

Lab 7

(Due your class date of July 14/19)

In this lab, you will learn the SQL injection attack and its countermeasures. We will start with SQL review and then show the data can be mixed with SQL query code to mislead the database, which results in an unauthorized response to the attacker.

Part A: Get familiar with SQL statements

1. log into mysql: `$mysql -u root -pseedubuntu`

`/*log in username (root) and password (seedubuntu). No space between -p and the password*/`

2. Practice the following commands (change the inputs as you want):

- `show databases;` `/*this will list the existence databases*/`
- `use Users;` `/*You are now using the existing database Users*/`
- `show tables;` `/* this shows the tables of Users database*/`
- `select * from credential;` `/*this will list all the records in table credential*/`
- `select * from credential where 1=1;` `/*list all records using where */`
- `select * from credential where EID='5000' or Name='Alice';` `/*more practices on where*/`
- `update credential set Salary=10000 where Name='Alice';` `/*update Salary of Alice*/`
- `select Name, EID, SSN, Salary from credential where Name='Alice' # or 1=1;`
`/* # is for comments, after which it is ignored by SQL*/`
- please output a record (EID, Name, SSN, Salary) of **Bob** or users with Salary>90000;
`/*Bob and 4000 can be changed as for your case*/`

Provide screenshots for your experiment in Part A (you can choose different inputs for the commands).

Part B: SQL Injection on SELECT statement

`/*Lab Environment introduction.` Our SQL injection attack will target on the site <http://www.seedlabsqlinjection.com>. This site is built on your VM and is accessible from our own VM only. This is achieved by configuring `/etc/hosts` file with an entry:

```
127.0.0.1    www.seedlabsqlinjection.com
```

So whenever you type URL of this site on your browser, it directly goes to the localhost (127.0.0.1).

There are several sites using 127.0.0.1 as their IP. To distinguish them, your VM will make each site as a virtual host, by configuring the file: `000-default.conf` at directory `/etc/apache2/sites-available`. To do this, just add a record to this file (already done for you on your VM):

```
<VirtualHost *: 80>
```

```
    ServerName http://www.SeedLabSQLInjection.com
```

```
    DocumentRoot /var/www/SQLInjection
```

```
</VirtualHost>
```

The second line indicates where the site files are located. */

/* Attention: you will work **frequently** on DocumentRoot: /var/www/SQLInjection*/

1. SQL injection at the webpage **www.SeedLabSQLInjection.com**.

/* To login a user account, you need to provide your username and password. Then, webserver will execute a php file **unsafe_home.php** to decide whether to let you in. The file is lengthy but its **essential idea** is as follows.

```
$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;
}
```

*/

In this problem, you are required to use the knowledge in the lecture to login as an admin at

www.seedlabsqlinjection.com, without using any password. After you log in, you will see the list of all users' records. Take a screenshot for your login input and the result from the server. Outline the procedure of the webserver to handle your request to justify why the attack can succeed.

2. SQL injection attack on a command line.

You are required to repeat the above attack. But instead of on a webpage, you are required to use curl on the command line. Sample usage of curl to a page AAA.php with username alice and password 111 is

\$curl 'www.SeedLabSQLInjection.com/AAA.php?username=alice&Password=111'

You can also save the returned page to a html file (and to be displayed by a browser such as firefox):

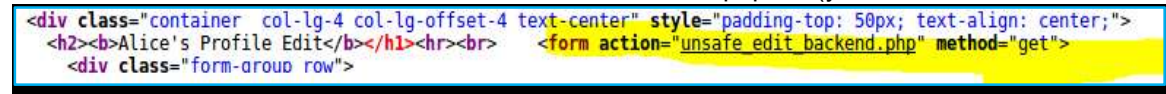
\$curl 'www.SeedLabSQLInjection.com/AAA.php?username=alice&Password=111' >BBB.html

But in your attack, if you use special symbols at the username or password, you need to encode it (not directly using it). Specifically, single quote is encoded as %27; while space is encoded as %20; # is encoded as %23. Show the screenshot of your curl command and your obtained result.

Part C. SQL Injection attack on Update statement

/* **SELECT** only gives you the right to view certain records. Attack on Update statements allows you to modify the database and hence the damage is more serious */

/* The update is done through an Edit Profile page (i.e., **unsafe_edit_frontend.php**). You can update Nickname, Email, Password and more. Check the source of php file (you will see



), the

updating data will be sent to **unsafe_edit_backend.php**, the main code of which is

```
$hashed_pwd = sha1($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);
```

It first generates query statement **\$sql** using the inputs from web browser and then sent **\$sql** to the database server. */

In this task, you do the following modification. You achieve the modification via Edit Profile page or via directly sending the request to **unsafe_edit_backend.php** through curl.

1. Modify Alice's Salary (you can provide NickName using the trick from Part B).
2. Modify another person's password (at Alice's profile edit page).

Note: if you modify through Edit Profile page, you need to log in this person's account first and then access Edit Profile page.

Part D. Counter measure: Prepared Statement.

/* In the vulnerable code below, the code is mixed with data during query.

```
$sql = "SELECT name, local, gender
    FROM USER_TABLE
    WHERE id = $id AND password = ' $pwd' ";
$result = $conn->query($sql))
```

The above code is vulnerable to SQL injection attacks. It can be rewritten to the following

```
$stmt = $conn->prepare("SELECT name, local, gender
    FROM USER_TABLE
    WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();
```

After the change, the original SQL injection attack presented above is no longer useful. A concrete example has been demonstrated in the lecture. */

Your task in this problem is to describe what is the change in `safe_home.php` in comparison with `unsafe_home.php`. Confirm that it carries the difference stated above. Then, try the attack at Part B.1 to see if you can still succeed. To do this, you need first to modify `index.html` so that

```
<form action="unsafe_home.php" method="get">
```

is changed to

```
<form action="safe_home.php" method="get">
```

Under this change, the input will be handled by `safe_home.php` (that uses the **prepare** statement above).

Give the screen shots for your findings in this problem.