

UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658

Computer Security and Privacy

Fall 2023
Yousra Aafer
Urs Hengartner

ASSIGNMENT 2

Milestone due date: **Friday, October 20, 2023 at 3:00 pm (48 hours automatic grace period)**

Assignment due date: **Friday, November 03, 2023 at 3:00 pm (48 hours automatic grace period)**

Total Marks: 66

Bonus Marks: 5

Written Response TA: Syeda Mashal Abbas Zaidi

Office hours: Thursdays 12:00pm–1:00pm

Programming TA: Parjanya Vyas

Office hours: Thursdays 12:00pm–1:00pm

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}.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 [30 marks]

Q1: Access Control

K-Software is a small start-up that provides software solutions to its clients. They have hired you as a security expert. The **sensitivity** and **clearance** levels within this company form the following hierarchy:

Intern \leq Software Engineer \leq Manager \leq Director

Terry is a Software Engineer with the following clearance level:

(Software Engineer, {source code, project plan, equipment, clients})

1. [5 marks] Using the Bell-La Padula Model, state if Terry has read and/or write access or no access, for each file:
 - (a) README: (Intern, {source code})
 - (b) Business Plan: (Director, {employee data, strategic plan})
 - (c) Equipment Registry: (Manager, {source code, project plan, equipment, clients, finances})
 - (d) Draft: (Intern, {source code, comments})
 - (e) Software Requirements Specification: (Software Engineer, {source code, project plan, equipment, clients})
2. [5 marks] K-Software uses the same hierarchy as above for its integrity levels as well. Kelly has the following integrity level:

(Manager, {source code, project plan, equipment})

Under the Dynamic Biba Integrity Model using the low watermark property described in the course slides, describe how the integrity level of Kelly and each file changes after each operation if she performs the following in sequence:

- (a) Kelly reads from file1 having integrity: (Manager, {clients, source code, project plan})
- (b) Kelly writes to file2 having integrity: (Software Engineer, {source code})
- (c) Kelly reads from file3 having integrity: (Intern, {comments, source code})

- (d) Kelly writes to file4 having integrity: (Director, {finances, project plan, source code, evaluations})
- (e) Kelly writes to file5 having integrity: (Manager, {project plan})

Q2: Firewalls

After seeing you succeed in understanding the company's access control, K-Software appoints you to reset its firewall. You are now responsible for recreating the firewall rules based on the requirements below.

Note that K-Software owns the IP range 16.35.125.0/24 (i.e., all IP addresses between 16.35.125.0 and 16.35.125.255) and uses a DENY by default firewall policy.

- All employees of the organization should be able to browse the internet (i.e. browse all HTTP and HTTPS web pages) from within the company network.
- Kim usually works remotely. She should be able to ssh into her work device. Her work device IP address is 16.35.125.25.
- There is known-to-be-malicious activity from the IP address range 85.63.28.0/24. The firewall should block all incoming traffic from this range.
- K-Software hosts a webserver at 16.35.125.13. This server runs over HTTP and HTTPS and needs to be accessible from anywhere in the world.
- K-Software trusts a special DNS server (located at the IP address 22.95.33.101) hosted by an allied organization to handle all of its DNS lookups. This DNS server only listens for DNS requests on port 53 and also expects the clients to send these requests from the ports in range 5000 to 5100. Assume DNS goes over UDP.

1. [5 marks] Configure the firewall by adding the required rules to meet the aforementioned requirements of K-Software . 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 (i.e., {}, {SYN}, {ACK}, or {SYN, ACK}). Not listing a flag means that a packet can or cannot have the flag set.

Here are example rules to allow access to HTTP pages from a server with IP address 5.5.5.5:
ALLOW 16.35.125.0/24 => 5.5.5.5 FROM PORT all to 80 BY TCP

For multiple ports 80 and 443:

ALLOW 5.5.5.5 => 16.35.125.0/24 FROM PORT all TO {80,443} BY TCP

For a range of ports from 80 to 443:

ALLOW 5.5.5.5 => 16.35.125.0/24 FROM PORT all TO [80,443] BY TCP

To allow anyone to access the server at any port:

ALLOW 0.0.0.0/0 => 5.5.5.5 FROM PORT all to all BY TCP

Note: More than one rule may be needed for implementing a requirement. There should be at most one rule for each traffic direction.

2. [2 marks] While configuring the firewall, you notice a series of IP packets from “outside” the company network that have their source IP addresses as 16.35.125.20. What is happening here? What can you do to stop it?
3. [3 marks] You notice that it may not be safe to run the organization’s web server inside the firewall. An attacker might be able to exploit faulty software running on 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?

Q3: Password Security

K-Software requires its employees to log in using a username and a password when they access their work laptops. You also notice that the office space is shared with another start-up and that access to the building is not restricted. You find that all passwords are hashed using MD5 and stored in a file available to untrusted individuals. The only rule is that passwords must be at least eight characters long.

1. [1 mark] What type of attack is this password scheme vulnerable to?
2. [3 marks] The company requires all employees to change their passwords every 90 days. Terry has been changing his passwords regularly. The following are the hashes of the recent password changes he has done:

- 25d55ad283aa400af464c76d713c07ad
- 25f9e794323b453885f5181f1b624d0b
- e807f1fcf82d132f9bb018ca6738a19f

Will this prevent attacks similar to the one in your answer to the previous question in the future? Explain why or why not using Terry's password hashes.

3. [2 marks] What are two improvements for how K-Software can store passwords that will make them more difficult to crack? Explain why these are improvements when compared to the basic password hashing mechanism.
4. [2 marks] K-Software now wants to use two-factor authentication along with passwords. They are trying to decide between (1) having an additional pin or (2) using an additional security key generated by an authenticator phone app. Can you comment on the validity of the two solutions? Justify your answer.
5. [2 marks] Why is MD5 not the best algorithm to create password fingerprints? What can be an alternative and why?

Programming Response Questions [36 marks + 5 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. If you are working 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

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 files in that language in your exploit tarball, and your script file should take care of

¹Note that, you should ensure that in your script, you run the version of `curl` that is on the ugster machine.

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 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.

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.

- (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: 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=q8ga3nh3uu9k5ttgf35m12fc7p; path=/
Location: index.php
```

Transfer-Encoding: chunked
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 Question 6 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 6 is a bonus question that can give a maximum of 5 bonus marks. Therefore, this question should be attempted at last. The marks obtained by this question will be considered as ‘Bonus Marks’, which means, this question is not mandatory to solve for this assignment. You can get full grades for this assignment even without solving this question. **No assistance will be provided for this question (unless a problem with its specification is found).**

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:  MySecretPassword
```

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 `MySecretPassword`).
- 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 “pvuser”.
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 “pvuser” 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 confirmation codes need to be preserved over time, and thus are probably stored in the database. Therefore, the confirmation page would likely 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 3 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: 5 bonus marks] **Blind SQL Injection**

A web application often unintentionally exposes APIs that are vulnerable to SQL Injection but do not return any result 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) (1 mark) 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) (4 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 2 marks, your script must retrieve the ‘short’ PIN within 4 seconds, and output the extracted PIN to stdout. For further 2 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’ and ‘0098’ are both 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.

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.
- 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.