

CYBR 435: Cyber Risk Spring 2022

Lab Assignment #8: SQL Injection Attack

- Name only: _____
- Release date: Apr 07, 2022 (Thursday), 2:00 pm
- Due date: Apr 14, 2022 (Thursday), 2:00 pm
- Assignment should be **SUBMITTED on Blackboard before Due Date**. Other submission methods will NOT be accepted.
- **LATE Submission will NOT Be Accepted** on Blackboard since the submission link will be closed automatically after due date;
 - Additional submission for missing answer **will NOT Be Accepted**.
- It should be done INDIVIDUALLY; **Show ALL your work and evidence to support your answers**.
 - Answer only without evidence receives half credits.
- Total: 20 pts (Lab Assignment #8 will be counted 10 pts. But you will be given 20 pts if you finish it correctly and completely. The extra 10 pts are provided for you to make up the lost points from previous lab assignments.)
- The Lab is adopted from Dr. Wenliang Du at Syracuse University.

Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so they can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. SQL injection is one of the most common attacks on web applications.

In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities and demonstrate the damage that can be achieved by the attack. This lab covers the following topics:

- SQL statements: SELECT and UPDATE statements
- SQL injection

Lab Environment

This lab has been tested on the SEED Ubuntu 20.04 VM. You can download a pre-built image from the SEED website (https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/), and run the SEED VM on your own computer. However, most of the SEED labs can be conducted on the cloud, and you can follow our instruction to create a SEED VM on the cloud.

Lab Environment

We have developed a web application for this lab, and we use containers to set up this web application. There are two containers in the lab setup, one for hosting the web application, and the other for hosting the database for the web application. The IP address for the web application container is 10.9.0.5, and The URL for the web application is the following:

```
http://www.seed-server.com
```

We need to map this hostname to the container's IP address. Please add the following entry to the `/etc/hosts` file. You need to use the root privilege to change this file (using `sudo`). It should be noted that this name might have already been added to the file due to some other labs. If it is mapped to a different IP address, the old entry must be removed.

```
10.9.0.5      www.seed-server.com
```

Container Setup and Commands

Please download the Labsetup.zip file (https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/) to your VM from the lab's website, unzip it, enter the Labsetup folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved Dockerfile can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build # Build the container image
$ docker-compose up    # Start the container
$ docker-compose down  # Shut down the container

// Aliases for the Compose commands above
$ dcbuild      # Alias for: docker-compose build
$ dcup        # Alias for: docker-compose up
$ dcdown      # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "docker ps" command to find out the ID of the container, and then use "docker exec" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps      // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id> // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

MySQL Database

Containers are usually disposable, so once it is destroyed, all the data inside the containers are lost. For this lab, we do want to keep the data in the MySQL database, so we do not lose our work when we shutdown our container. To achieve this, we have mounted the mysql data folder on the host machine (inside Labsetup, it will be created after the MySQL container runs once) to the /var/lib/mysql folder inside the MySQL container. This folder is where MySQL stores its database.

Therefore, even if the container is destroyed, data in the database are still kept. If you do want to start from a clean database, you can remove this folder.

About the Web Application

We have created a web application, which is a simple employee management application. Employees can view and update their personal information in the database through this web application. There are mainly two roles in this web application: Administrator is a privilege role and can manage each individual employees' profile information; Employee is a normal role and can view or update his/her own profile information. All employee information is described in Table 1.

Table 1: Database

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

Task I: SQL Injection Attack on SELECT Statement

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers.

We will use the login page from www.seed-server.com for this task. The login page is shown in Figure 1. It asks users to provide a user name and a password. The web application authenticate users based on these two pieces of data, so only employees who know their passwords are allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

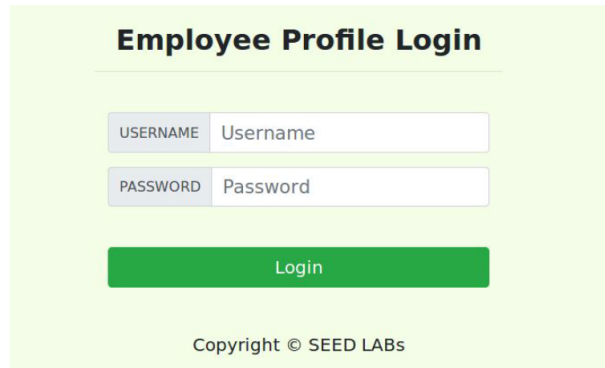


Figure 1: The Login page

To help you started with this task, we explain how authentication is implemented in the web application. The PHP code unsafe home.php, located in the /var/www/SQL_Injection directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

```
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
        nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password=' $hashed_pwd' ";
$result = $conn -> query($sql);
```

```
// The following is Pseudo Code
if(id != NULL) {
    if(name=='admin') {
        return All employees information;
    } else if (name !=NULL){
        return employee information;
    }
} else {
    Authentication Fails;
}
```

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the credential table. The SQL statement uses two variables input uname and hashed pwd, where input uname holds the string typed by users in the username field of the login page, while hashed pwd holds the sha1 hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

Task 1.1: SQL Injection Attack from Webpage

Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is admin, but you do not the password. You need to decide what to type in the Username and Password fields to succeed in the attack.

Task 1.2: SQL Injection Attack from Command Line

Your task is to repeat Task 1.1, but you need to do it without using the webpage. You can use command line tools, such as curl, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as &) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (username and Password) attached:

```
$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
```

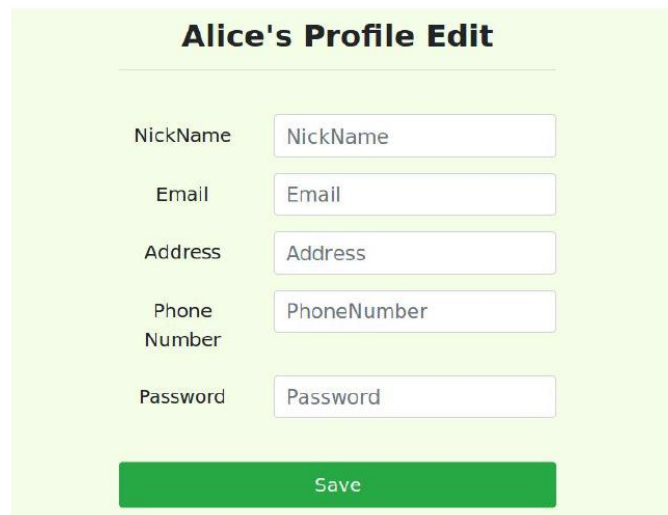
If you need to include special characters in the username or Password fields, you need to encode them properly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use %27 instead; if you want to include white space, you should use %20. In this task, you do need to handle HTTP encoding while sending requests using curl.

Task 2: SQL Injection Attack on UPDATE Statement

If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases. In our Employee Management application, there is an Edit Profile page (Figure 2) that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first.

When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in unsafe edit backend.php file is used to update employee's profile information. The PHP file is located in the /var/www/SQLInjection directory.

```
$hashed_pwd = sha1($input_pwd);  
$sql = "UPDATE credential SET  
    nickname=' $input_nickname',  
    email=' $input_email',  
    address=' $input_address',  
    Password=' $hashed_pwd',  
    PhoneNumber=' $input_phonenumber'  
    WHERE ID=$id;";  
$conn->query($sql);
```



Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

Figure 2: The Edit-Profile page

Task 2.1: Modify Your Own Salary

As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Bobby did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called salary.

Task 2.2: Modify Other People' Salary

After increasing your own salary, you decide to punish your boss Bobby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

Task 2.3: Modify Other People' Password

After changing Bobby's salary, you are still disgruntled, so you want to change Bobby's password to something that you know, and then you can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demonstrate that you can successfully log into Bobby's account using the new password. One thing worth mentioning here is that the database stores the hash value of passwords instead of the plaintext password string. You can again look at the unsafe edit backend.php code to see how password is being stored. It uses SHA1 hash function to generate the hash value of password.

Questions for the Lab

Your submission should include the following:

- I. A WORD file containing the **screenshot along with action explanation** for the following tasks
 - Task 1.1: SQL Injection Attack from Webpage [2 pts]
 - Task 1.2: SQL Injection Attack from Command Line [2 pts]
 - Task 2.1: Modify Your Own Salary [2 pts]
 - Task 2.2: Modify Other People' Salary [2 pts]
 - Task 2.3: Modify Other People' Password [2 pts]

Happy Hacking!