these scripts will be ran on the air-gapped system in the Faraday cage in the CrySP$^{\text{TM}}$ lab. It means the execution environment will **not** have any Internet access. It means that scripts injected via exploits can only rely on themselves and cannot rely on any resource served on the Internet.

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. Any student caught performing DoS attacks, will receive an **automatic zero** on the assignment, and further penalties may be imposed.

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.

Note 3: These scripts will be performing requests and saving responses of the server. You should save the *exact* response of the server. For example, if the server replies with the redirect message, then this is what should be logged (as there might be more information in the reply of the server). In particular, if you choose to complete the assignment using `curl` as your main tool, do **not** use the flag `-L`.

Attacks and protocols

The following links may be helpful in designing exploit scripts.

- Cross-site scripting attacks and defenses: https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29
- Cross-site request forgery and defenses: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29
- SQL Injection attacks and defenses: https://www.owasp.org/index.php/SQL_Injection
- HTTP: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol (if you are new to how HTTP/web works)
- JS: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Document Object Model: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (API for accessing parts/objects of an HTML page)

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 on 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 during programming, etc.).

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 or
https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms

For some of the requests, you might need to consult with
https://developer.mozilla.org/en-US/docs/Glossary/percent-encoding
in order to successfully deliver data to server over the URL.

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)
- Web sessions: http://en.wikipedia.org/wiki/Session_(computer_science) (go to the section on Web Server Session Management)
- Netcat: http://www.stearns.org/doc/nc-intro.v0.9.html — Making network connections from a command line

Notes about JavaScript

Here is a template for the parts where you need to come up with JavaScript code. This should help with the task, but it does not mean that your solution has to follow this template.

```
function fn(content) {
  var aTags = ...;
  var foundTag;
  for (var i = aTags.length - 1; i >= 0; i--) {
    if (aTags[i].textContent === content) {
      foundTag = aTags[i];
      break;
    }
  }
  var postID = ...;
  var request = ...;
  ...; // send the request
}
fn('...');
```

In order to fill those missing gaps in the template, you might consult with the following pages from reputable sources:

- document.querySelectorAll()
- RegExp
- XMLHttpRequest.send()

You can also test out your JavaScript source code in the browser's developer tools console: for chrome, for firefox.

What to hand in

Using the "submit" facility on the student.cs machines (**not** the ugster machines or the UML virtual environment), hand in the following files:

a2.pdf: A PDF file containing your answers for all written questions and programming questions when requested. It must contain, at the top of the first page, your name, UW userid, and student number. **-3 marks if it doesn't!** Be sure to "embed all fonts" into your PDF file. Some students' files were unreadable in the past; if we can't read it, we can't mark it.

exploit{1_{a,b,c,d},2,3}.tar: Tarballs containing your exploits for the programming portion of the assignment. To create the tarball, for example for the first question, `cd` into the directory containing your `exploit.sh`, and run the command

```
tar cvf exploit1_a.tar .
```

(including the `.`).