UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2023**

Yousra Aafer
Meng Xu

ASSIGNMENT 2

Milestone due date: **Friday, March 03, 2023 at *3:00 pm* - MANDATORY**
Assignment due date: **Friday, March 17, 2023 at *3:00 pm***

Total Marks: 85
Bonus Marks: 3
Written Response TA: Vasisht Duddu
Office hours: Fridays 1:00pm–2:00pm

Programming TA: Parjanya Vyas
Office hours: Fridays 1:00pm–2:00pm

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). 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{2,3,4,5,6,7}.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/`, ... `exploit7/`), run the command:

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

Written Response Questions [33 marks]

Q1: Access Control

ECorp is one of the largest multi-national conglomerates in the world. Among their enterprises, they manufacture computers, phones, and tablets, and maintain a banking and consumer credit division. This makes it critical to protect their infrastructure from malicious hacker groups such as FSociety. AllSafe Cybersecurity Technologies has been hired to conduct a variety of security tests and protect ECorp's infrastructure.

You as an employee of ECorp have been hired to conduct a security analysis to identify potential vulnerabilities.

Specifically, you are informed that ECorp uses a Bell-LaPadula security model with the following sensitivity/clearance levels:

$$\text{Top Secret} >_c \text{ Secret} >_c \text{ Classified} >_c \text{ Unclassified}$$

and the following compartments:

$$\text{Finance}, \text{ Human Resources}, \text{ Operations}, \text{ Public Relation}, \text{ Legal}$$

During your security analysis, you know the existence about two files: (a) `scandal.txt` must be given sensitivity level Top Secret and restricted to the compartments {Human Resources, Operations, Legal}; and (b) `audit.txt` has sensitivity level Secret and compartments {Human Resources, Operations}. `scandal.txt` will be a very sensitive document, and ECorp wants to ensure that only the most privileged administrative employees can read it.

In addition to those two documents, you are able to find a list of ECorp's employees from an internal memo hosted on an unsecured S3 bucket. You want to use this internal memo to launch a highly targeted spear phishing attack to compromise a ECorp employee's account. You only have the resources to carefully craft one such attack, so you must choose your target carefully.

1. [6 marks] You want to know whether each employee can:

   (a) read `audit.txt` (Secret, {Human Resources, Operations})

   (b) write `scandal.txt` (Top Secret, {Human Resources, Operations, Legal})

   (c) both

   (d) neither

   From those 4 options, indicate each employee's abilities:

(i) Susan Jacobs: (Secret, {Human Resources, Legal})

(ii) Terry Colby: (Top Secret, {Finance, Human Resources, Operations, Public Relation, Legal})

(iii) Scott Knowles: (Classified, {Human Resources, Operations, Legal})

(iv) Saul Weinberg: (Unclassified, $\emptyset$)

(v) Phillip Price: (Secret, {Human Resources, Operations})

(vi) Tyrell Wellick: (Unclassified, {Finance, Public Relation})

2. [6 marks] After careful consideration, you decide to target Phillip Price. Your phishing attack succeeds, and you now have access to Phillip Price's computer at ECorp! You quickly find and exfiltrate the `audit.txt` file, but you decide to look around some more in case there are other interesting things to be found.

   Phillip Price has access to an internal system which instead uses the dynamic Biba Model with the low watermark property. This system uses the same compartments but the following integrity levels:

   $$\text{High Integrity} >_c \text{ Medium Integrity} >_c \text{ Low Integrity} >_c \text{ Untrusted}$$

   Acting as Phillip Price: (Medium Integrity, {Human Resources, Operations}), you want to read and modify a few files. However, as you perform these actions in sequence, the integrity levels of Phillip Price and the files may change. **Note: If a change occurs, the modified integrity level is used for the subsequent operations.**

   Indicate:

   (i) The integrity level of Phillip Price after this action (including the compartments and how the compartments change)

   (ii) The integrity level of the file after this action (including the compartments and how the compartments change)

   for each of the following actions performed in order. (Please indicate "no change" if there is no change.)

   (a) Write to File 1 with integrity: (Medium Integrity, {Human Resources})

   (b) Read from File 2 with integrity: (High Integrity, {Human Resources, Operations})

   (c) Write to File 3 with integrity: (Medium Integrity, {Human Resources, Operations})

   (d) Write to File 2

   (e) Read from File 4 with integrity: (Untrusted, {Operations})

   (f) Write to File 5 with integrity: (Low Integrity, {Public Relation})

Q2: Password Security

Now, you want to evaluate the security of the company's password authentication mechanism. Its current scheme stores a password hash (fingerprint) file, and authenticates their employee login attempts against this fingerprint. The process for creating the fingerprint and verifying login attempts is as follows:

- Every password entry $\mathcal{P}$ maintains an 8-bit random salt $\mathcal{S}$ used for generating a fingerprint $\mathcal{F}$. They use hash function $\mathcal{H}$ for this system
- The fingerprint of a password is computed as $\mathcal{F} = \mathcal{H}(\mathcal{P}) \oplus \mathcal{S}$ and is stored in their password fingerprint file along with the username and $\mathcal{S}$ for that user, where $\oplus$ is the bitwise XOR operator.
- When an employee attempts to log in with a password $\mathcal{P}'$, the system verifies the password by computing $\mathcal{H}(\mathcal{P}') \oplus \mathcal{S}$ where $\mathcal{S}$ is the salt for that user in their password fingerprint file.

You are tasked with answering the following questions when writing your report on the security of this password mechanisms.

1. [3 marks] Is this mechanism secure? Why or why not (describe a potential attack if not secure)?
2. [2 marks] Without changing the underlying hashing function, what two ways can the company improve their authentication scheme?
3. [2 marks] You are informed that they are using an implementation of SHA-512 algorithm to hash the passwords. Do you think this is the right choice? Explain the reasoning behind your answer, and provide and alternative function and justification if you don't believe this is the correct choice.
4. [2 mark] Phillip Price read a book on computer security in 1994, and wants to change the hash function used for storing passwords. Here is an example password hash he provides, which he believes was computed with an unbreakable hash function:

   2034f6e32958647fdff75d265b455ebf

   (a) Name a hash function that could have produced this hash.

   (b) What is a password that hashes to this value? (You may use the internet, but please don't share it with your classmates.)

   (c) Is this a good suggestion for a password hashing function?

Q3: Firewalls

During the security analysis of ECorp's systems, you accidentally mess up the firewall configuration. You have to recreate all the firewall rules based on the requirements below. (For this question, you can ignore the order of the rules.)

- ECorp owns the IP range 17.34.152.0/24. (You don't need an explicit rule for this)
- AllSafe Cybersecurity uses a "DENY by default" firewall policy. (You don't need an explicit rule for this)
- All employees of AllSafe Cybersecurity should be able to browse the internet (both HTTP and HTTPS pages) from within the network.
- AllSafe Cybersecurity hosts a webserver at 84.56.32.48. This server runs over HTTP and HTTPS and needs to be accessible from anywhere in the world.
- Finally, ECorp hosts an IRC server with IP 17.34.152.37 on port 1337. AllSafe Cybersecurity needs to be able to connect to this IRC server with address 243.82.77.16.
- Scott Knowles does much of their work remotely and needs to be able to ssh into their work device from anywhere in the world. They work device has the IP address 243.132.51.32.
- ECorp blocks all incoming traffic from the IP address range 64.56.91.0/24, as this range is known for abusive behavior from FSociety.
- ECorp has shifted to exclusively doing DNS lookups through a DNS server. This DNS server is hosted on IP address 243.82.76.43. This server only listens for DNS requests on port 53 and expects clients to send requests only from ports 1200 through 1550. Assume DNS goes over UDP.

1. [12 marks] Configure the firewall by adding the required rules to meet the aforementioned requirements of ECorp. Rules must include the following:
    - DROP 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

Here is an example rule that allows a server at 5.5.5.5 to serve HTTP pages to ECorp's network but prevents it from creating new connections:

ALLOW 5.5.5.5 => 17.34.152.0/24 FROM PORT 80 TO all BY TCP ACK

For multiple ports 80 and 443:

ALLOW 5.5.5.5 => 17.34.152.0/24 FROM PORT all TO {80,443} BY TCP

For a range of ports "1700-1750":

ALLOW 5.5.5.5 => 17.34.152.0/24 FROM PORT all TO [1700-1750] BY TCP

For connection to any IP address:

ALLOW 5.5.5.5 => 0.0.0.0/0 FROM PORT all TO [1700-1750] BY TCP

HINTS:

- CIDR Notation may be helpful for this portion of the assignment.
- Some requirements may need more than one rule.
- Ports can be specified as a singular value, range, as a set, or as 'all' as seen in the example above.
- Section 10.1 of of the van Oorschot textbook may be helpful in this task.

Programming Response Questions [52 marks + 3 bonus 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 will be working off-campus this term, 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 2 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

1. 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.
2. We have specified names for each required file; these names must be used. Submitting files with different names will result in a penalty.
3. 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

[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.

files in that 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, EXCEPT Question 6, your `exploit.sh` has to accept **one** parameter — a URL of the website under attack. (Question 6 accepts two parameters as described in the question description, the first of which is a URL 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 pointing to an exact replica of the freshly reset website that you currently see). 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. **Note that, HTTP request headers do not have a guaranteed order. Therefore, the exact order of the headers might be different in the marking server than what you see currently in your test website. So your solution scripts should NOT rely on a particular header being present at a particular location in the HTTP response.** 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: Fri, 07 Oct 2022 01:00:21 GMT
Server: Apache/2.4.29 (Ubuntu)
Set-Cookie: CS458=5krmh0r188rtbsq5m0knvu5ojn; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
```

```
Location: index.php
Content-Length: 19
Content-Type: text/html; charset=UTF-8
```

4. 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, except Question 6. (All questions except that one 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.

Bonus Question: Question 7 is a bonus question (worth 3 bonus marks). It is not mandatory to solve – i.e., you could get a full grade in the assignment without solving it.

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:  Password123
   ```

   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.

2. [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 2a 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: `exploit2.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

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

For this question, please use the following credentials:

```
username:  alice
password:  Password123
```

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 3a and 3b, 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 3 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 "mallory". That is, the function should replace the webpage's text "posted by __" with "posted by mallory".

(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 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 3a and 3b. 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 "mallory" and the user's browser should automatically leave a comment with all of their cookies for this site.

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

- `xss.js` — File with the Javascript functions as described in parts 3a and 3b. Please keep the code in this file readable.
- `exploit.sh` — Shell script that creates the new post as described in 3c, using the code from `xss.js`.
- `response.txt` — File with concatenation of all HTTP response headers from the server.

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 "`parjanya`".
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 "`parjanya`" 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: 14 marks] **Advanced SQL and Path Traversal**

A web application often hosts many files that are not webpages (that is, in PHP or JS) and are intended to remain confidential. Website developers often forget to configure the web server to restrict access to these files during the processing of HTTP requests. As a consequence, any visitor to the website might access confidential files and directories; this is known as a path traversal attack.

In this question, you will conduct a simple path-traversal attack to learn sensitive information about the content and structure of the database. Based on this information, you will execute an SQL injection attack to stealthily insert a new user into the database.

(a) (2 marks) Examine the content of the web portal for possible paths for the SQLite database. Write an exploit script that obtains a dump of this database and saves it as `data.db` alongside the exploit file.

(b) (12 marks) By examining the structure of the database file from part 5a, construct SQL queries that insert a new user into the database of the website, such that the new user is indistinguishable from a real user registered on the website. (You may need to insert the user into multiple tables in order to satisfy the above condition.) Include these queries as human-readable strings within your file. Again, you can refer to the Background Knowledge section for necessary background and examples on SQL and SQL injection attacks.

Next, identify an appropriate target for your second SQL injection attack. The log-in page is vulnerable to the SQL injection attack in question 4. The confirmation codes need to be preserved over time, and thus are probably stored in the database. Therefore, the confirmation page would likely also be using SQL queries to obtain the correct confirmation code as well as to reset the activation status of a user upon successful confirmation.

i. (8 marks) Conduct an SQL injection attack on one of the log-in or confirmation pages to insert a new user into the database, using the queries that you constructed above. Save the username and the password of the new user into the file `credentials.txt`. Note that you will need to go through either the previous part (5a) or question 2 to obtain information to include in your SQL injection.

ii. (4 marks) Log-in as the new user that you created, using the known username and password, and create a post as that user.

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

- `exploit.sh` — shell script that downloads the database, injects a user into it, logs-in as that user and creates a post by that user.
- `data.db` — database file downloaded from the server.
- `credentials.txt` — file containing the username and password of the 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 concatenation of all HTTP response headers from the server.

6. [Total: 10 marks] **Blind SQL Injection**

A web application can sometime unintentionally expose APIs that are vulnerable to SQL Injection but do not return any results directly. They silently succeed when the SQL query is successful, but show a generic error if the SQL query fails. An attacker can leverage this fact and observe the behaviour of such APIs with different injected inputs to infer confidential information. Such attacks are known as 'Blind SQL Injection' attacks.

In this question, you will conduct a blind SQL Injection attack on such an API to learn two secret PIN numbers. The website has two secret PIN values stored somewhere in the server as integers. One of them is a 'short' PIN of 4 random digits, whereas the other one is a 'long' PIN of 8 digits. If you visit `pin_check.php`, you will find that the website provides functionality to verify if these PIN values are set up properly. Additionally, it also provides functionality to reset both these PINs to fresh random values. Your task is to write a *single* script that performs blind SQL injection and retrieves the PIN value of a given type. Additionally, you need to ensure that your script can retrieve the value in a timely manner. Therefore, your script will be force stopped after a timeout of 4 seconds during marking.

(a) (2 marks) Examine the requests made by `pin_check.php` and figure out an API that has an SQL Injection vulnerability. Write an exploit script that performs SQL Injection attack on this API resulting in an error response. Save the error response with headers into the file `response.txt`.

(b) (8 marks) By examining the behaviour of the API identified above, construct SQL queries that can result in success or failure based on the PIN value. Include these queries as human-readable strings within your script. Again, you can refer to the Background Knowledge section for necessary background and examples on SQL and Blind SQL injection attacks.

Next, come up with an algorithm that can utilize these queries to extract the PIN value. (*Hint* : First, try to write a straightforward solution that might take significantly more time than 4 seconds. Then, think about how you can optimize.)

For 3 mark, your script must retrieve the 'short' PIN within 4 seconds, and output the extracted PIN to stdout. For a futher 5 marks, your script must retrieve the 'long' PIN within 4 seconds, and output the extracted PIN to stdout. (You have 4 seconds for *each* PIN, not total.)

Ensure that your script does not modify the existing PINs, but extracts them using SQL Injection. We will test your script by comparing its output to the initial random values that were used to initialize the PINs. Note that the PINs may start with a number of leading zeros. It is okay for your script to keep or discard these leading zeros in the output. For example, if the extracted short PIN is 98, then '98', '098' and '0098' are all valid outputs that will be marked as correct.

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

- `exploit.sh` — shell script that accepts **TWO** arguments: (i) a base URL, and (ii) a string, which will either be `response`, `short`, or `long`. Depending on this second argument value, your script should then do the following:

  `response:` It should generate a `response.txt` file with contents as described above in Question 6a.

  `short:` Perform the blind SQL injection operations as described in Question 6b above, and output the extracted 'short' PIN value.

  `long:` Perform the blind SQL injection operations as described in Question 6b above, and output the extracted 'long' PIN value.

- `response.txt` — Generated by your script when *you* ran it, as described above in Question 6a.

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.

7. [Total : 3 bonus 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 `Password123`).
   - 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) (1 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: `exploit7.tar` file which contains the following files (**not** inside a folder): `exploit.sh` script and the `response.txt` text file.

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 2 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 3: 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.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://ugster50X.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.

- Questions 4, 5 and 6: 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 and Blind SQL Injection examples on the OWASP SQL and Blind SQL Injection sites respectively.