KOÇ UNIVERSITY

COMP 430/530: Data Privacy and Security – Fall 2021

Homework Assignment #4

Question 1: Authentication [total: 40 pts]

DigitalCorp, an imaginary company, stores customer usernames and passwords on their "secure" servers. They use SHA-512 for hashing passwords but no salts. You just hacked into DigitalCorp's servers and stole a part of their username-password list. You stored the stolen list in a file called digitalcorp.txt (digitalcorp.txt is available in the homework folder).

Since you are an elite hacker, you are also aware of the RockYou breach and you suspect some RockYou users are also customers of DigitalCorp. You download the RockYou password dataset. A small version, which is sufficient for this question, is available on Blackboard (rockyou.txt).

(a) [15 pts] Create a dictionary attack using RockYou. Given rockyou.txt, your code should generate a csv file that contains the dictionary attack table. Submit your source code and the output of your code (the attack table).

(b) [4 pts] What are the passwords of DigitalCorp users Alice, Bob, Charlie and Harry? Use the attack table from part (a) to infer their passwords.

Now consider that DigitalCorp has updated its security and uses salts when storing passwords. Yet, they have not updated remaining aspects of their security; thus, you were able to hack into their servers again and this time you stole the salted username-password file: salty-digitalcorp.txt.

(c) [3 pts] Does your dictionary attack from parts (a) and (b) work? Why or why not?

(d) [3 pts] The file salty-digitalcorp.txt contains users Dave, Karen, Faith and Harrison. Devise an attack to find their passwords. Verbally describe your attack strategy and discuss whether this attack requires less computation or more computation than what you did in parts (a) and (b).

(e) [15 pts] Implement your new attack. Submit its source code as well as the true passwords of Dave, Karen, Faith and Harrison.

Note: In this homework, salts are prepended to passwords (added to the beginning) rather than appended (added to the end).

Hint: You can use the **hashlib** module in Python for SHA-512 hashing.

Deliverables

For this question, you need to submit your Python source code, supporting files (e.g., attack table), and a report including your answers to each part.

Question 2: SQL Injection [total: 60 pts + 10 pts bonus]

Many web applications take input from users and use it to construct SQL queries to retrieve information from databases. SQL injection is a code injection technique that exploits the vulnerabilities in how a web application interacts with the database server. When the user's input is not properly checked in the web application before sending it to the backend database server, these vulnerabilities are introduced.

In this assignment, we are providing you a web application that includes common mistakes that web developers make, which cause the web application to be vulnerable to SQL injection attacks. The goal of this assignment is for students to find ways to exploit these vulnerabilities and demonstrate the damage that can be done by SQL injection attacks.

This assignment includes two files: ***auth.php*** and ***union.php***, which include SQL injection vulnerabilities. The first file should be exploited using tautology or other types of SQL injection attacks that enable logging in without knowing the password; the second file should be exploited using union-based SQL injection.

Installation

Before you start, you need to make sure you have PHP and SQLite3 installed.

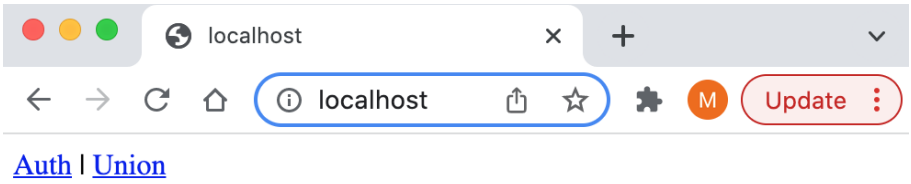
On Ubuntu, you can get them both installed via: ***sudo apt install php php-sqlite***.
On macOS, you can get them installed after having **Homebrew** installed. Use the command:
brew install php

To check whether PHP and its SQLite3 module are installed, you can type ***php -m*** and see if ***sqlite3*** is listed as one of the installed modules.

Running

To run the assignment, change the current directory to the assignment directory,
then type ***php -S 0.0.0.0:80***
This will run a PHP web server on your machine available on port **80**.
You can access this server via your web browser on the same machine by going to
http://localhost

After running the web server and accessing the server via localhost, you will see a webpage like the image below. You can choose Auth to start authentication-based challenges and Union to start the union-based challenge. You can also see the web server access logs in the terminal window where you run PHP, for every click or refresh that you do in the web app.



Note: To complete the SQL injection challenges, you don't need to modify the PHP code. You simply need to find the correct payload to exploit the vulnerabilities. You are encouraged to look at the code to understand how the queries are constructed for each challenge.

Useful links:

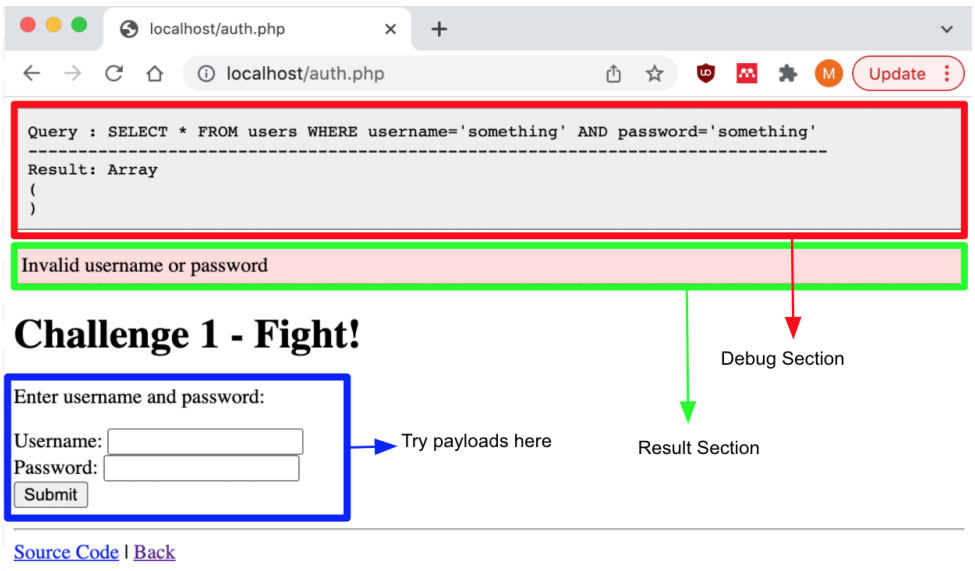
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection

Part 1 (Authentication Based Attacks)

In this part, you need to bypass the login screen without actually knowing a proper username and password combination. As passwords are generated randomly on the first run of the app, you cannot know them unless you explicitly open the SQLite3 database and look for it.

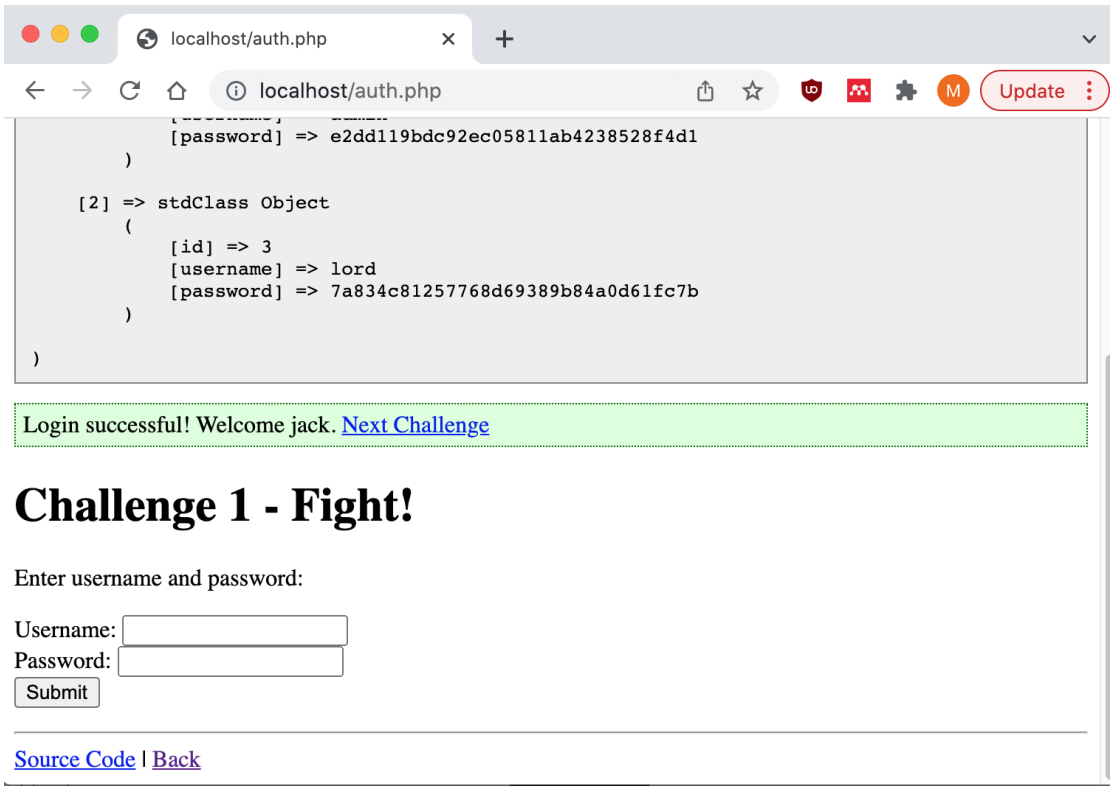
This part has 4 challenges: The first 3 are mandatory and the last one is optional (bonus). Once you solve a challenge, you will receive a link that points you to the next challenge. You can also navigate between challenges by changing the challenge number in the URL: **http://localhost/auth.php?challenge=1.**

For every input that you enter, the web app shows you a debug section in a gray box on the top that displays both the generated SQL query as well as the result fetched from the database.



Challenge #1 [10 pts]

In this challenge, there is no protection against SQL injection attacks. Therefore, you need to craft a relatively basic payload to bypass the login. You can submit different payloads in the username and password textboxes and look at the debug section to see how the SQL query was constructed using your input. Once you figure out the correct payload, you will be able to see a successful login message in the result section and a link to the next challenge, just like the image below.



Challenge #2 [15 pts]

In this challenge, one layer of protection was used against SQL injection, as you can see in the ***auth.php*** code – the function responsible for constructing the SQL query uses a method to prevent SQL injection attacks known as the *escaping* method. The payload you used for the previous challenge may or may not work for this challenge. You need to craft a proper payload in the username and password section to be able to bypass this level of protection and log in successfully.

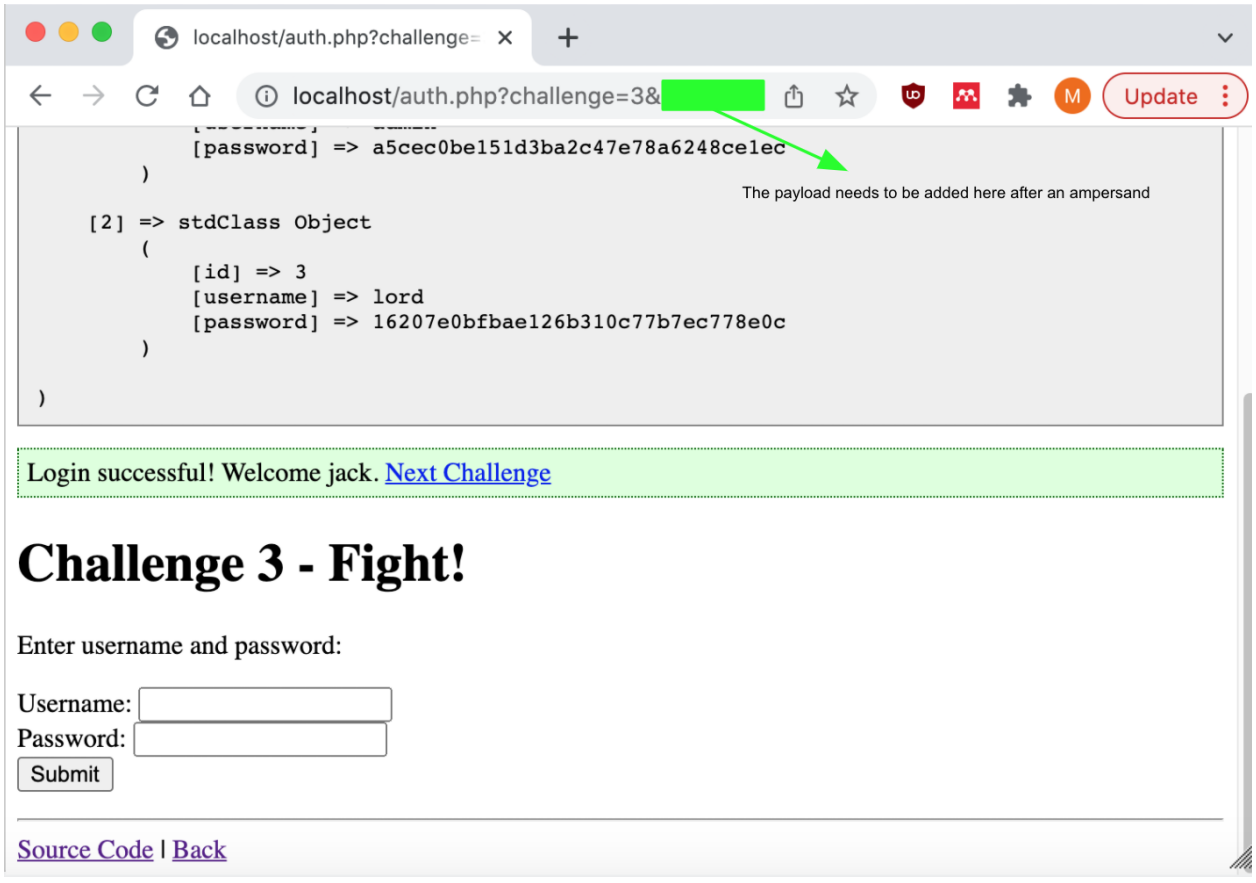
Challenge #3 [15 pts]

In this challenge, parameterized SQL queries (also commonly known as prepared statements) have been used to prevent the attacker from bypassing login without correct credentials. Prepared statements are pre-compiled and reusable, forcing the web application to send the query and the user input separately to the database. They are among the most effective techniques to prevent SQL injection attacks.

In this challenge, the `order by` clause has been added to prepared statements which introduces a vulnerability and creates an opportunity for you to exploit it. You are required to craft a payload to bypass the vulnerable prepared statement.

For this challenge, **you need to append your payload to the URL** with an ampersand character separating parameters. Your new URL would look like: *URL&payload*

To be able to bypass the login you need to **refresh the new URL** then **enter something in the username and password section** and **submit**, allowing you to see the "Login Successful" message in the result section and the constructed query in the debug section.



Challenge #4 (bonus) [10 pts]

This challenge is not a mandatory part of the homework and is a bonus. Since it is more difficult in comparison to the first three challenges, we recommend you complete the other parts of the homework first, before attempting to solve this challenge.

In this challenge, prepared statements are used but due to the `limit` clause there is still an opportunity for you to bypass the login using a SQL injection payload. Just like the third challenge, you need to add the payload to the URL.

You can watch this video for a hint.

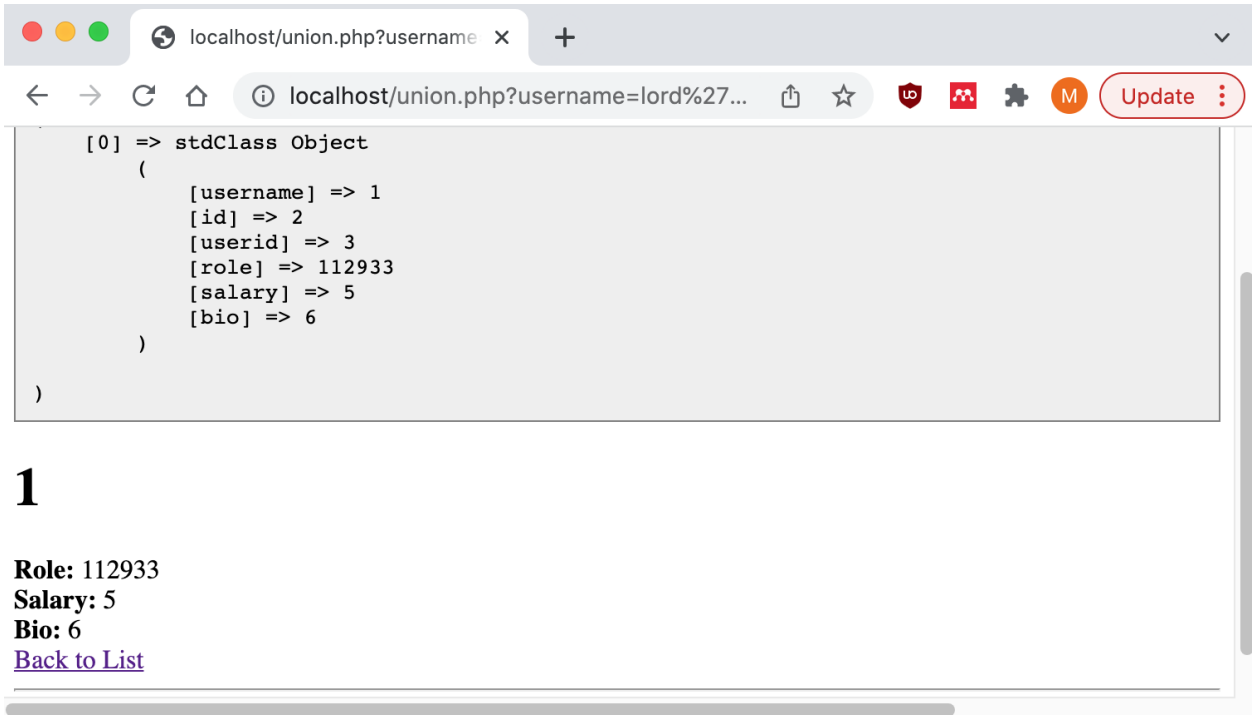
Part 2 (Union Based Attack) - Challenge #5 [20 pts]

In this challenge, you need to extract the classified salary of a user that cannot be seen normally in the system. The system allows you to select employees and see their profiles, including their salaries by default. However, if an employee has a salary above 100,000, the number will not be shown by the app and will instead be classified.

First, manually browse the employees until you find the classified one. Then, using union-based SQL injection, extract the classified salary. Note that the salary is not the deliverable for this challenge, but the payload you used for the injection. You should add your payload to the URL of the application as shown in the image below. Experiment with different values and observe via debug information how it affects the generated and executed SQL query.

Note that you should not rely on the debug data shown at the top of the page, but rather you should make the app display the salary as part of its normal flow. Also keep in mind that each time you restart the program you will get a new number in the salary section because the desired salary above 100,000 is created randomly when the database is created the first time.

Note: It is acceptable if you craft the union payload in a way that after refreshing the URL, the salary number above 100,000 is shown in place of other user profile information such as their role. For example, see the following image.



Deliverables

You need to submit a report including the following information:

- The payloads that you used to cause the SQL injection. Be precise when you are writing your payloads, every character (spaces, apostrophes, dashes, etc.) may be important!
- A brief explanation of the exploited vulnerability and the reason why this payload works.
- Screenshot(s) of your successful exploit for each part. Please make sure that you take the screenshots in a way that proves they were taken on your own machine/account.

Note: Screenshots of the attack with no payload and explanation will not get any credit.

SUBMISSION

When you are finished, submit your assignment via Blackboard as follows.

- Combine the report for question 1 and 2 into one single file. Convert your report to pdf.
- Move all of your relevant files (including code files, pdf report, etc.) into a folder named **your_KUNet_ID.**
- Compress this folder into a single zip file. Do not use compression methods other than zip.
- Upload your zip file to Blackboard.

Some reminders:

- After submitting, download your submission and double-check that: (i) your files are not corrupted, (ii) your submission contains all the files you intended to submit.
- This homework is an individual assignment. All work needs to be your own. Submissions will be checked for plagiarism.
- Your report should be readable on any computer. We cannot grade files that are only readable on a Mac.
- Submit only through Blackboard. Do not e-mail your assignment to the instructor or the TAs. Make sure to submit ahead of time to avoid Blackboard-related problems or delays.
- If your code does not run (e.g., syntax errors) or takes so long that grading it becomes impossible, you may receive 0 for the corresponding question.

GOOD LUCK!