COMP 430/530: Data Privacy and Security – Fall 2023

Homework Assignment #4

Question 1: Password-Based Authentication [total: 50 pts]

Part 1: No Salts, No Key Stretching [15 pts]

DigitalCorp, an imaginary company, stores customers' usernames and passwords on their servers. They use SHA-512 for hashing passwords but no salts and no key stretching. You 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` (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 homework, is available in the homework folder (`rockyou.txt`).

Assuming that all users in `digitalcorp.txt` have a password equal to one of the passwords stored in `rockyou.txt`, create a dictionary attack using `rockyou.txt`. Given `rockyou.txt`, your code should find the passwords of all DigitalCorp users in `digitalcorp.txt`. Implement your attack in Python and submit its source code. Also submit the true passwords of the users.

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

Part 2: Yes Salts, Still No Key Stretching [15 pts]

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, which is named `salty-digitalcorp.txt`.

Devise an attack to find the passwords of users in `salty-digitalcorp.txt`. Implement your attack in Python and submit its source code. Also submit the true passwords of the users.

Hint: You need to figure out if salts are prepended (added to the beginning) or appended (added to the end) of the passwords in DigitalCorp's system.

Part 3: Yes Salts, Yes Key Stretching [20 pts]

Assume that DigitalCorp has updated its security again, and this time, they use both salts and key stretching when storing passwords. The name of the file you stole from their servers is: `keystreching-digitalcorp.txt`.

In this scenario, while you know that key stretching was used, you do NOT know how key stretching was used, and how many iterations of key stretching was used. That means you have to try multiple combinations and multiple possible numbers of iterations in order to break the passwords. For example, the "next" hash x_{i+1} could have been computed in any one of the following ways; it is up to you to figure out which one:

$$x_{i+1} = \text{hash}(x_i + \text{password} + \text{salt})$$

$$x_{i+1} = \text{hash}(\text{password} + x_i + \text{salt})$$

$$x_{i+1} = \text{hash}(\text{password} + \text{salt} + x_i)$$

… or some other order of combination …

It is also up to you to figure out the total number of iterations of key stretching. **Hint:** Number of iterations is somewhere between 1-2000.

Devise an attack to find the passwords of users in `keystreching-digitalcorp.txt`. Implement your attack in Python and submit its source code. Also submit the true passwords of the users.

Deliverables. For this question, you need to submit:
- 3 separate py files containing your attack implementations (one file for each part 1-3).
- Supporting files containing your answers, e.g., a short report containing brief explanations of your solutions and users' passwords in each part.

Question 2: SQL Injection [total: 50 pts]

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, SQL injection vulnerability is introduced.

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

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

Installation

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

On Ubuntu, you can install them both via: *sudo apt install php php-sqlite*.
On macOS, you can install them 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.
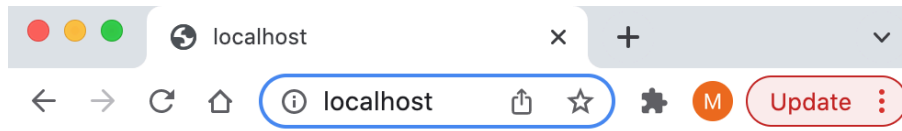
Note: To complete the SQL injection challenges, you don't need to modify the PHP code. You simply need to find the correct payload to exploit the vulnerabilities. You are encouraged to look at the PHP code to understand how the queries are constructed for each challenge.

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

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 them.

This part has 3 challenges. 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.
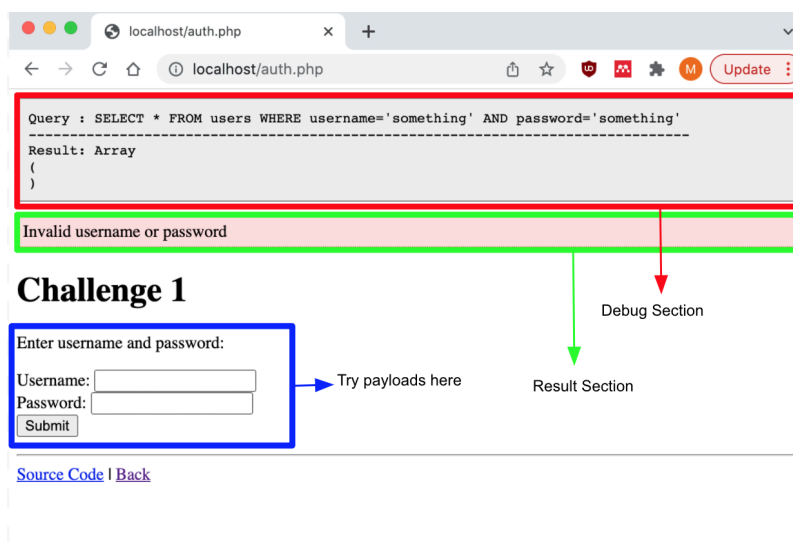
Challenge #1 [10 pts]

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.
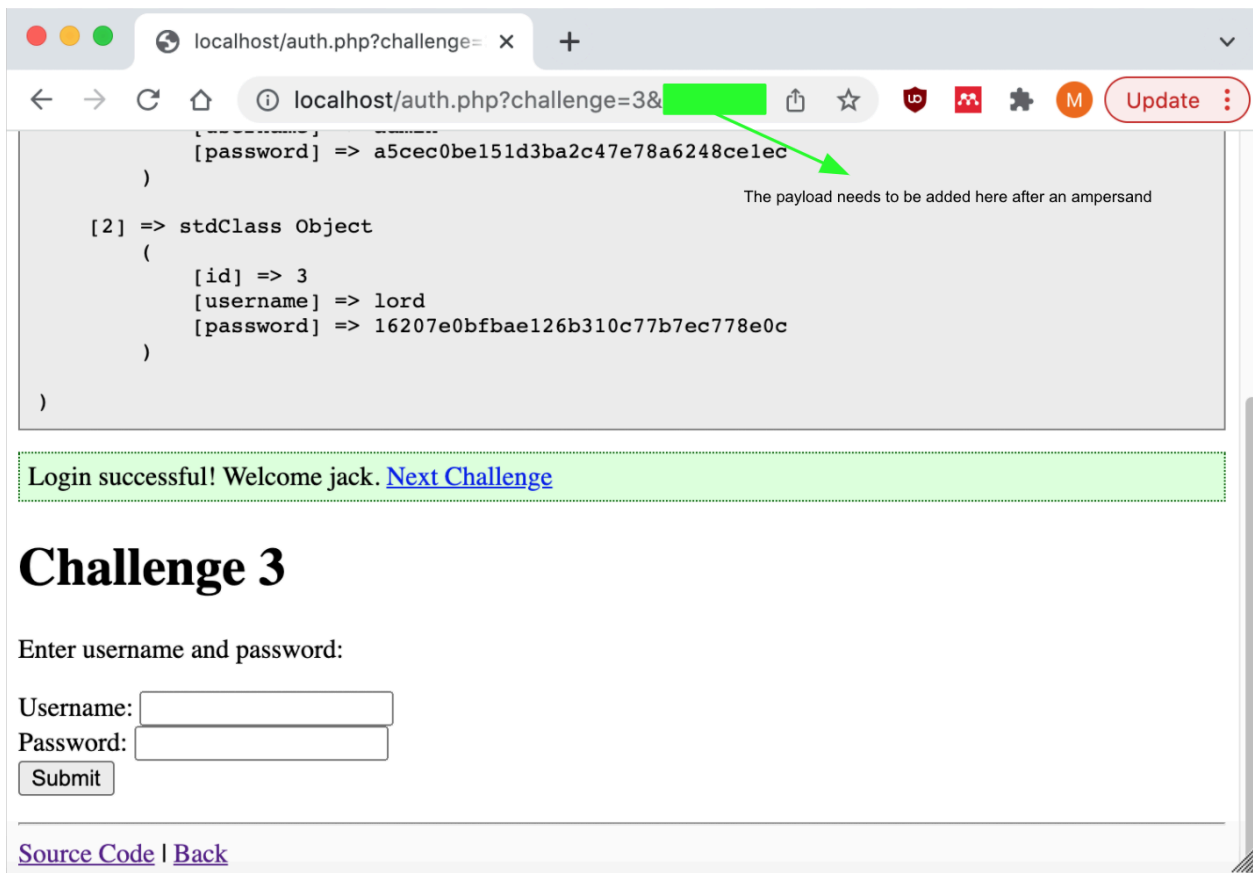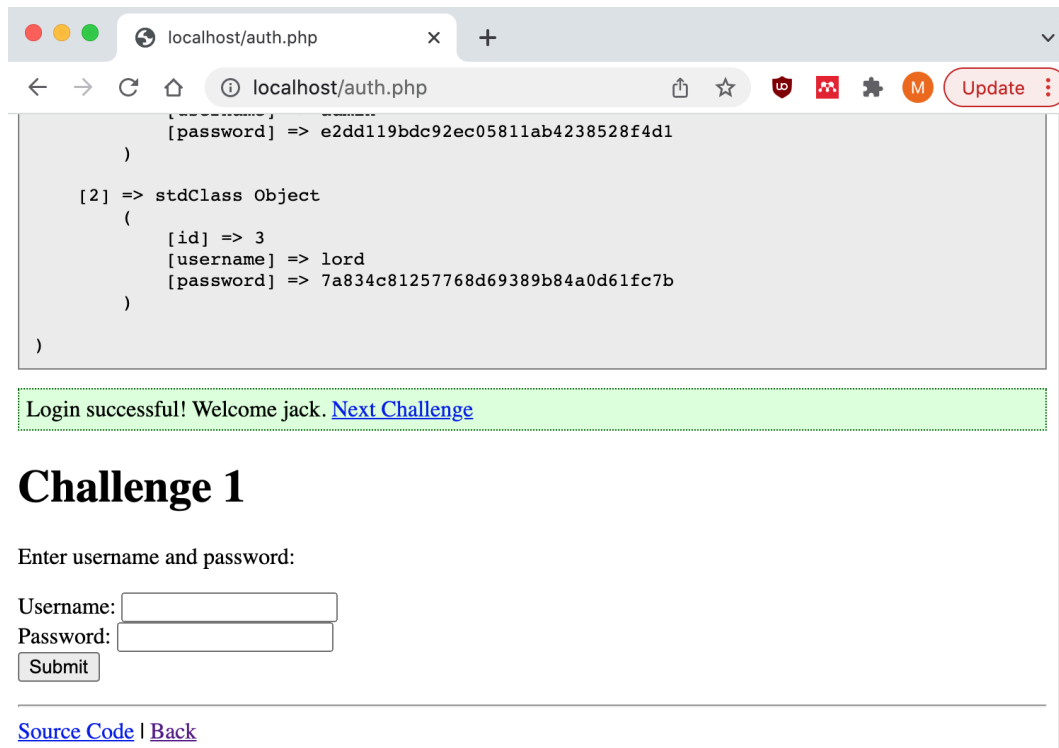

Challenge #2 [10 pts]

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 known as prepared statements) have been used to prevent the attacker from bypassing login without correct credentials. However, for the purposes of this homework assignment, the `order by` clause has been added to prepared statements which introduces a vulnerability and creates an opportunity for you to exploit. You are required to craft a payload to bypass the vulnerable part.

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: Union Based Attack [15 pts]

In this challenge, you need to extract the id, role and salary information of users who have a salary above 12,000 and age above 40. The system allows you to select employees and see their profiles, including their ids, roles, salaries and ages by default. Using union-based SQL injection, extract the information of the users, and then, put them in the place of related profile information. For example, the following output should be displayed by the web application if you choose username=admin.

Username:
ID: 2
UserID:
Role: sysadmin
Salary: 20000
Bio:
Age:

Username:
ID: 5
UserID:
Role: ceo
Salary: 40000
Bio:
Age:

Username: admin
ID: 2
UserID: 2
Role: sysadmin
Salary: 20000
Bio: Admin manages our systems effectively.
Age: 52

Note that the id, role, salary and age are not the deliverables for this challenge, but the payload you used for the injection. You should add your payload to the URL of the application. Experiment with different values and observe via debug information how it affects the generated and executed SQL query.

You should add your payload to the URL after selecting a username. In the display, all profile information of the chosen user is always shown apart from the result of your union-based SQL injection. 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.

Deliverables for SQL Injection Question. Submit a report including the following:
● The payloads that you used to cause the SQL injection. Be precise when you are writing your payloads, every character (spaces, apostrophes, dashes, etc.) can 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:

- Move all of your Python files and your report into a folder named `your KU ID`.
- **Compress this folder into a single zip file**. (Don't use compression methods other than zip.)
- Upload your zip file to Blackboard.

Notes and 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, including all of your source code and your report. If we cannot run your code because some of your code files are missing, we cannot give you points!
- This homework is an **individual assignment**. All work needs to be your own. Submissions will be checked for plagiarism (including comparing to previous years' assignments).
- **Your report should be a pdf file.** Do not submit Word files or others which may only be opened on Windows or Mac (or opening them may remove table/figure formatting).
- Only Blackboard submissions are allowed. Do not e-mail your assignment to the instructor or TAs.
- If your code does not run (e.g., syntax errors) or takes an extremely long amount of time (e.g., it takes multiple hours whereas it should take 2-3 minutes), you may get 0 for the corresponding part.

GOOD LUCK!