Introduction to Website Database Vulnerabilities

There are three types of SQL Injection attacks:

Error based: If the application displays database errors to the user, an attacker may learn key information, e.g. database name, version, user names.

Union based: Union-based SQL injection as the name suggests abuses the union operator. The basic idea is to append the data that the attacker wants to a table that is already displayed in the page.

Blind: Even if database errors are turned off, an attacker may learn information about the database by launching blind attacks. E.g. you could send the statement "if the first name of the username is an *a*, wait 10 seconds. If the application takes 10 seconds to perform the query, the username starts with an *a*. Obviously brute force is time intensive, so this approach should be considered a last resort.

Remember that you are testing a vulnerability in a permitted, contained environment. Attempting any of these techniques on another website that is not wholly owned by you is considered a **cyber crime** and very likely to fall under **federal jurisdiction**.

Today, we will study SQL injection which tricks databases to do things that the programmers did not intend to allow. A cybersecurity analyst or tester uses SQL injection to examine their applications and to ensure there are no vulnerabilities that can be exploited.

You will log onto the following website: http://34.197.52.233/DVWA-master/index.php
This is a sample web application.

Each task is linked to its hint, which is **below this table**. But you may challenge yourself to not read them until afterwards. Either way, fill in the table with your answers first, and then supporting screenshots when applicable. Read carefully!

Task 0*. Did you log in WITHOUT the hint?	Yes/No
Task 1. Choose "SQL Injection". In the text field, enter the number `1` and hit submit. What is the output?	
Task 2. After completing Task 1, copy the resulting URL from the web browser's location bar, and paste it here.	
Task 3. Modify the URL from task 2 to access different data (from another user). What are the different user names and numbers that have data?	

Introduction to Website Database Vulnerabilities

Task 4. Write the SQL query that the webpage issues to the database in Task 2. (Click the link & read the hints for the right syntax).	
Nothing so far constitutes a database vulnerability. Let's change that.	
Task 5. Type the following in the User ID box: `1' or ''='` Note that there are two single quotes before the = sign and the keyword `or` is part of the query. Post the results and the new URL.	If you get an error, reread the instructions.
Task 6. Based on Task 4 come up with the SQL code that would have the same result as on task 5. Hint it should start with `SELECT user_id`	
Task 7. Modify your answer to Task 6 to receive the same result. (Write another SQL query.) The more different from the original on task 6 you can make it, the better.	

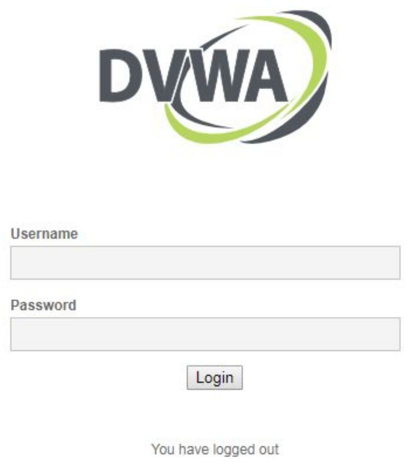
Exit Slip. Based on Task 7, what would you type now in the box on the DVWA site?	
Explain in your own words what the security vulnerability was, and why the attack worked. Include in your explanation why this is called an SQL **injection** attack.	

Read the instructions below. Even if you didn't need them, read them afterwards. There are also two extra credits.

Introduction to Website Database Vulnerabilities

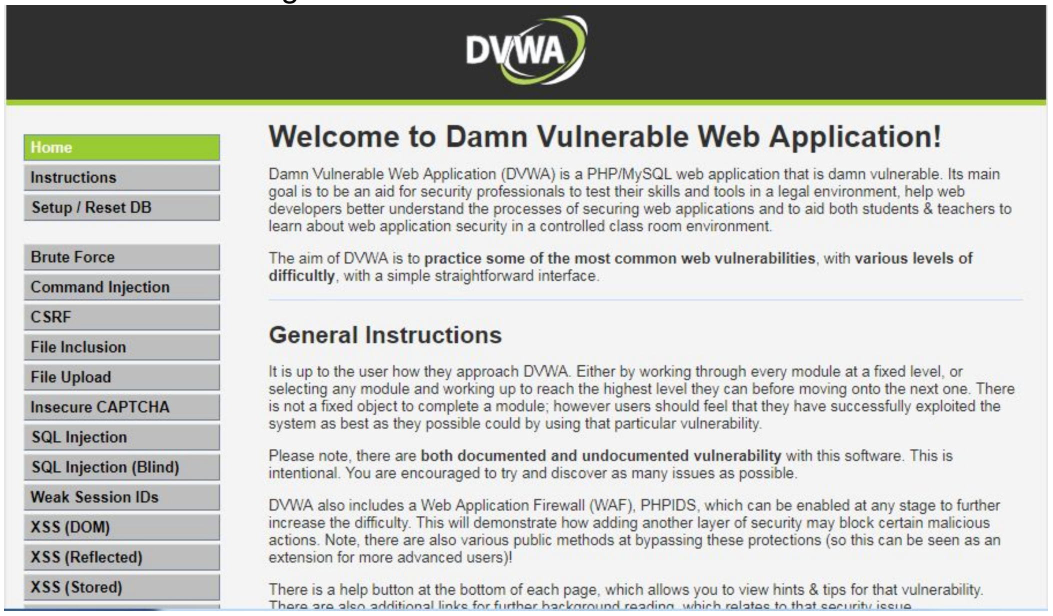
Instructions for Tasks

Point your browser to <http://34.197.52.233/DVWA-master/index.php>. You should see the following login page.



Log in using the following credentials:
username: admin
password: password

You will then face the following:



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

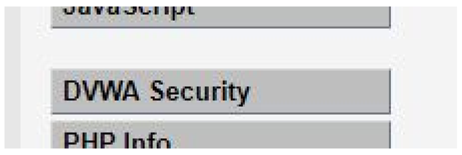
It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

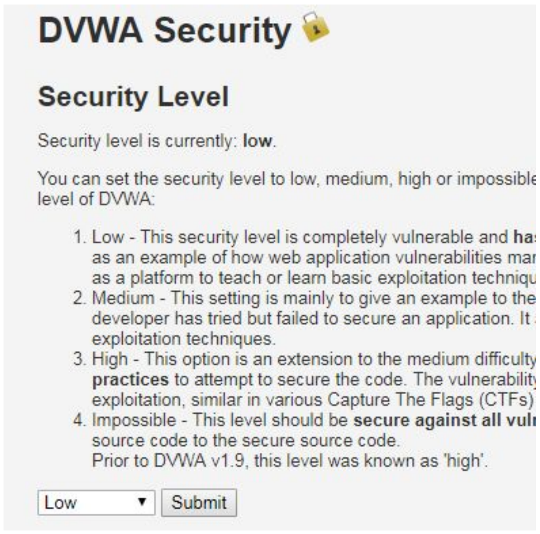
DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users)!

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

Scroll to the bottom of the page and click on DVWA Security



You will then see the following page:



DVWA Security

Security Level

Security level is currently: **low**.

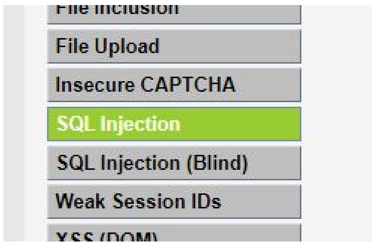
You can set the security level to low, medium, high or impossible
level of DVWA:

1. Low - This security level is completely vulnerable and **ha**
   as an example of how web application vulnerabilities mar
   as a platform to teach or learn basic exploitation techniqu
2. Medium - This setting is mainly to give an example to the
   developer has tried but failed to secure an application. It
   exploitation techniques.
3. High - This option is an extension to the medium difficulty
   practices to attempt to secure the code. The vulnerabilit
   exploitation, similar in various Capture The Flags (CTFs)
4. Impossible - This level should be **secure against all vul**
   source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Low ▾ Submit

Make sure the security level is on Low and click submit.

Then on the left hand side menu choose SQL Injection



Then you will perform the following tasks. (back to top)

Task 1. Enter in the text field the number 1 and hit submit. What is the output? (go back)

Task 2. After submitting the number 1, paste the resulting url from the web browser's address bar. (go back)

Task 3. Modify the URL from task 2 to access different data (from another user). What are the different user names and numbers that have data?

Task 4: When you submit the form, the webpage issues a query to the database behind the scenes to retrieve matching user info. The command will be in the following form:

```
SELECT user_id, first_name, last_name FROM users WHERE user_id='<query>';
```

Hint: The field names being returned are user_id, First_Name, and Last_Name, from the table users, by checking the field named user_id. Replace the angle brackets (and what's inside) with your actual input. (go back)

Task 5. Now nothing up to this point constitutes a database vulnerability (although Task 3 is undesirable. Why?). All that has happened is that the web page is instructing the database to perform the following command:

```
SELECT user_id, first_name, last_name FROM users WHERE user_id='<query>';
```

What is wrong is that you can simply identify where to inject a change to make the database do what you want. In this case you change the 1 in the URL to a 2 and so forth.

So we can perform an injection by trying to change the 1. (go back)

Type the following in the User ID box:
`1' or ''='`
Note that there are two single quotes before the = sign and the keyword `or` is part of the query.

Post the contents and the resulting URL

What has just happened is that we injected a different value that caused the database to perform a different query. Now we found a vulnerability. (go back)

Task 6. Based on Task 4 come up with the SQL code that would have the same result as on task 5.

Hint it should start with `SELECT First_Name`
(go back)

Task 7. Modify your answer to Task 6 to receive the same result. (Write another SQL query.) The more different from the original on task 6 you can make it, the better. Try different combinations.(go back)

Exit Ticket: See Tasks #5-7 again.

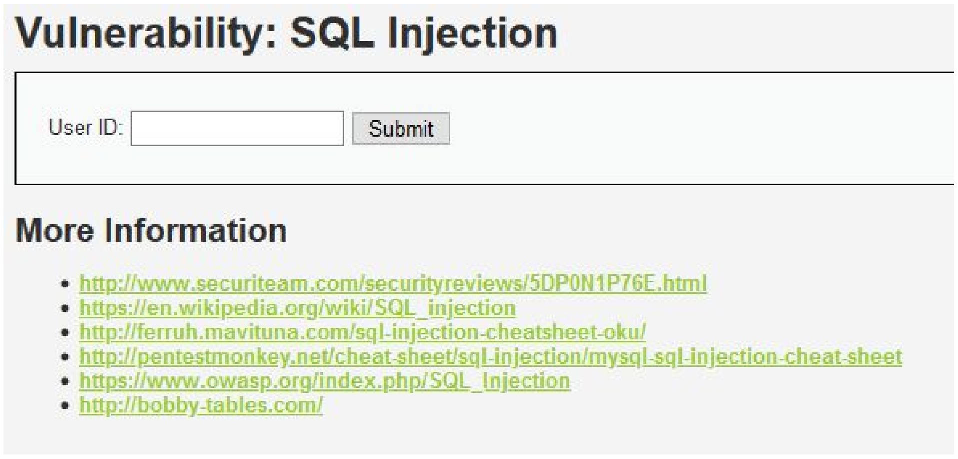
Extra Credit Challenge #1:

This will take effort and dedication.
You may have noticed that the DVWA app (webpages and databases) is hosted at the same IP address as what you use to log into the remote server to do unrelated SQL work. Therefore, you may be able to find information about the DVWA database on which the app is based.

The challenge, should you accept it, is **to be first** to take over the DVWA database. Populate it with the names of whoever executed the commands for the takeover. Email the info to <Teacher's Name> at <Teacher's Email Address>

Extra Credit Challenge #2:

You may have noticed links for more details on various vulnerabilities. There are also many other categories of site vulnerabilities, each with its own set of resources.

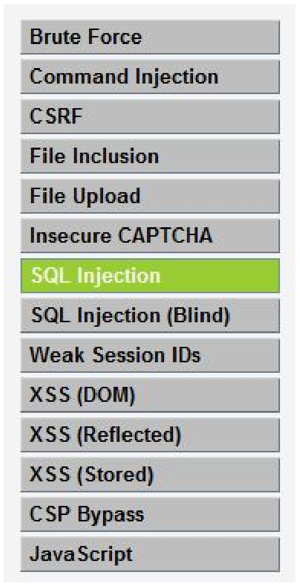


Vulnerability: SQL Injection

User ID: Submit

More Information

- http://www.securiteam.com/securityreviews/5DP0N1P76E.html
- https://en.wikipedia.org/wiki/SQL_injection
- http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
- http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
- https://www.owasp.org/index.php/SQL_Injection
- http://bobby-tables.com/



Research any of them for an activity we can do in class, or for next year’s class. Write up an explanation of what you found and what could be done in class.