Chair of IT Security
Brandenburg University of Technology

Asya Mitseva, M.Sc.
Fabian Mildner, B.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Summer Term 2025

Part IV: Web Security **Deadline: 18th June 2025**

Cyber Security Lab (Ethical Hacking)
SQL Injections and Cross-Site Scripting (XSS)

Introduction

The web has been a rapidly developing technology with the constant addition of new standards and complexities. Hence, it should come as no surprise that there is a wide range of possible vulnerabilities a web operator must prepare for and defend against. In this lab, we will concern ourselves with two of the most classic and still commonly discovered vulnerabilities in web applications: Cross-site scripting (XSS) and SQL injections.

SQL injection is a code injection technique that exploits vulnerabilities in the interface between web applications and database servers. Many web applications take inputs from users and then use them to construct SQL queries and get information from the database, with the goal of displaying this information to the user. In addition, web applications also use SQL queries to save and modify information in the database. Usually, these queries are meant to achieve a specific task, each, and provide only specific sets of information to the user's web client. However, when SQL queries are not carefully constructed, SQL injection vulnerabilities may occur. The threat of the SQL injection arises when user inputs are not correctly checked and/or sanitized by the web application before being sent to the back-end database server. This can allow an attacker to escape the intended scope of the original query, letting him gain vast amounts of information and tamper with the data kept and managed by a webserver.

In contrast, Cross-Site Scripting (XSS) is a type of vulnerability commonly found in web applications themselves. An XSS vulnerability makes it possible for attackers to inject malicious code (e.g., JavaScript programs) into the web browser of a victim user by tricking the website into serving a webpage running an attacker's script. Using this malicious code, attackers can steal secrets and user credentials, such as session cookies. This is especially troublesome, because it circumvents the Same Origin Policy typically employed by web browsers to prevent theft of credentials and cookies through a website other than the one associated with them. Further, such XSS vulnerabilities can potentially lead to large-scale attacks.

Please Note: Although you are provided with the server-side code of the web application for the purposes of being able to host the webserver on your own, an attacker in a real scenario would not have access to these resources in most cases. Therefore, **we expect that you solve tasks 1–5 without analyzing the server-side PHP code**. Doing so would only hinder you in learning how to approach your attacks in the black-box case, which is what you commonly encounter as a first step when attacking a real web resource.

Preparation of the Experimental Setup

In this laboratory exercise, we provide you with a webserver giving a very basic forum functionality. A user visiting the landing page of web application is prompted to log in to the platform. After successful authentication, the user gets redirected to a page where he can read already posted comments and post comments of his own. Furthermore, the website hides a couple of secrets that we will uncover bit by bit as we work to exploit its inherent flaws. Once again, we'd like to remind you to avoid looking into the provided application source code at this stage, so that you can discover these secrets naturally through your exploits and what is available in your web browser.

Setting up the Web Application

For the purposes of this laboratory exercise, we again provide you with a `docker compose` [1, 2] configuration that will enable you to build, start and stop the associated container. The operations that you can perform are as follows:

1. **Creating and starting the infrastructure:** To setup the infrastructure, navigate to the folder in which you saved the `docker-compose.yml` file provided by us. From there, you can execute the command:

   ```
   $ sudo docker compose up
   ```
   The docker-compose tool will automatically download the needed image files for the containers, create the required networks and connect and configure containers as specified in the `docker-compose.yml` file. The containers will run as long as you do not stop the command. Afterwards, you should be able to see running containers using `docker ps` (e.g. run in a different terminal).

   Please Note: Some of you may be familiar with the command `sudo docker-compose up` instead. This is also a possible option, however, `docker compose up` is the more modern solution, which is why we suggest it. It should be noted that there are slight behavioral differences between these commands.

2. **Stopping the infrastructure:** There are two main ways to stop the running containers. On the one hand, you can simply stop the command issued before using `CTRL + C` from the terminal window from which the network was launched. On the other hand, you can also issue a `sudo docker compose stop` command from within the same folder as the `docker-compose.yml` file to achieve the same effect. This procedure has the advantage of also being usable when you started the containers in detached mode, i.e. without keeping the previous command in the foreground of your terminal window.

3. **Cleaning up:** If you are running the infrastructure on something other than a disposable virtual machine, you may want to clean up the containers created by docker compose after you are done. You can do this with the `sudo docker compose down` command. Note that this command may miss some containers if you performed changes to the configuration

file over time. You can also use `docker container prune`, `docker network prune` and `docker image prune`. Please be aware that this will delete all unused docker containers, networks and images, which may play a role if you already use docker for other purposes than just this lab.

Using the Provided Application

Once you started it, the web application specified by the `docker-compose.yml` file will begin to listen for incoming requests on the IP address `10.9.0.90`. If you prefer to use an alias like `sql-lab` instead of the IP address `10.9.0.90`, you could elect to modify your system's `hosts` file (under linux: `/etc/hosts`) accordingly. To verify that your setup is working, you can visit `http://10.9.0.90/`. You should be redirected by your browser to the index page of the website where a login prompt is shown.

For the purpose of this laboratory exercise, you are assumed to be the user *ada*. Your credentials are:

```
Username: ada
Password: lovelace
```

Resetting the Web Application

If you encounter any error with the web application during your attacks or want to clean up any side-effects from your previous attacks, you can reset the application to its initial state at any time. Resetting the application will result in a loss of all posted comments and will restore the enrolled users to the default user database. It will also discard any database entries modified by SQL injection. Please note that resetting this data can not be reverted! If you want to perform a reset, you can visit the page `http://10.9.0.90/reset.php`.

Getting Familiar with Web Developer Tools

To succeed in this laboratory exercise, you will have to analyze and construct HTTP requests. There are many useful tools to help you investigate the requests sent between your browser and the web server. The most basic set of tools to help you with this can be found in the *Web Developer Tools* typically bundled with any common web browser. In Firefox and Chromium, this toolset will come up by pressing the `F12` key. These tools allow the user to inspect, debug, and analyze certain aspects of a web application like the source of the page, stored local data, and network transmissions. These built-in tools provided by your web browser should already be sufficient to solve the tasks of this lab.

However, it should be noted that there are many more advanced tools available that have additional advantages, such as capturing and delaying HTTP requests to allow tampering with them before they reach the server. Common examples are the *OWASP Zed Attack Proxy (OWASP ZAP)* or the *Burp Suite* set of tools. There is also a useful Firefox add-on called *HTTP Header Live*. If you want to utilize any of these tools, feel free to. In any case, be sure to familiarize yourself with the tools you want to use before you start your work on the following tasks.

Getting Familiar with the Provided Web Application

Finally, to get a feel for the application and possible attack vectors, you should take some time to play around with the application. What pages can a normal user reach? Which of them require you to be logged in, which do not? This type of reconnaissance is a first step in determining where your following attacks can be mounted. You can start by visiting the landing page of the website at `http://10.9.0.90` and seeing what pages can be reached naturally from here. In addition, you can play around with the comment functionality of the website. You will notice quickly that there is not much this simple forum offers to a normal user – however, more will come to light during the course of the next tasks.

Tasks

Task 1: Obtaining the Admin Password

It is reasonable to suspect that `ada`, which is our username, is not the only user enrolled in the system. In particular, systems often have an admin user account by the name `admin`. Your goal in this task is to find out the password of the admin user by means of an SQL injection. Your ultimate goal is to use these credentials to log in and post an arbitrary comment as the admin user. Furthermore, you should provide a list of all users and their passwords in a file `users.txt`

When explaining your solution process for this task, make sure to explain how you found the SQL injection vulnerability and how you were able to discover the results of your malicious query.

Hint: While a properly configured website would save only salted hashes of the input passwords, this one saves each password in clear text. Whoops.

Task 2: Listing the Pages of the Web Application

While stealing the admin password is a powerful attack already, in practice, it does not provide us with anything new in our simple forum. Furthermore, in a properly secured password database, it may be difficult to retrieve a strong admin password even if the according database entry is dumped. However, beside the credentials of enrolled users, web applications often store more information in their back-end databases, which may help us discover further avenues for attack. In this task, you shall use an SQL injection to obtain a list of all pages provided by the web application, without having to resort to exhaustive enumeration techniques.

Are there any webpages of the website that are (normally) not shown to its users? List and investigate the additional webpages you discovered, highlighting whether they can be visited only by *authenticated* or also by *unauthenticated* users.

Hints:

1. When executing SQL injection attacks, it can be helpful to learn about the exact database back-end that is used. Think about how you can figure out this information, and why it is of help in solving this task.
2. Because of another vulnerability of our website, it would be possible to solve this and the previous task without any SQL injections at all. Feel free to think about what this vulnerability could be - however, we will focus on SQL injections here.

Task 3: Discovering a Reflected XSS Vector

In the previous task, you have found two pages that are not usually accessible by the user. One of those pages provides managers of the website with a statistical tool to lookup the visit counts of each of the webpages. Play around with it to discover how to works. When you look closely at what happens when you lookup a webpage in the provided search field, you may start to suspect that this field might be vulnerable to an XSS Reflection Attack. Think about why this is, and how you can test whether your suspicions are true.

In the process of this task, you should perform some reconnaissance work to help build your next exploit. In particular, you should be able to answer the following questions for yourself:

1. Does the search text field provide input validation before sending the request to the server? Is it vulnerable to an XSS Reflection Attack?
2. What HTTP method is used to send this information, i.e., the search query, to the web server?
3. How is the sent data encoded / structured?

In addition, think about the website's authentication mechanisms and how they relate to this webpage. Why does this makes this page particularly interesting to us, especially in regards to the attacks built in the following tasks?

Task 4: Using the Reflected XSS Vector to Hijack User Sessions

One of the major threats of reflected XSS vulnerabilities is that they allow an attacker to steal credentials of a logged in user, i.e., to hijack user's sessions. In particular, because input to the vulnerable search field can be given through a URL parameter, it is possible to craft a malicious URL of the form `http://10.9.0.90/<webpage>?q=<malicous-code>`, which leads to the reflected XSS vulnerability being exploited as soon as the user clicks a link.

The goal of this task is to craft a malicious URL that steals the session information of an already logged-in user and sends it to the attacker in a suitable manner. More specifically, we assume that the user `ada` is logged in to the commenting platform and opens the malicious link as sent by the attacker. To facilitate the theft of the session information, we suggest that you host an additional, attacker-controlled web server which may receive the stolen data as soon as the user unwittingly clicks the malicious URL. Afterwards, the attacker is able to use this information to perform actions with the user's name and privileges, such as viewing the comments page.

To mark this task as succeeded, you should be able to demonstrate how your malicious URL allows the attacker to steal ada's session info and how he can use it to view the comments page. It is not necessary to be able to post any comments as ada yet, as this will require overcoming one more challenge discussed in the next task.

Hints:

1. While in the previous task, you directly inserted malicious JavaScript into the page, it is infeasible to encode the whole malicious payload into the malicious URL for several reasons. Instead, the main-stage payload is usually loaded from a remote server as to facilitate more complex attacks. Therefore, we recommend you to write the code for the attack in a separate file `exploit.js` and to serve it to the web application using `<script src=http://.../exploit.js> </Script>` tags.
2. Python provides many useful tools out of the box. For example, the `python3` module `http.server` provides resources to serve an HTTP server. A basic server can be started directly using the command `python -m http.server`, serving all the files in the current working directory and logging incoming connections to the console.

Task 5: Posting Malicious JavaScript to the Comments using XSS Attacks

In the previous task, we have seen how a reflected XSS attack can be utilized to hijack a user's session at the click of a malicious link. While this is a powerful attack already, it is far from all that this XSS vulnerability can lead to. Ideally, we want to attack not just one user, but several. Fortunately for us, this is possible by chaining together multiple XSS vulnerabilities. Furthermore, we don't want to have to make use of the stolen credentials by hand. Our ideal goal is to run the exploit fully within the breached user's web browser, leading to them posting a malicious comment in their name without any trace of our own IP. At the end of this task, one careless click of a logged-in user will be enough for us to infect every other visitor of the forum, executing the following evil piece of JavaScript code on their systems:

```
<script>
        alert("ALL YOUR SCRIPT ARE BELONG TO US.");
</script>
```

There are a couple of obstacles to overcome, however. First of all, we want to determine whether it is possible at all to perform an XSS reflection attack on the `comments.php` webpage. If so, we would be able to inject the above script as a persistent threat that infects every user that visits this webpage. However, when trying out a few payloads, you will notice that the according text field is validated against malicious inputs. Can you determine how this validation is performed and how to circumvent it?

Furthermore, you must investigate how you can use the stolen credentials in your exploit payload to automatically post a comment in ada's name. To this end, you should investigate the HTTP requests/responses associated with posting a comment. Are you able to simply change the request and insert ada's session information to post a comment in her name? What additional protection measure is preventing you from doing this? To help you proceed, you should research what a *CSRF* token is, what it defends against, and how it is utilized and checked. A main part of your exploit will concern itself in how to gain a valid CSRF token such that you can automatically post a comment with ada's information. To this end, it may be necessary to make additional requests and parse the contents of resulting webpages.

In the end, you should have a malicious URL and a javascript exploit code that allows you – an unathenticated attacker that is not enrolled in the forum at all – to trick a logged-in user into compromising the entire userbase. Once the logged-in user clicks your malicious link, *every user* that visits the `comments.php` webpage should be presented with a javascript alert window such as the one created by the malicious code provided above. You should verify this fact by logging into the webpage as different users. You can base the development of your exploit on the solutions you created in the previous task.

Task 6: Fixing the Discovered Vulnerabilities

Since this lab concerns itself with ethical hacking, we remain responsible to fix or at least report any vulnerabilities we discovered during our investigations. Normally, we would do this by reporting our findings along with a detailed report to the website owner. However, in this case, we do have the source code of the website available. Therefore, for this task only, you are asked to take a look at the provided PHP files of the webserver. What lines of code are responsible for the vulnerabilities you observed during the previous tasks? Explain what is wrong with these lines and how they can be fixed to prevent the discovered issues.

Preparation of Your Submission

Your submission for this lab should contain the final versions of any and all source code you have written and files you have generated, along with a detailed writeup about the tasks you solved. Your writeup should be given in an appropriate format, containing detailed descriptions of how you solved each task, what theoretical insights you gained, what observations you have made as well as images / screenshots / listings to aid the understanding of your approach. Your writeup should be submitted in PDF format.

Remember to submit the list of users/passwords for Task 1 and the list of discovered pages (authenticated or unauthenticated) for Task 2. In addition, it is important that you submit your malicious URL and your exploit payload for Task 4 and Task 5, as well as the fixes you propose for Task 6.

Please submit your solutions in a packed zip file named after your full name, with the following structure and minimum required contents:

```
firstname_lastname.zip
├── Writeup.pdf
├── task01
│   └── users.txt
├── task02
│   └── pages.txt
├── task04
│   └── exploit.js
├── task05
│   ├── exploit.js
└── task06
    └── <php files with highlighted fixes>
```

Preparation for the Q&A Session

Prepare yourself for a Q&A session of up to 40 minutes. During the Q&A session, you should be prepared to *explain and demonstrate* how you solved the tasks. Take special care to explain how your solution works and why you took the approach you took. Note that simply presenting a piece of code without any explanation will not yield any points! Last but not least, you should be ready to answer spontaneous (theoretical) questions asked by the reviewer in relation to the topics of this lab.

References

[1] *Docker*. URL: https://www.docker.com/ (cit. on p. 3).

[2] *Docker Compose*. URL: https://docs.docker.com/compose/ (cit. on p. 3).