ROYAL INSTITUTE
OF TECHNOLOGY

School of Electrical Engineering and Computer Science
Division of Theoretical Computer Science

LAB W
Web Application Security

NAME	KTH USERNAME

DATE :: _____-_____-_____

TEACHING ASSISTANT'S NAME :: _____

LAB W PASSED (TA'S SIGNATURE) :: _____

Cybersecurity Overview

DD2391 / HT2025

Computer Security

DD2395 / HT2025

Compile date: August 22, 2025

Contents

1 Introduction

The goal of this lab is to provide students with hands-on practice on web application security. The lab covers four common web application security vulnerabilities: Cross-Site Scripting (XSS), Cross-Site Request Forgery (XSRF/CSRF), Insecure Direct Object References (IDOR), and SQL injection (SQLi). According to the Open Web Application Security Project (OWASP), these vulnerabilities are among the top ten critical web application security flaws. This lab will help you find and exploit vulnerabilities in several web applications that run on a web server.

This lab must be done with the help of containers. Both Docker and Podman have been tested and should be usable for this lab. You will find the Docker commands you need for each section in this sheet. You may also want to get Burp Suite Community for the second part of this lab as an HTTP-proxy to allow inspection and interception of HTTP traffic between the browser and the web server, as well as URL encoding and decoding.

Docker & Burp Suite

For instructions on setting up Docker you can follow the instructions on the Canvas page: Instructions on How To Setup And Submit Lab O, Lab Z, And Exercises. Installation instructions for Burp Suite are provided in Section 4.
If you're using the CompSec VM, both Docker and Burp Suite are already installed.

This lab assignment is carried out in a **group of three students** and it requires an in-person demo and presentation of solutions to the TAs. The presentation and demo are mandatory for all group members and require booking a time slot with the TAs, as explained in the Canvas course page. During the presentation, you will demonstrate your working solution to each exercise and answer questions from the TA.

Throughout these instructions, several "Guiding Questions" are present to help steer you towards a solution and help you validate your understanding of the vulnerabilities at hand. Your assignment submission should include written answers to all guiding questions, and during the lab presentation each individual group member should be capable of explaining all parts of the lab.

Deadline

The deadline for the lab can be found on the course website:
`https://canvas.kth.se/courses/56170/assignments/338609`

Merit Point

Finishing all tasks without the "(Merit)" prefix is mandatory to pass this lab assignment (graded as 1 out of 2 on Canvas).
Additionally, you may optionally solve all the "(Merit)" tasks to receive a merit point (graded as 2 out of 2 on Canvas), which can help you unlock a higher overall course grade.

2 Preparation for the lab

The lab exercises require background knowledge about HTTP, HTML, CSS, and JavaScript as well as SQL for the second part. If you lack such knowledge, we strongly advise to prepare by reading material from MDN or W3Schools for a general introduction to these topics and the OWASP Attack overview for a general introduction to web attacks. You can also try out this XSS game by Google to get an understanding of XSS and the SQL-Insekten game or SQL injection walkthrough to get an understanding of SQLi. If you want to go deeper you can also try the relevant sections of the OWASP Webgoat or Google Gruyere projects.

It is unethical and criminal to attack websites beyond those intended for this lab

Do not try anything you learn in this lab on other websites beyond those specified in the lab instructions. Be aware that simply inserting a strange string containing some JavaScript syntax into, e.g., a search field on a website can be considered an attack and that you are potentially identifiable via your IP address.

The first part of the lab covers Cross-Site Scripting (XSS) and Cross-Site Request Forgery (XSRF/CSRF). If you need help, we recommend the OWASP XSS filter evasion cheat sheet.

The second part of the lab covers SQL injection (SQLi). If you need help, we recommend this SQLi cheat sheet and this filtered SQLi guide.

It is good to know that browsers ship with developer tools including a *Web Console* and source *Inspector*. The Web Console lets you see which JavaScript exceptions are being thrown and why they are happening. The Inspector tool lets you peek at the structure of the page and the properties and methods of the corresponding DOM tree nodes. You can also use the Web Console to analyze HTTP request headers, including the content of cookies sent alongside the requests (clicking on a GET request line opens a popup with detailed information). You may need to use Cascading Style Sheets (CSS) to make your attacks invisible to the user, which may require learning some basic CSS syntax, for instance, `<style>.warning{display:none}</style>`.

If you need to encode certain special characters, such as newlines, take a look at URL encoding.

3 Cross-site Scripting (XSS) and Cross-site Request Forgery (XSRF)

This section will help you understand what cross-site scripting (XSS) and cross-site request forgery (XSRF/CSRF) are. You will craft a series of attacks against the *Zoobar* website to exploit various vulnerabilities in the website's design. Each attack presents a distinct scenario with unique goals and constraints, though in some cases you may be able to re-use parts of your solution code.

Zoobar website

To run the Zoobar server for this exercise use the following commands:
`docker pull ghcr.io/kth-langsec/zoobar:latest`
`docker run -d --rm -p 7070:80 --name zoobar ghcr.io/kth-langsec/zoobar`
Use `sudo docker ...` if it shows permission denied.
The website will be available at http://localhost:7070; open it in your web browser.
To stop the server run: `docker stop zoobar` (this might take a few seconds)

Do not use real passwords

You will need to create user accounts on the Zoobar website to test your attacks. Do **not** use passwords that you use somewhere else to create the accounts here. The login information will be transferred unencrypted and the passwords are not stored securely on the server, so other students or attackers might be able to read your passwords.

Use Chromium for desktop

The attacks in this lab have been tested with an up-to-date Chromium browser. We suggest using an incognito window to do the lab.
If you're using the CompSec VM, Chromium is already installed.

3.1 Cross-site Scripting: Cookie Theft

You will construct an attack that will steal a victim's cookie for the Zoobar website when the victim's browser opens an URL of your choice. This type of attack is called Reflected Cross-site Scripting. You do not need to do anything with the victim's cookie after stealing it for the purposes of this exercise, although in practice an attacker could use the cookie to impersonate the victim and issue requests as if they came from the victim.

Your goal is to steal the document cookie and post it to a log using the Log-Write script found in
https://dasak-vm-lab-server.eecs.kth.se/logger/log.php,
a link to the log is found in the same page.

Log Script

In https://dasak-vm-lab-server.eecs.kth.se/logger/log.php, we have instructions on how to use the logger, which is just a convenience service we provide to help you solve this exercise. Messages sent to the logger server can be viewed at https://dasak-vm-lab-server.eecs.kth.se/logger/print.php. This log is cleared periodically.
Since the message storage is easily accessible, be sure to not send any sensitive information there.

The victim will already be logged in to the Zoobar website before loading your URL. Except for the browser address bar (which can be different), the victim should see a page that looks exactly as it normally does when the victim visits the website. No changes to the website appearance or extraneous text should be visible. Avoiding the red warning text is an important part of this attack (but it's fine if the page looks weird briefly before correcting itself).

Guiding Questions

What is the *Same Origin Policy*? How does the reflected XSS attack bypass this policy?

A web application vulnerable to reflected XSS will send the received unfiltered input back to the user (e. g., in a search result page). How can you test for a page's susceptibility to reflected XSS based on this behavior?

Hint: Sending the string `<script>alert("XSS works");</script>` via an input field might not suffice. Look at the source code of the returned page and check how and where exactly your input script is reflected.

Which is the vulnerable input element on the Zoobar website?

How can you use the log script to send data to the logger? Provide an example in the JavaScript language.

Which HTML DOM object has a cookie property?

The victim is not supposed to see any error messages. What does the following script do?
`document.getElementById("x").style.display = ...;`

Alternatively, how could you use a Location object to ensure the victim does not see any error messages?

3.2 Cross-Site Request Forgery: Stealing Zoobar Credits

In this scenario, you will transfer ten (10) Zoobars from the victim's account to your account. The victim is very naive and will load any HTML document that you send (e. g., as an e-mail attachment – an attacker might use social engineering on less naive victims to accomplish this).

You need to create one or more short HTML documents that the victim will open using their web browser. If needed, you can assume that the victim is logged in (authenticated with the Zoobar application) before loading your document.

It is acceptable for the victim to realize that a transfer of credits has occurred (though preferably they wouldn't), but only after the process has been completed and it is too late for them to abort it (i.e., the attack should not require any active action/confirmation on their part except for opening the HTML file).

Guiding Questions

How can XSRF work? Describe the browser behavior and problems with session management that make the attack possible.

Look at the source code and the HTTP headers during the transfer of Zoobars. Your crafted request should look exactly like a real transfer. What information (input) is being sent? Which type of request should you craft, GET or POST?

Normally, HTML forms are sent when users manually click a Submit button, but here we cannot rely on the victim to do that. How can you submit a form using JavaScript? Give a simple example.

3.3 Protection against XSS and XSRF

Questions

Describe how to prevent XSS flaws in a web application.

What can be done by a user to protect themselves against XSRF when browsing the web? Name at least two mitigations.

How can you prevent XSRF vulnerabilities on the server-side?

3.4 (Merit) Cross-Site Scripting: Password Theft

Create an attack that steals the victim's username and password, even if the victim is diligent and only enters their password when the URL address bar shows exactly `http://localhost:7070/`.

Your solution is a short HTML document that the victim will open using their web browser. The victim will not be logged in to the Zoobar website before loading your page. Upon loading your document, the browser should immediately be redirected to `http://localhost:7070/`, such that the address bar shows nothing else. The victim will then enter their username and password and press the "Log in" button. When the "Log in" button is pressed, the username and password (separated by a comma) should automatically be sent to the log script. The login form should appear normal to the user and, assuming the username and password are correct, the login should proceed the same way it always does.

Hints

- The website uses htmlspecialchars() to sanitize the reflected username, but something is not quite right.
- For this attack, you may find that using `alert()` to test for script injection does not work; Browsers may block it when it's causing an infinite loop of dialog boxes. Try other ways to probe whether your code is running, such as `document.loginform.login_username.value=42`.

Guiding Questions

What mistake did the Zoobar developers make that allows for reflected XSS on the login page, even though the input is being sanitized?

Even though this vulnerability allows injecting some payloads into the page, here a lot of characters cannot be used directly, since they would be sanitized or interpreted differently than an attacker would intend them to be. How can you get around this and run arbitrary JavaScript code despite these limitations?

3.5 (Merit) Cross-Site Request Forgery: Invisible Transfer

Expanding on the scenario described in exercise 3.2, devise an attack to transfer 10 Zoobars from the victim's account to your own, but without the victim ever noticing anything suspicious is taking place.

Create one or two short HTML documents that the victim will open using their web browser. You can assume that the victim is logged in (authenticated with the Zoobar application) before loading your document but it should not notice that a transfer of credits has occurred nor that *any* Zoobar-related operation is taking place. Thus, you should either redirect the victim to the course web page `https://canvas.kth.se/courses/56170/assignments/338609` after the transfer has been completed (fast enough that the victim does not notice the transfer or any intermediate operations whatsoever) or show something else to the victim without redirecting. The main point is that the victim does not become suspicious. In particular, the location bar of the browser must not contain "localhost:7070" (or equivalent) at any point in time (not even for a millisecond). Notice that a funny cat GIF suffices and does not require a redirection - after all, who doesn't like cat images?

For this exercise, an attack is only considered successful if the entire interaction with the victim never makes any reference to Zoobar nor even shows them part of the Zoobar website, which would make them suspicious. The exploit must be truly invisible to an unaware victim (who will never inspect the page source code nor web requests initiated by your documents).

If necessary, you may assume that your HTML file(s) are being served on a specific port on localhost when the victim accesses them. In order to serve files locally, you can for example use `python3 -m http.server 1234`.

Guiding Questions

Briefly explain what Cross-Origin Resource Sharing (CORS) is and how the restrictions it imposes help secure against XSRF attacks.

What is the Zoobar application's CORS policy?

4 SQL injection

The following exercise will help you understand some SQL injection techniques and systematic vulnerability testing. While the main focus is on the SQL injection, there is also one Insecure Direct Object References vulnerability that you are intended to exploit.

The first part of the lab exercise is based on the Hackxor webapp hacking game. The game consist of a set of websites that you are supposed to hack. The first two steps of this hacking game, which include *Wraithmail* (an e-mail service) and *Cloaknet* (an anonymizing proxy service), are used for this exercise.

If you want to try your skills directly, we encourage you to try solving these first two steps of the game without reading the lab instructions in the following section. Nevertheless, you will in any case have to answer all of the lab's questions.

Wraithmail & Cloaknet website

To run the Wraithmail & Cloaknet server for this exercise use the following commands:
`docker pull ghcr.io/kth-langsec/wraithmail-cloaknet:latest`
`docker run -d --rm -p 8080:8080 --name wraithcloak ghcr.io/kth-langsec/wraithmail-cloaknet`
Use `sudo docker ...` if it shows permission denied.
The websites will be available at `http://localhost:8080/wraithmail/` and `http://localhost:8080/cloaknet/`, respectively.
To stop the server run: `docker stop wraithcloak` (this might take a few seconds)

4.1 Preparation: Burp Proxy Setup

It is recommended to use an intercepting web proxy for the exercises; we recommend using Burp Suite Community. Download and install Burp Suite and then run it. You can choose to use either a temporary or stored project with Burp defaults. In the Burp suite, go to the tab named `Proxy`. Make sure the button `Intercept is on`/`Intercept is off` is set to `on` and then click the `Open browser` button. This will start a browser with Burp as an intercepting proxy. Once a request has been intercepted, you always have the possibility to change it (just edit the displayed text) before actually sending it out (by pressing the button `forward`). For a more in depth introduction we recommend the Getting Started with Burp Suite tutorial; for this lab you only need to read steps 1, 2, and 3.

4.2 Wraithmail

The game scenario is as follows: you are a hacker who got the task to track down a person who performed an attack on one of the Wraithmail accounts. For that, you should login to Wraithmail and read the message named "trace job".

Wraithmail

Website: `http://localhost:8080/wraithmail/`
Username: `algo`
Password: `smurf`

In this first part, you will exploit an Insecure Direct Object References vulnerability. You are encouraged to try any other attack to check whether the website is vulnerable to any of them.

Hints

- Pay attention to the *referer* line in the "trace job" letter. What does the referer field mean?
- Explore the website. Which functionalities (like composing a message, etc.) does it offer? Is one of them vulnerable? Note: You will need an intercepting proxy to exploit the vulnerability.
- If your attack succeeds, you will be able to see that the *abuse contact* is from *Cloaknet*.

Questions

What can you infer about the attack attempted by the hacker that you are tracking? Briefly describe what it might be doing (without details, just the rough idea).

What username did the hacker use for the attack and what was the logged IP address?

4.3 Cloaknet

By now, you have found evidence that the attacker used a proxy service called Cloaknet to hide their real IP address. In order for you to find out who the attacker is, you should proceed to the Cloaknet website:

Cloaknet

Website: `http://localhost:8080/cloaknet/`

There, your task is to perform an SQL injection attack to retrieve the account credentials (username, password, ID, etc.) of all registered users at Cloaknet. You should also be able to confirm that the source of the attack was from `GGHB`, that the target was `wraithmail:8080/send.jsp`, and that the date matches the one you could see in the Wraithmail logs.

You might need to look at the SQL Injection Cheat Sheet mentioned in Section 2 to complete this task. Note that testing for the right way to get around the input filters can take some time. Try to test it systematically, but if you get stuck you can ask the lab assistants for hints.

Guiding Questions

Name 3 mechanisms that a website might use to transfer data from the browser to the server within an HTTP request (e.g., when submitting a form, or to remember that user is logged in).

As most web applications, Cloaknet performs database queries internally when loading its pages. Since content is contextual, these SQL queries take certain values as inputs, such as the username you type on the login page (`user`). Name two other values that you know Cloaknet uses as inputs in SQL queries.

Hint 1: These two values are sent to the server using two different mechanisms (see the first question above).

Hint 2: Look at the two HTTP requests sent to the server when logging in.

How many SQL queries do you think are issued when you log in to the website? What inputs might these queries use? What do you think these SQL queries look like? Write them down as precisely as possible (e.g., using SQL syntax/pseudo-code).

Check whether the Cloaknet web application's inputs are vulnerable to SQL injection. How can you do that? Give two examples of possible checks.

Were you able to cause an SQL error or a server error (HTTP status 500) to be displayed on the webpage? How did you do it? What information did you learn? What kind of database and web server are being used?

Why is it useful to know which database engine is used?

When you try to inject a meaningful SQL string, a first obstacle you might encounter is that some characters are not accepted by the server in cookie values - in this case, whitespace. How can you get around this specific limitation in your injected query (i.e., perform SQL queries without spaces)?

All inputs are filtered! The filter escapes some characters and removes some words (e.g. `SELECT`). Name at least four techniques to bypass filters in general (with examples).

Hint: Check the cheat sheets mentioned in the lab's Introduction.

What was the source IP address for the attack we are tracking, coming from `GGHB` on the correct date?

Hint: When you log in you already see a list of attacks, but you need to see all of them, not just yours.

Table and column names are very predictable in this lab. What could be the name of the table containing all usernames and passwords?

The `UNION` operator is used to combine output from several `SELECT` statements. What requirements should these statements meet in order to be "unioned"? What do you have to do if the tables you want to union do not have the same number of columns?

You should be able to infer the number of columns used in the query statement from the output you see on the page. Sometimes, though, not everything is shown, and then it is important to find out the number of columns. How can this be done? Provide an example.

You can union tables even without specifying column names. How can you include all columns of a table in the `UNION` statement without explicitly naming them?

What is the hacker's username and password? How did you get the list of usernames and passwords? How did you know which one was the attacker?

4.4 Protection against SQL injection

There are many mitigation techniques for SQL injection.

Questions

Describe how parametrized queries help protect against SQL injection. What other protection techniques can be used on the application's code level?

What is a stored procedure? Describe how it protects against SQL injection. What other protection techniques can be used at the database level?