UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Spring 2023**
Diogo Barradas
Adithya Vadapalli

ASSIGNMENT 2

Milestone due date: **Wed, June 21, 2023 at *3:00 pm***
Assignment due date: **Wed, July 5, 2023 at *3:00 pm***

Total Marks: 83

Written Response TA: Solomon Levi Davidson
Office hours: Thursdays, 2:30pm–3:30pm, online on BBB

Programming TA: Shreya Arun Naik
Office hours: Wednesdays, 3:00pm–4:00pm, DC 2310

Instructors:

Diogo Barradas
Office hours: Tuesdays, 10:00am–11:00am, DC 2631

Adithya Vadapalli
Office hours: By appointment only

Please use Piazza for all communication. Ask a private question if necessary.

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 for the appropriate deadlines, using the command `submit cs458 2 .` (dot included). If you are submitting late for the final delivery (in the 48-hour grace period), you will require the "`-t`" (tardy) option to `submit`: `submit cs458 2 -t .` (including the dot). After submitting, you should verify the files you submitted are the ones you intended with the "`-p`" (print) option to `submit`.

1. **Milestone:**

   exploit1.tar Tarball containing your exploit for the first question of the programming assignment part. **Note that you will not be able to submit the tarball for the first exploit after this deadline.** To create the tarball, for example for the first question, which is developed in the directory `exploit1/`, run the command

   ```
   tar cvf exploit1.tar -C exploit1/ .
   ```

   (including the `.`). You can check that your tarball is formatted correctly by running "`tar tf file.tar`". For example (note that there are no folder names in the output):

   ```
   $ tar tf exploit1.tar
   ./
   ./exploit.sh
   ./response.txt
   ```

2. **Rest of assignment:**

   a2-responses.pdf: A PDF file containing your answers for all written questions. It must contain, at the top of the first page, your name, UW userid, and student number. **You will lose 3 marks if it doesn't!**

   Be sure to "embed all fonts" into your PDF file. The file must be a valid PDF file; renaming a txt file to pdf will be corrupt and unreadable. Some students' files were unreadable in the past; **if we can't read it, we can't mark it.**

   exploit{2,3,4,5,6}.tar: Tarballs containing your exploits for the rest of the programming portion of the assignment.

   To create the tarball from a directory that contains the exploit directories (`exploit1/`, `exploit2/`, ... `exploit6/`), run the command:

   ```
   for i in $(ls -d */); do tar cvf ${i%/}.tar -C $i .; done
   ```

Written Response Questions [37 Marks]

Q1: Access Control

Alex is a rookie student-athlete that was recruited to join the track and field team. Alex needs to read and/or write to certain documents as part of the registration process. Therefore, the system administrator has ensured that individuals can only access documents they intend to access by employing the **Bell-La Padula Confidentiality Model**. Below are the sensitivity levels:

$$\text{Coach} \geq_{dom} \text{Trainer} \geq_{dom} \text{Athlete} \geq_{dom} \text{Fan}$$

Alex has the following clearance level:

$$(\text{Athlete}, \{\text{medical}, \text{fees}, \text{policy}\})$$

a) [5 marks] Determine which of the following documents Alex has read and/or write access to.

1. Doc1: $(\text{Coach}, \{\text{medical}, \text{fees}\})$
2. Doc2: $(\text{Fan}, \{\text{policy}\})$
3. Doc3: $(\text{Athlete}, \{\text{fees}, \text{policy}\})$
4. Doc4: $(\text{Fan}, \{\text{ticket}, \text{fees}\})$
5. Doc5: $(\text{Athlete}, \{\text{staff}, \text{medical}, \text{fees}\})$

b) [5 marks] Barbara is a trainer for the track and field team. Her clearance level is as follows:

$$(\text{Trainer}, \{\text{medical}, \text{policy}\}) \tag{1}$$

She was recently given a task that requires her to read and modify some documents. State how Barbara's and the documents' clearance levels change over time with each read/write action, assuming that the **Dynamic Biba Integrity Model** with the low watermark is used.

1. Barbara reads from Doc1 that has integrity level: $(\text{Trainer}, \{\text{medical}\})$.

2. Barbara reads from Doc2 that has integrity level: $(\text{Athlete}, \{\text{medical}, \text{policy}\})$.
3. Barbara writes to Doc3 that has integrity level: $(\text{Coach}, \{\text{policy}, \text{fees}\})$.
4. Barbara writes to Doc4 that has integrity level: $(\text{Trainer}, \{\text{medical}\})$.
5. Barbara reads from Doc5 that has integrity level: $(\text{Fan}, \{\text{medical}, \text{policy}, \text{fees}\})$.

Q2: Password Security

You work at Take1010, where employees must log in using a username and password before starting work each day. All passwords are hashed using SHA256 and stored in an encrypted password file. The only rule is that passwords must be at least eight characters long. Unfortunately, you recently found out that the password file had been leaked, and a hacker was able to decrypt the file and determine some of the employees' passwords.

a) [1 mark] What type of attack was likely used to discover the employees passwords?

b) [3 marks] In order to prevent future attacks, Take1010 made it mandatory for employees to change their password every 2 months. Your friend Carlton is a good employee, so he follows Take1010's new policy. Below are Carlton's password hashes over the last 8 months:

   - `a1bc8df0756b9223ac9542de3ba2a782d6c5ae231308e35699410748238824f0`
   - `e5e6da5a6783628129c4e425e9fc056e06ba6d3b013f19926edc964f8288f81e`
   - `2a527f61126a131956893d50635f4934802c24388e1c737bf19257e9894bce04`
   - `2a9c648c8200d4d9d8ee2f0c5f0781ace1b601c236b9552578b5e675b5a38b41`

   Will this prevent similar attacks in the future? Explain why or why not using Carlton's hashes as examples. (You may use the internet to help with answering this question)

c) [2 marks] What are two improvements for how Take1010 can store passwords that will make them more difficult to crack? Explain why these are improvements when compared to the basic password hashing mechanism.

d) [3 marks] Take1010 now wants to use two-factor authentication along with passwords. They are trying to decide between having an additional pin, an SMS message or a magnetic key card as the second factor. Explain which of these three options is the most secure. What are the pitfalls of the other two options?

e) [3 marks] Why is SHA256 not the best algorithm to use when storing passwords? Without using salts, what are two ways to eliminate this problem?

Q3: Firewalls

As a security specialist, You are responsible for configuring a firewall to secure all the computers on Take1010's internal network. You are told that the firewall must satisfy the following conditions.

- Take1010 owns the IP range 18.28.148.0/24. (You don't need an explicit rule for this)
- The firewall's policy is "DENY by default". (You don't need an explicit rule for this)
- All employees should be able to browse the internet using both HTTP and HTTPS from within the network. These external web servers should not be able to establish a connection with internal computers.
- There is an external server where Take1010 publishes results. This server has IP address 20.140.130.178 and accepts TCP traffic only on port 55445. The service running on this external server expects traffic only originating from Take1010 on port 55445. Connections should not be initiated by this external server.
- The firewall must block all data coming from IP address range 48.67.66.128/25, as this is known to be an abusive range.
- Dave is an employee that works from home occasionally. He must be able to ssh to servers running on Take1010's network. His laptop's IP address is 123.22.34.23.
- Take1010 has a web server that runs only over HTTPS. The web server has the IP address 18.28.148.13, and must be accessible from anywhere in the world.
- The organization exclusively does DNS lookups through a DNS server with IP address 86.243.82.76. The DNS server only listens for DNS requests on port 53 and expects requests to come from port range 1300 through 1500. Assume DNS goes over UDP.

a) [12 marks] Configure the firewall by adding the required rules to meet the aforementioned requirements of Take1010. Rules must include the following:

   - DENY or ALLOW
   - Source IP Address(es)
   - Destination IP Address(es)
   - Source Port(s)
   - Destination Port(s)

- TCP or UDP or BOTH
- For TCP, a set of TCP flags that must be set (SYN and/or ACK). Note not all TCP rules may require these flags

Below is an example for how a server at 1.1.1.1 can create a new HTTP connection with computers on Take1010's network. The external server cannot create new connections.

ALLOW 1.1.1.1 => 18.28.148.0/24 FROM PORT 80 to all BY TCP ACK

Use $\{1, 2, 3, ...\}$ to denote a set of ports and use [1-10] to denote an inclusive range of ports.

b) [3 marks] You notice that it may not be safe to store the organization's web server inside the firewall. An attacker might be able to exploit faulty software running the web server to gain access to the rest of the network. What technique could you use to still expose the web server while keeping the rest of the network secure? How does this change the firewall's configuration?

Programming Response Questions [46 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 that users in the process of using the website can inadvertently leak information that may assist you in solving your tasks. You have been provided black-box access to this application, that is, you can only 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 an SQLite3 database on the server. There are two systems, a relational database and HTML forms, that by default do not validate user inputs.

You will be writing exploits that exploit various vulnerabilities in these underlying systems. The Background Knowledge section contains references for all background knowledge that is necessary to develop your exploits.

Web Application Setup

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://ugster50X.student.cs.uwaterloo.ca/userid
```

where `ugster50X` and `userid` are the machine and credential assigned to you previously from the Infodist system.

- **Connecting to the ugsters:** As with the previous assignment, the ugster machines are only accessible to other machines on campus. Since many of you may choose to work off-campus, you will need to first connect to the ugsters through another machine (such as `linux.student.cs` or the campus VPN). You can use SSH Tunnelling to forward website access to your local machine:

  ```
  ssh -L 8080:ugster50X:80 linux.student.cs.uwaterloo.ca
  ```

  This allows you to access your instance of the website at `http://localhost:8080/userid`.
- **Resetting the web app:** If you need to reset the webserver to its original state, such as to

debug question 3 or to remove all new posts, comments and votes, simply access this URL:

`http://ugster50X.student.cs.uwaterloo.ca/reset/userid`

- **Do not** connect to the web server with a userid different from the one that is assigned to you.
- **Do not** perform denial of service attacks on either the ugster machine or the web server.
- Please avoid uploading large files to the web server as it has limited disk space. Any student caught interfering with another student's application or performing DoS attacks in any manner, will receive an **automatic zero** on the assignment, and further penalties may be imposed.

Writing Your Exploits

We recommend using the command-line utility `curl`,[1] as you can solve all questions with just tweaks in command-line options and input arguments, with no need to interface to HTTP-request processing libraries. However, you may use any language that is installed on the ugster machines to write your code. The only restriction is that your scripts should run correctly on the ugster machines in order to exploit the server, *without installing additional dependencies*. You may use external tools to help you gather information during programming, etc. Note that when tested, your scripts will not have access to further download and run any code.

Assignment Submission Rules

Each question lists **required** files. You can submit more files if needed. For instance, if you use anything other than a command-line utility (e.g., Python), you will need to include your source-code files in that language.

We have specified names for each required file; these names must be used. Submitting files with different names will result in a penalty.

All questions require a script file `exploit.sh` and a text file `response.txt` to be submitted.

(a) `exploit.sh` — Our marking scripts will execute this file to mark the corresponding question. If you are using `curl`, then your script file `exploit.sh` for each question should invoke `curl` directly, passing the arguments supplied to the script.

If you are using an interpreted language, you should include your source code files in that language in your exploit tarball, and your script file should invoke the interpreter on those files, taking care to pass the URL argument.

If you are using a compiled language, first you must ensure that you use a compiler that is already installed on the ugster machines. You should include your source code files in that

[1]Note that the versions of `curl` on the linux.cs student machines and the ugsters are different; you should ensure that in your script, you run the version of `curl` that is on the ugster machine.

language in your exploit tarball, and your script file should take care of compiling the source code and generating the executable file. Additionally, it should also take care to invoke the executable file while passing the arguments.

For all questions, your `exploit.sh` has to accept **one** parameter — a URL of the website under attack, as described here. While you are working on the assignment, the URL under attack will be:

```
http://ugster50X.student.cs.uwaterloo.ca/userid
```

However, during marking it will be something else (but it will be a valid URL). Also note that there will be **no** trailing slash at the end of the URL.

You must not hardcode the URL of your ugster machine anywhere in your exploit files (including in HTML/JavaScript). Hardcoding will result in a penalty, if your submission can be graded at all.

For example, if you are using `python`, your source-code file named `exploit.py` can be invoked like this by your `exploit.sh` script as:

```
python3 exploit.py $1
```

Refer to the Background Knowledge section for help on languages and scripting.

(b) `response.txt` — Your exploit should output the HTTP responses when *you* tested your code into this file. In case your script does not work on the ugsters in our marking setup, we may use your submitted file, along with the main script and the regenerated file, to determine what went wrong and give partial marks. It should contain the HTTP response headers for all HTTP requests that are sent in the exploit. Refer to the Background Knowledge section for background on all of these terms.

Following is a sample *response.txt* file generated for Question 1a:

```
HTTP/1.1 302 Found
Date: Wed, 07 Jun 2023 02:54:54 GMT
Server: Apache/2.4.38 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Set-Cookie: CS458=oi5ut12s4unqub6df3vv1ntpo5; path=/
Location: index.php
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8
```

For each part of each question, submit your files as an *uncompressed* tar file with the files at the top level (do **not** submit your files inside a folder). Submitting files inside a folder will result in a penalty. To create the tarballs correctly, refer to the instructions at the start of the assignment under What to Hand In.

Question dependencies: Question 1 is required to solve all questions. (All questions require some combination of logging in as a user or impersonating one and then either creating a post or a comment or an upvote.) None of the questions require that you have solved a previous question other than Question 1; therefore, if you struggle with one of these questions, proceed to the next one. However, a given question may require previous parts of the same question to be solved first.

Questions

1. [Total: 6 marks] **Get, Set, Go!**
   This question is designed to help you write setup code that is essential for solving the other questions. To solve this question, you should experiment with your site and examine the form fields sent in GET or POST requests for logging in, creating a post, commenting on a post, and voting. **You should go through the Background Knowledge section before attempting to solve this question.**

   For this question, please use the following credentials:

   ```
   username:  alice
   password:  TheEasyPassword
   ```

   Write a *single* script named `exploit.sh`, that takes in the URL of a copy of the web application as an argument and performs the following tasks in the given order:

   (a) (1 mark) logs in as the user `alice` with the above password.

   (b) (1 mark) creates an *article* post as that user.

   (c) (2 marks) creates a comment on the post made in part 1b.

   (d) (2 marks) upvotes the post made in part 1b.

   Note that you will only get half marks for each of 1c and 1d if you comment or upvote on any post other than the one made in 1b. You should test your script with the input URL being the URL for your copy of the web application, that is, `http://ugster50X.student.cs.uwaterloo.ca/userid`. Note that no ending forward slash will be supplied in the input URL. We will be testing your script from a different, but valid URL, without the ending forward slash. Your script should concatenate the HTTP response headers corresponding to the HTTP requests for each of the above parts into a single file named `response.txt`.
   What to submit: `exploit1.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

User Impersonation

Your organization has heard of the dangers of allowing arbitrary internet visitors to sign up to their website, and thus they have now removed the signing-up feature for this website. However, an arbitrary visitor to this website could still *impersonate* an existing user. The next three questions examine different ways that a visitor could do this.

2. [Total: 4 marks] **Easily Guessable Password**
The site registration process did not enforce any password hardness measures. You are concerned that some user may have a password that is very easy to guess or discover. If this is the case, any other user could log in and post as them! Check out the website and identify the user with the weak password. Note the following:

   - The username-password pair used in question 1, is not the correct answer (`alice` and `TheEasyPassword`).
   - The password is easy to guess *by human* users who are exploring the website, so simply using rainbow tables, which are generic, or brute-forcing is not the correct approach here.

   Once you've guessed the password, write a *single* script that does the following:

   (a) (2 marks) Logs in to the website as that user. Saves the response from the login request into a file named `response.txt`. This text file should contain the username and the password.

   (b) (2 marks) Creates a post on behalf of that user.

   What to submit: `exploit2.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

3. [Total: 6 marks] **Confirmation Code**
One of the users on the website has been inactive for some time and now must confirm their account using a confirmation code in order to be able to use the website. This confirmation code should be pseudorandom, however, the website developers may have used deterministic inputs to compute the confirmation code for any user. Note that a confirmation code can only be used once, so while debugging this question, you should reset your website after each attempt.

   Identify an inactive user by examining the website. Write a script that:

   (a) (2 marks) Includes a simple function that computes the confirmation code for any user.

   (b) (2 marks) Logs in as the inactive user, using the confirmation code generated by running your function in part 3a for that user. Save the response from the code confirmation request into a `response.txt` file.

   (c) (2 marks) Creates a post on behalf of the user you just impersonated.

What to Submit: `exploit3.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

4. [Total: 6 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. Refer to the Background Knowledge section for necessary background and examples on SQL and SQL injection attacks.

   Write a script that:

   (a) (4 marks) Uses an SQL injection attack on the login form to log in as the user "`sanaik`".
   Saves the response from the login request into the file `response.txt`. Note that you will only receive half of the total marks for this question if you log in as another user. You must log in as this user to receive full marks.

   (b) (2 marks) Creates a post on behalf of the "`sanaik`" user.

   What to submit: `exploit4.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

5. [Total: 10 marks] **Cross-Site Scripting Attack**

   For this question, please use the following credentials:

   ```
   username:  alice
   password:  TheEasyPassword
   ```

   Cross-site scripting attacks involve injecting malicious web script code (known as the "payload") (including HTML, Javascript, etc) 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 changed DOM persists across reloads of the webpage, XSS attacks are classified as persistent or non-persistent attacks.

   In parts 5a and 5b, you will be writing Javascript functions that modify the webpage (that is, affect the *integrity* of the webpage) and log the user's presence on that webpage (break *confidentiality* of the client-side cookie) respectively. These functions will then form the "payload", by being included in a post, such that all users clicking on the post will be victims of two XSS attacks. Refer to the resources for Question 5 in the Background Knowledge section for necessary background, examples, and sample code for XSS attacks.

   (a) (3 marks) *Non-persistent XSS:* Write a Javascript function in a Javascript file named `xss.js`, that first checks if it is running on a page with the URL `view.php?id=<any post id>`. Otherwise, it should return immediately without doing anything. If the function is running on a `view.php` page, then it should modify the webpage HTML (not the database) so that it appears as if the post was written by the user “victim”. That is, the function should replace the webpage’s text “posted by __” with “posted by victim”.

(b) (4 marks) *Persistent XSS:* You will write two Javascript functions, that when chained together, will allow you to leak the user’s cookies in a comment on any page with the URL `view.php?id=<any post id>`. (Note that again, these functions should do nothing on pages that do not have this URL and all functions should be included in the `xss.js` file.)

   i. (1 mark) Write a JavaScript function `getID` that returns the integer ID of the current post when run on any page with the above URL.

   ii. (3 marks) Write a JavaScript function `leakCookies` that accepts a post ID `postID`, and leaves a *comment* on the `view.php?id=postID` page that contains *all* cookies for this website.

   Your functions, when composed like this: `leakCookies(getID())` should leave the user’s cookies as a comment on a page with the URL `view.php?id=<any post id>`.

(c) (3 marks) Write a shell script that logs in as the user `alice` and creates a new post that contains and runs the JavaScript functions from parts 5a and 5b. When a user views this post on a page with the URL `view.php?id=<new post id>`, it should appear as if it was written by “victim” and the user’s browser should automatically leave a comment with all of their cookies for this site.

What to submit: `exploit5.tar` file with the following files (**not** inside a folder):

- `xss.js` — File with the Javascript functions as described in parts 5a and 5b. Please keep the code in this file readable.
- `exploit.sh` — Shell script that creates the new post as described in 5c, using the code from `xss.js`.
- `response.txt` — File with concatenation of all HTTP response headers from the server.

6. [Total: 14 marks] **Cross-Site Request Forgery Attack**
Consider the following attack: the user `alice` has logged in to our web portal in one tab on her browser and in another tab, she views a promotion email that contains a link to the portal. This link can upvote content or create many posts on behalf of Alice, without her knowledge. If Alice clicks on a link for the portal while being logged in to it, then the CS458 cookies which were used for authenticating her to the client would also be sent to the portal. This is a cross-site request forgery (CSRF) attack. Here, the attacker exploits the fact that state information maintained by the user’s client (browser) for a given domain is sent with subsequent requests to that domain, *even* if the user did not intend to send the request.

As we do not currently host other means of delivering the link, e.g. through emails, we will host these links on the web portal itself. (That is, we use the web portal as both the attacker-controlled site, which hosts the malicious links, and the target website, which is affected by the user's unintended behaviour.) Specifically, in the following questions, we host links that *execute code* on the web portal, by exploiting stored XSS vulnerabilities, which we explored in question 5b. URLs on the website are created using the html tag `<a>`, as follows: `<a href="[URL]">[TITLE]</a>`. For all parts of this question, you can assume that your `<a>` tags will be clicked/executed from the main `/index.php` page of the website by a logged-in user.

Refer to the resources for Questions q:XSS and q:CSRF in the Background Knowledge section for necessary background, examples and sample code for CSRF attacks.

(a) (4 marks) Write a script that creates a post, such that whenever a user clicks on the `<a>` tag, the post with id "4" is upvoted by 700 points on behalf of that user. **You should not use any JavaScript for this question.** Essentially, you will be replacing the **[URL]** with HTML code that performs upvoting. Include the HTML code for the `<a>` tag in a separate file named `upvote_url.html`.

(b) (4 marks) Similarly to 6a, create a post, such that whenever a user clicks on the link, a *new* post is created on behalf of that user. **You may use JavaScript for this question.** As before, include the HTML (and any Javascript) code for the link in a separate file named `post_url.html`; for this question, also include any Javascript code that's in this HTML file, in a human-readable format (with proper formatting) in a file named `post_human.js`.

Note that clicks on `<a>` links result in GET requests by the browser, which might not work for creating a post. There are some tricks how to override this behaviour of the `<a>` tag, for example, as in this post on stackoverflow.

(c) (6 marks) Create a new post, which contains the `<a>` link with the malicious URL similar to the one in part 6b. **You may use JavaScript for this question.** Construct it such that whenever a user clicks on the link, a new post that has the *same link* is created on behalf of the user who clicked on the link. So, the link in the new post should create another new post, which has a link which will create another new post, which has a link... and so on, recursively.

As before, include the HTML (and any Javascript) code for the link in a separate file named `recursive_post_url.html` and also include any Javascript code that's in this HTML file, in a human-readable format in a file named `recursive_post_human.js`.

Go through the resources for Questions 5 and 6 in the Background Knowledge section for help.

What to submit: `exploit6.tar` file with the following files (**not** inside a folder):

- `upvote_url.html` — `<a>` tag with [URL] substituted for 6a.
- `post_url.html` — `<a>` tag with [URL] substituted for 6b.

- `post_human.js` — A human-readable version of the JavaScript code contained in the [URL] section of the `<a>` tag in 6b.
- `recursive_post_url.html` — `<a>` tag with [URL] substituted for 6b.
- `recursive_post_human.js` — A human-readable version of the JavaScript code contained in the [URL] section of the `<a>` tag in 6c.
- `exploit.sh` — Shell script that creates three new posts, ones for each of the three parts, using the HTML files.
- `response.txt` — File with concatenation of all HTTP response headers from the server.

Background Knowledge

- Question 1: You must familiarize yourself with HTTP request methods (GET and POST for our application) and sending data through the `<form>` element. Refer to the HTTP status response codes while debugging this question. Go through the first section on MDN Cookies to understand why cookies are necessary and how they work. You should figure out how to access the cookies set by the website, through your language of choice (e.g., for curl), in order to conduct stateful transactions (questions 1b, 1c and 1d) once you have logged in as a user. In general, understanding HTTP 1.1 messages will be helpful for this assignment (e.g., for generating the correct `response.txt` file).

- Question 3 and onwards: For some requests, you will need to understand URL encodings in order to successfully deliver data to the server over the URL.

- Question 4: If you are unfamiliar with relational databases or SQL syntax, you can refer to the W3Schools site or the Tutorialspoint. You will need to understand how the following SQL clauses work: SELECT, FROM, WHERE and INSERT. Once you are familiar with this syntax, refer to the SQL injection examples on the OWASP SQL site.

- Question 5 and 6 : For step-by-step examples with sample code of non-persistent and persistent attacks, go through the short “Exploit examples” section on the OWASP XSS site. Similarly, for simple step-by-step examples of CSRF attacks for the GET and POST HTTP request methods, read the “Examples” section on the OWASP CSRF site. Note that these examples and sample code are intended to provide you with some background in order to develop your own code. To develop Javascript code for this question, you should have a basic understanding of the following syntax:

  – In order to send HTTP requests through Javascript, you may use the XMLHTTPRequests API or the more recent Fetch API.

    You can specify URLs in the XMLHttpRequest API using either the absolute URL or the relative URL (difference between the two). Since the URL that you are using to test

your exploits and the URL that we will use for marking will be different, you may not assume that a request to
`http://ugsterXX.student.cs.uwaterloo.ca/userid/index.php`
will always do what you expect. Therefore, you should use relative URLs, so that you do not have issues where you do not know the rest of URL.

– The Document Object Model (DOM) is used by the browser to process HTML documents. You will need to interact with the DOM for some parts of these questions. The following DOM features may be useful: document.querySelectorAll():, document.cookie, document.location, and outerHTML.

– You may find Javascript regular expressions and string manipulation methods helpful for processing URLs.

You can easily test your JS code in chrome's or firefox's developer tools console.