Before You Begin

Welcome! The Virtual Security Cloud Labs are your opportunity to gain valuable hands-on experience with professional-grade tools and techniques as you work through the guided lab exercises provided in the on-screen lab manual. The use of virtualization enables you to perform all of the tasks in the lab manual in a live environment without putting your personal device or institution's assets at risk.

Before you begin the guided lab exercises, please review the following preparation checklist.

1. **Run the** System Checker. The System Checker will confirm that your browser and network connection are ready to support virtual labs.

2. **Review the** Common Lab Tasks document. This document provides an overview of the virtual lab environment and outlines several of the recurring tasks you may need to complete your lab exercise.

3. **When you've finished, use the Disconnect button to end your session and create a StateSave**. To end your lab session and save your work, click the Disconnect button in the upper-right corner of the Lab View toolbar. When prompted, assign a name for your StateSave (we recommend using the Section, Part, and Step number where you stopped) and click Continue. Please note that a StateSave will preserve any changes written to disk in your lab session. A StateSave will not preserve any open windows or active processes, similar to restarting your computer.
   If you close your browser window without disconnecting, your lab session will automatically end after 5 minutes.

4. Technical Support **is here to help!** Our technical support team is available 24/7 to help troubleshoot common issues.
   Please note that the 24/7 support team is Level 1 only, and cannot assist with questions about lab content or the array of software used in the labs. If you believe you've identified an error in the lab guide or a problem with the lab environment, your ticket will be escalated to the Jones & Bartlett Learning product team for review. In the meantime, we recommend resetting the lab (Options > Reset) or reaching out to your instructor for assistance.

Introduction

Web application developers and software developers are responsible for the secure coding and testing of their application. Database developers and administrators are responsible and accountable for ensuring data integrity by following best security practices and ensuring regular backups of the database are performed. System engineers and security practitioners help ensure and maintain web application security through regular penetration testing.

Penetration testing of web applications and web servers is a critical step in ensuring the confidentiality, integrity, and availability (CIA) of the web application and service. If e-commerce or privacy data is entered into the web application, the company is under additional compliance laws and standards to ensure the confidentiality of customer data. Penetration testing should be performed on a regular schedule and whenever the web application and service is modified. The organization's security policy should dictate that no production web application, whether it resides inside or outside of the firewall, be implemented without proper penetration testing and security hardening. This is especially critical if your organization is under a compliance law and your web application requires inputting of private customer data.

Two of the most common and most preventable web attacks are cross-site scripting (XSS) and SQL Injection. Hackers use XSS to execute arbitrary scripts through the web browser. XSS becomes possible when a web form allows HTML or JavaScript code as valid input. Likewise, SQL Injection allows valid (albeit odd looking) SQL commands to run within a web form. Regular testing and secure software development practices can reduce the likelihood of exposure to attack.

In this lab, you will perform simple tests to verify a cross-site scripting (XSS) exploit and an SQL injection attack using the Damn Vulnerable Web Application (DVWA): a tool left intentionally vulnerable to aid security professionals in learning about web security. You will use a web browser and some simple command strings to identify the IP target host and its known vulnerabilities, and then attack the web application and web server using cross-site scripting (XSS) and SQL injection to exploit the sample web application running on that server.

Learning Objectives

Upon completing this lab, you will be able to:

1. Identify web application and web server backend database vulnerabilities as viable attack vectors

2. Develop an attack plan to compromise and exploit a website using cross-site scripting (XSS) against sample vulnerable web applications

3. Conduct a manual cross-site scripting (XSS) attack against sample vulnerable web applications

4. Perform SQL injection attacks against sample vulnerable web applications with e-commerce data entry fields

Lab Overview

Each section of this lab is assigned at your instructor's discretion. Please consult your instructor to confirm which sections you are required to complete for your lab assignment.

SECTION 1 of this lab has two parts, which should be completed in the order specified.

1. In the first part of the lab, you will perform a cross-site scripting (XSS) attack against a sample vulnerable web application.

2. In the second part of the lab, you will perform SQL injection attacks against the same web application.

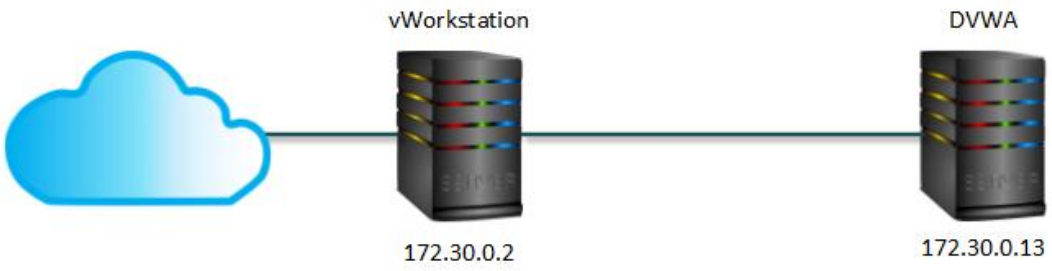
SECTION 2 of this lab allows you to apply what you learned in **SECTION 1** with less guidance and different deliverables, as well as some expanded tasks and alternative methods. You will also practice with different SQL injection attacks.

Finally, you will explore the virtual environment on your own in **SECTION 3** of this lab. You will answer questions and complete challenges that allow you to use the skills you learned in the lab to conduct independent, unguided work, similar to what you will encounter in a real-world situation.

Topology

This lab contains the following virtual devices. Please refer to the network topology diagram below.

- vWorkstation (Windows Server 2016)
- DVWA (Debian Linux)



Tools and Software

The following software and/or utilities are required to complete this lab. Students are encouraged to explore the Internet to learn more about the products and tools used in this lab.

- Damn Vulnerable Web Application (DVWA)

Deliverables

Upon completion of this lab, you are required to provide the following deliverables to your instructor:

SECTION 1:

1. Lab Report file including screen captures of the following;

- XSS form showing *yourname*;
- XSS form showing the vulnerability pop-up alert;
- SQL injection user hash information;

2. Files downloaded from the virtual environment:

- none;

3. Any additional information as directed by the lab:

- none;

4. Lab Assessment (worksheet or quiz - see instructor for guidance)

SECTION 2:

1. Lab Report file including screen captures of the following:

- XSS form showing *yourname*;
- XSS form showing the vulnerability pop-up alert;
- submissions that return a response for scripts used to determine the number of columns in a table;
- submissions that return a response for scripts seeking the correct spelling of a field name;
- submissions that return a response for scripts that search for a possible hit on the database's characters;
- SQL injection user hash information;

2. Files downloaded from the virtual environment:

- none;

3. Any additional information as directed by the lab:

- none;

SECTION 3:

1. Analysis and Discussion
2. Tools and Commands
3. Challenge Exercise

Section 1: Hands-On Demonstration

Note: In this section of the lab, you will follow a step-by-step walk-through of the objectives for this lab to produce the expected deliverable(s).

1. On your local computer, **create** the **Lab Report file**.
   Frequently performed tasks, such as how to create the Lab Report file, make screen captures, and download files from the lab, are explained in the Common Lab Tasks document. You should review these tasks before starting the lab.

2. **Proceed** with **Part 1**.

Part 1: Cross-Site Scripting Attacks

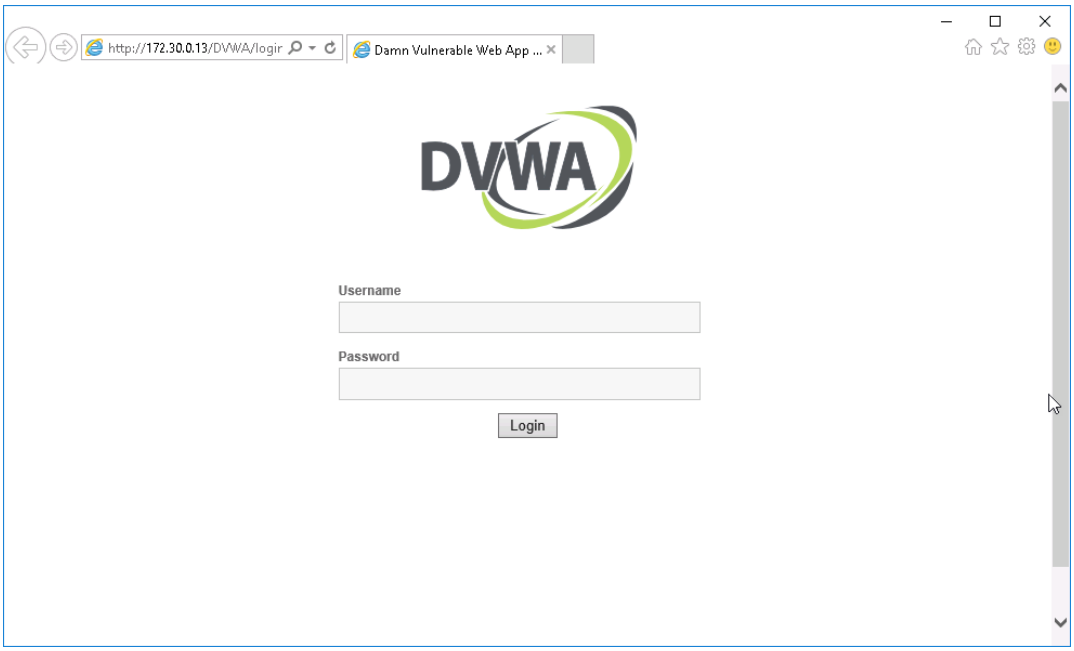
Note: Cross-site scripting (XSS) is the malicious insertion of scripting code to extract data or modify a web site's code, application, or content. There are two common varieties of XSS vulnerabilities: persistent (stored) and non-persistent (reflected). In a persistent XSS attack, the most serious variety, data that can modify how applications or services operate is downloaded (stored) onto the targeted server. The victim triggers the attack. In a non-persistent attack, the most common variety, the attacker attempts to use scripting commands in the URL itself, or through a device, such as a web form, to gain administrator, or some other elevated level of user privileges in an attempt to force the victim's server to display the desired data on-screen.

In this part of the lab, you will use the Damn Vulnerable Web Application (DVWA) to perform an attack that exploits a cross-site scripting (XSS) vulnerability. Before attempting the script tests, you must set the security level of DVWA to low which exposes the vulnerabilities.

1. From the vWorkstation taskbar, **click** the **Internet Explorer icon** to open the Internet Explorer browser.

   You can access the DVWA using any Internet browser, but the steps in this lab will use the Internet Explorer browser.

2. If necessary, in the browser's address box, **type** `http://172.30.0.13/DVWA` and **press Enter** to open the DVWA login page.



DVWA login screen

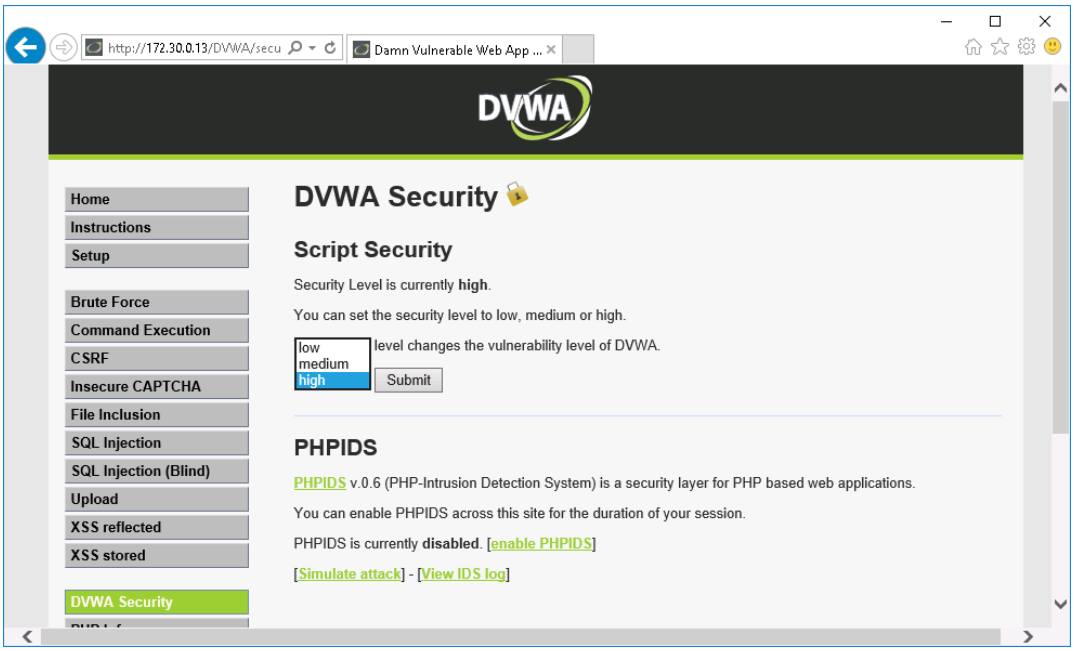
3. At the DVWA login page, **type** the following credentials and **click Login** to continue.

    - Username: `admin`
    - Password: `password`

    If prompted to save the password, **click Not for this site**.

4. From the DVWA navigation menu, **click** the **DVWA Security button** to open the DVWA Security settings.

5. From the Script Security drop-down menu, **select low** and **click Submit** to change the security level.



Changing the script security level in DVWA

6. From the DVWA navigation menu, **click** the **XSS reflected button** to open the XSS reflected page.

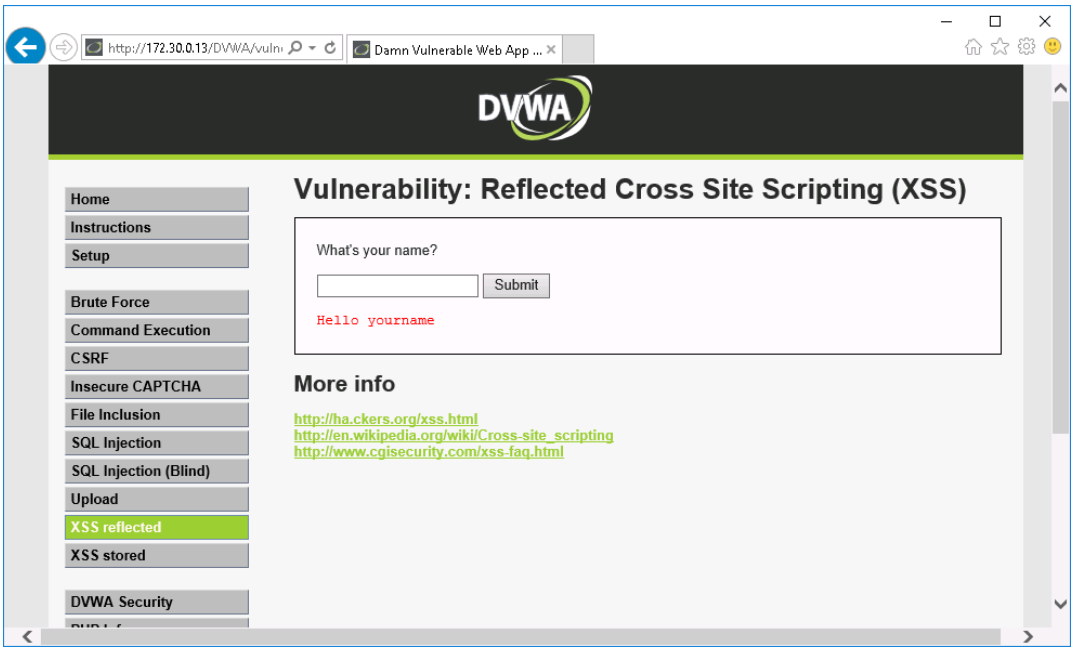
   XSS vulnerabilities are commonly found in web forms that send and retrieve data to databases via HTML.

Note: The scripts in this lab are all typed in clear text to make it easier for you to understand the process. Often hackers will use hexadecimal character strings instead of clear text to make the scripts even harder to detect. Database administrators should monitor their SQL databases for unauthorized or abnormal SQL injections and write scripts that send off alarms as well as sending network management alerts. Additional safeguards can include encrypting the data elements that reside in long-term storage of the SQL database.

7. In the What's your name? box, **type** ***yourname*** (replacing *yourname* with your own first name) and **click Submit**.

   The web form is intended to take the name you entered and repeat it back to you in a friendly

welcome. If prompted to remember web form entries, **click No**.

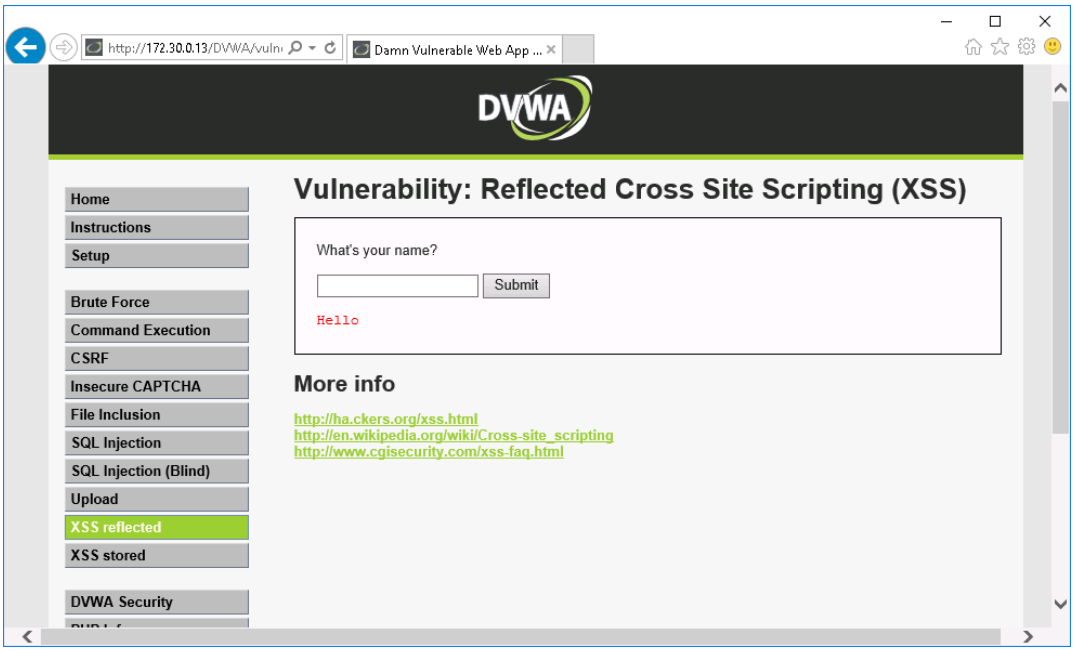


Intended output

8. **Make a screen capture** showing ***yourname*** and **paste** it into the Lab Report file.

9. In the What's your name? box, **type** `<this is a test>` and **click Submit**.

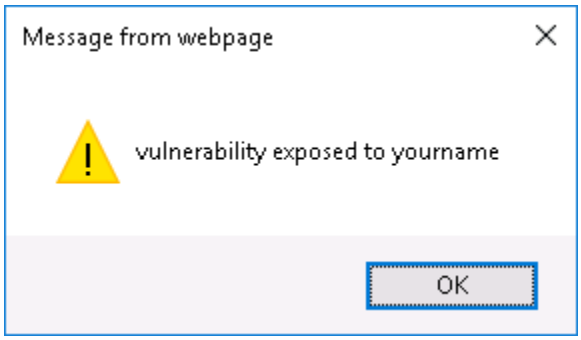
   Doing this passes in scripting tags (<>) and in this case, because the web server did not respond with an error message, you know that the web server has accepted the scripting tags. This opens the door to inputting scripts into the web form, which can be confirmed in the next step.



Reflected XSS vulnerability identified

10. In the What's your name? box, **type** `<script>alert('vulnerability exposed to yourname');</script>`, replacing *yourname* with own name, and **click Submit**.

    Note that the quote marks are straight single quotes, not curly opening and closing quotes. The web form will process the script and return a pop-up alert window containing the message you specified.



Alert window processed by the form

11. **Make a screen capture** showing the **exposed vulnerability** and **paste** it into the Lab Report file.

12. **Click OK** to dismiss the alert message.

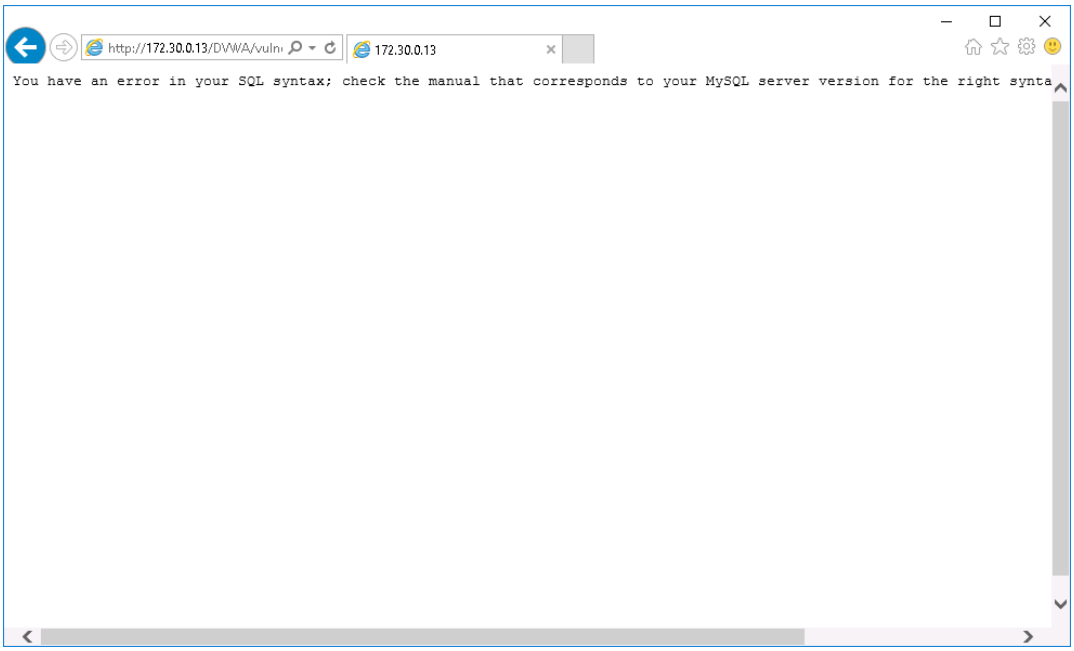
Part 2: SQL Injection Attacks

Note: Poorly designed or improperly secured web forms can be exploited to output passwords, credit card information, and other data. In the next steps, you will insert a series of SQL statements into a web form to find and then exploit a SQL injection vulnerability. SQL Injection attacks are used to extract privacy data elements out of a database by inserting real SQL query commands into web forms. Many of these commands will not return any errors or produce any results, but determined hackers will continue probing until they find the data they were seeking.

1. From the DVWA navigation menu, **click** the **SQL Injection button** to open the SQL Injection page.

2. In the User ID box, **type** `O'Malley` and **click Submit** to determine if the web server is vulnerable to SQL injection attacks.

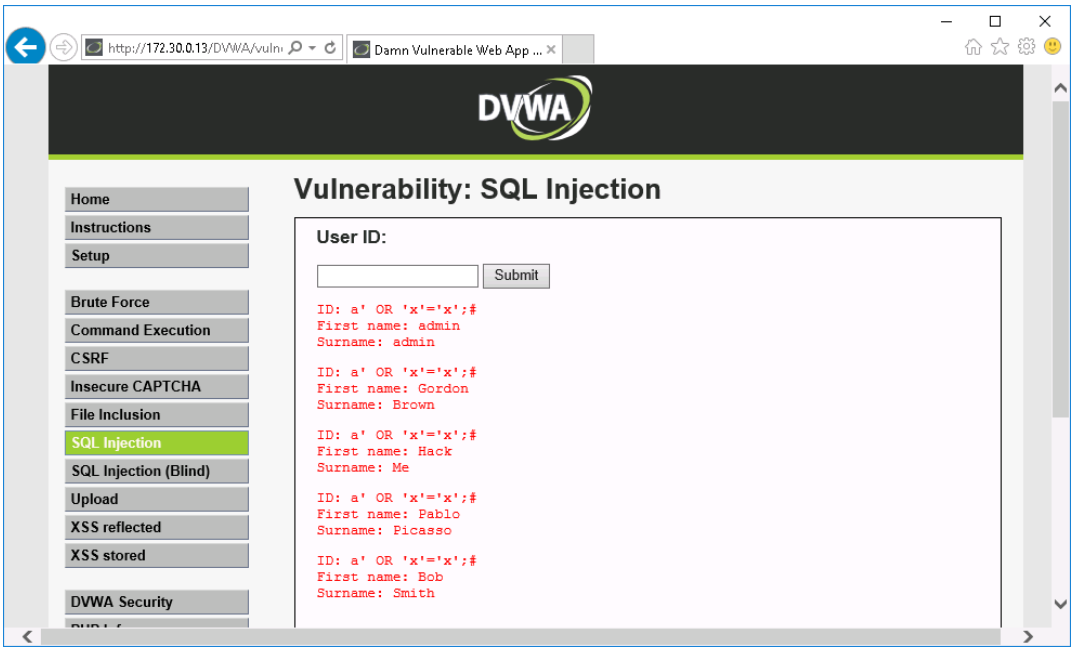
   The web form was unable to handle the special character of an apostrophe. Sometimes programmers forget to include script handling for special characters in data input forms. This type of error can make an application vulnerable to SQL injection.



SQL error

3. On the browser toolbar, **click** the **Back button** to return to the SQL Injection page in DVWA.

4. In the User ID box**, type** `a' OR 'x'='x';#` and **click Submit** to return the first and last names of everyone in the application's database.



Results from the SQL injection test

5. In the User ID box, **type** `a' ORDER BY 1;#` and **click Submit**.

   - If there is no error, indicating a SQL injection vulnerability, proceed to the next step.
   - If you see an error statement, **click** the browser's **Back button** and proceed to step 8.

6. In the User ID box, **type** `a' ORDER BY 2;#` and **click Submit**.

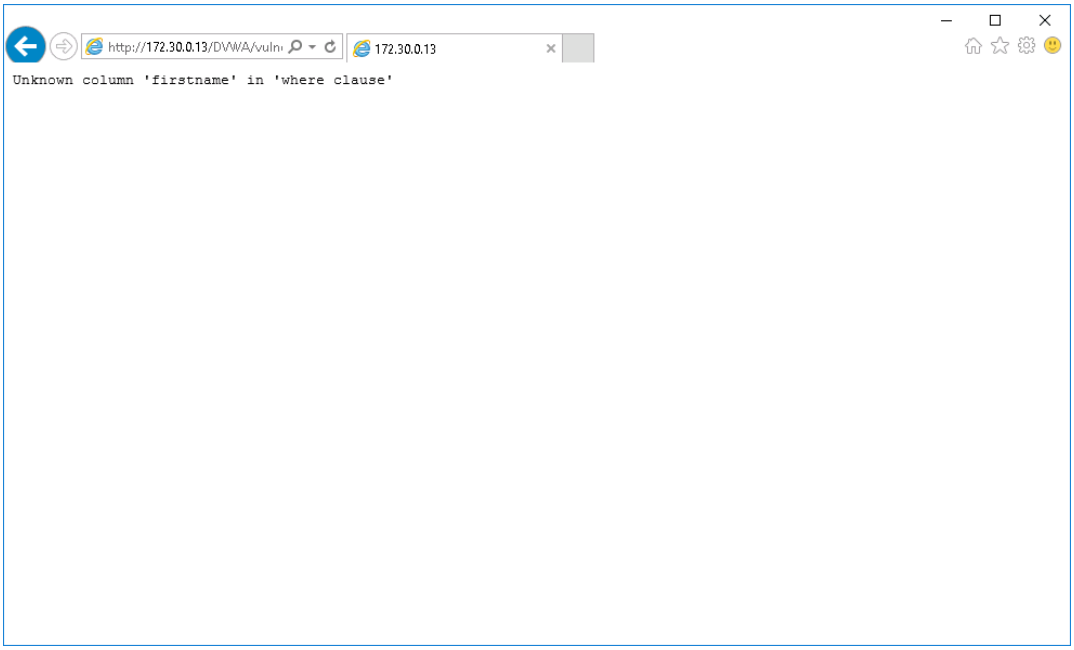
   - If there is no error, indicating a SQL injection vulnerability, proceed to the next step.
   - If you see an error statement, **click** the browser's **Back button** and proceed to step 8.

7. In the User ID box, **type** `a' ORDER BY 3;#` and **click Submit**.

   - If there is no error, indicating a SQL injection vulnerability, proceed to the next step.
   - If you see an error statement, **click** the browser's **Back button** and proceed to step 8.

8. In the User ID box, **type** `a' OR firstname IS NULL;#` and **click Submit**.

The error message indicates that there is no field named *firstname*. That doesn't mean that the information is not in the database, so continue with the next step to try again.



Results from the "firstname" test

9. On the browser toolbar, **click** the **Back button** to return to the SQL Injection page in DVWA.

10. In the User ID box, **type** `a' OR first_name IS NULL;#` and **click Submit** to try another common spelling for the name of this field.

    The lack of an error message indicates that the field name is now spelled correctly.

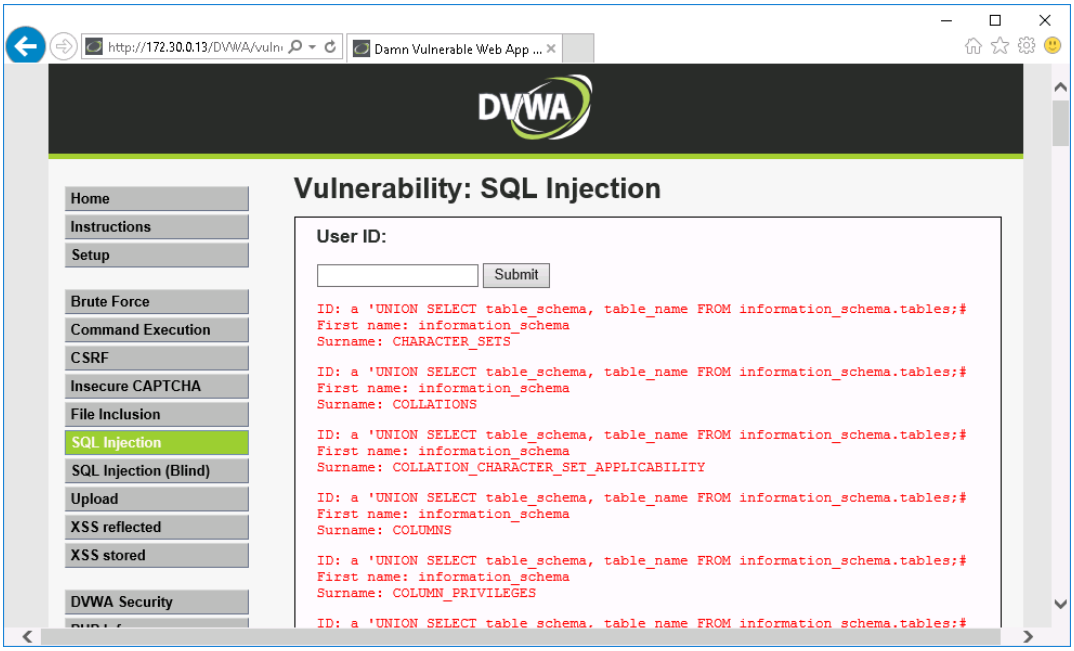
11. In the User ID box, **type** `a' OR database() LIKE 'DB';#` and **click Submit** to search for a possible hit on the database's characters.

12. In the User ID box, **type** `a' OR database() LIKE 'd%';#` and **click Submit**.

Like the previous script, this one searches for a possible hit on the database's characters, but the percent sign (%) will split the fields.

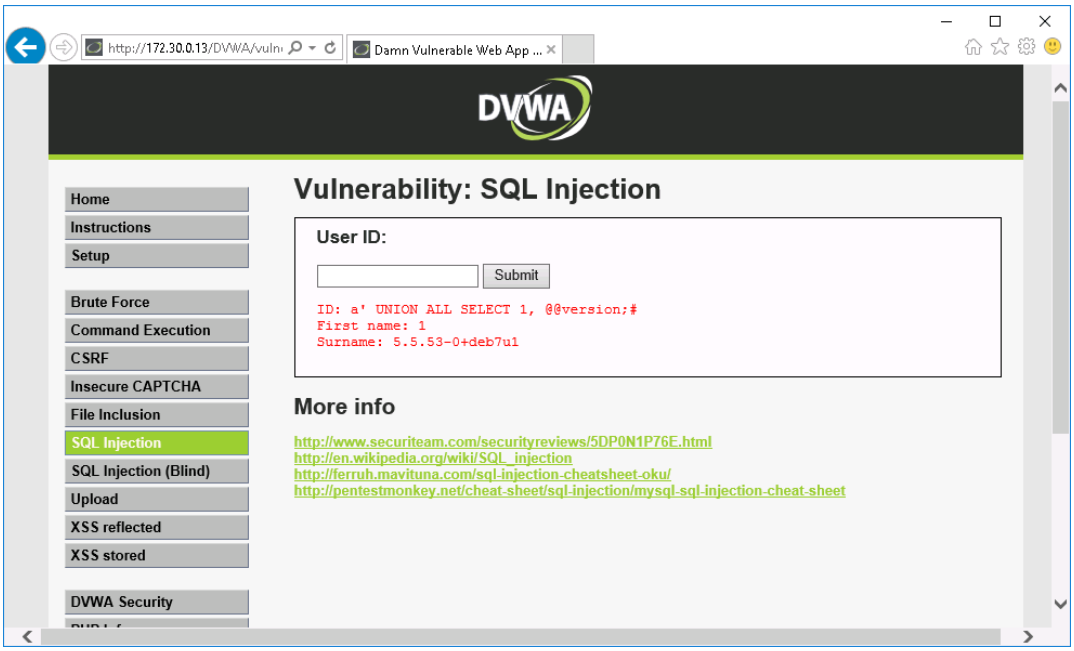
13. **Click** the browser's **Back button**.

14. In the User ID box, **type** `a' UNION SELECT table_schema, table_name FROM information_schema.tables;#` and **click Submit** to return all of the table and column names in the database.



Results from the "table_schema, table_name" test

15. In the User ID box, **type** `a' UNION ALL SELECT 1, @@version;#` and **click Submit** to return information about the version of SQL being used on the server.



Results from the “@@version” test

16. In the User ID box, **type** `a' UNION ALL SELECT system_user(), user();#` and **click Submit** to return information about the user name that you are using as you make queries on the server.

17. In the User ID box, **type** `a' UNION ALL SELECT user, password FROM mysql.user;# priv;# '` and **click Submit** to display a hash for the user to the backend database.

18. **Make a screen capture** showing the **hash information** and **paste** it into the Lab Report file.

19. In the User ID box, **type** `'UNION SELECT 'test', '123' INTO OUTFILE 'test1.txt` and **click Submit**.

Note: This script, if successful, indicates that data can be written to a file, and later downloaded. How do you know that it is successful? If the file can be created, there will be no error message and the script will pass through the Submit button and clear the User ID box. If the file cannot be created, either because the file already exists or because the script was faulty or the developer prevented it, you will receive an error or the script will remain in the User ID box. Together with the information you

gathered in earlier tests, you now have a user with elevated permissions, user IDs, passwords, and the table structure where all of this data is being held—in other words, an injectable database.

20. **Close** the **browser** to exit the DVWA.

Note: This completes Section 1 of this lab. There are no deliverable files for this section.

Section 2: Applied Learning

Note: SECTION 2 of this lab allows you to apply what you learned in **SECTION 1** with less guidance and different deliverables, as well as some expanded tasks and alternative methods. You will also practice with different SQL injection attacks.

Please confirm with your instructor that you have been assigned Section 2 before proceeding.

1. On your local computer, **create** the **Lab Report file**.
   Frequently performed tasks, such as how to create the Lab Report file, make screen captures, and download files from the lab, are explained in the Common Lab Tasks document. You should review these tasks before starting the lab.

2. If you already completed Section 1 of this lab, you will need to reset the virtual environment before beginning Section 2. To reset the virtual environment, complete one of the following options.
   a. **Click Options > Reset Lab** to restore all virtual machines to their base state. This will take several minutes to complete. If you do not see the vWorkstation desktop after five minutes, **click Options > Reload Lab** to reload your lab connection.

   b. **Click Disconnect**, then **select Discard Changes** to end your lab session without creating a StateSave. If you previously created a StateSave, delete the StateSave at the launch page, then start a new lab session.

3. **Proceed** with **Part 1**.

Part 1: Cross-Site Scripting Attacks

Note: Cross-site scripting (XSS) is the malicious insertion of scripting code to extract data or modify a web site's code, application, or content. There are two common varieties of XSS vulnerabilities: persistent (stored) and non-persistent (reflected). In a persistent XSS attack, the most serious variety, data that can modify how applications or services operate is downloaded (stored) onto the targeted server. The victim triggers the attack. In a non-persistent attack, the most common variety, the attacker attempts to use scripting commands in the URL itself, or through a device, such as a web

form, to gain administrator, or some other elevated level of user privileges in an attempt to force the victim's server to display the desired data on-screen.

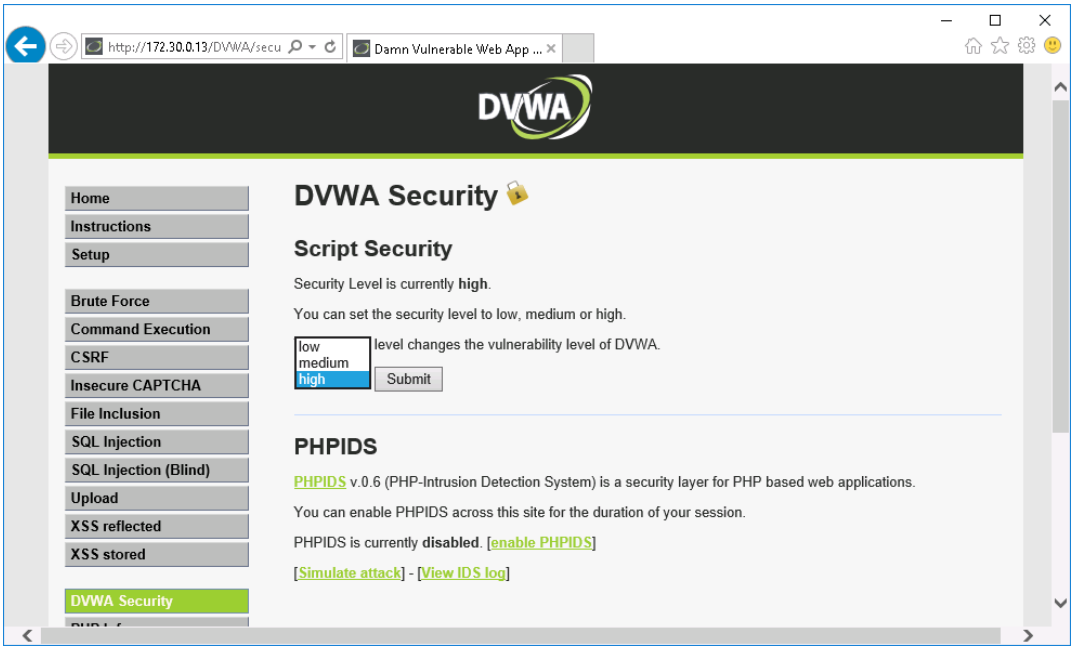
In this part of the lab, you will use the Damn Vulnerable Web Application (DVWA) to perform an attack that exploits a cross-site scripting (XSS) vulnerability. Before attempting the script tests, you must set the security level of DVWA to low which exposes the vulnerabilities.

1. **Open Internet Explorer** and open the Damn Vulnerable Web Application at **`http://172.30.0.13/DVWA`**, then **log in** with the following credentials:

   - Username: **`admin`**
   - Password: **`password`**

   You can access the DVWA using any web browser, but the steps in this lab will use the Internet Explorer browser.

2. **Set** the **DVWA Security level** to **low**.



Changing the script security level in DVWA

3. **Navigate** to the **XSS reflected** page.

   XSS vulnerabilities are commonly found in web forms that send and retrieve data to databases via HTML.

Note: The scripts in this lab are all typed in clear text to make it easier for you to understand the process. Often hackers will use hexadecimal character strings instead of clear text to make the scripts even harder to detect. Database administrators should monitor their SQL databases for unauthorized or abnormal SQL injections and write scripts that send off alarms as well as sending network management alerts. Additional safeguards can include encrypting the data elements that reside in long-term storage of the SQL database.

4. In the web form, **submit** ***yourname*** (replacing *yourname* with your own first name).

   The web form is intended to take the name you entered and repeat it back to you in a friendly welcome.

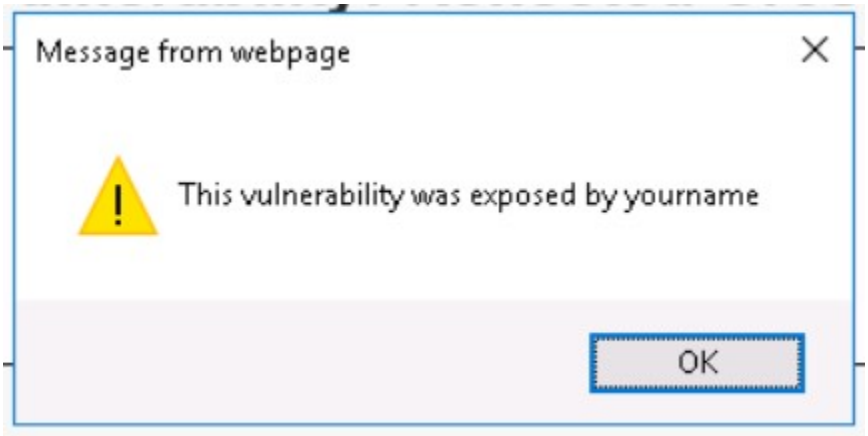
5. **Make a screen capture** showing ***yourname*** and **paste** it into the Lab Report file.

6. In the web form, **submit** `<this is a test>` to identify if the web server has a XXS vulnerability.

   Doing this passes in scripting tags (<>) and in this case, because the web server did not respond with an error message, you know that the web server has accepted the scripting tags. This opens the door to inputting scripts into the web form, which can be confirmed in the next step.

7. In the web form, **submit** `<script>alert('This vulnerability was exposed to yourname');</script>`, replacing *yourname* with your own name.

   Note that the quote marks are straight single quotes, not curly opening and closing quotes.



Alert window processed by the web form

8. **Make a screen capture** showing the **exposed vulnerability** and **paste** it into the Lab Report file.

9. **Dismiss** the **alert**.

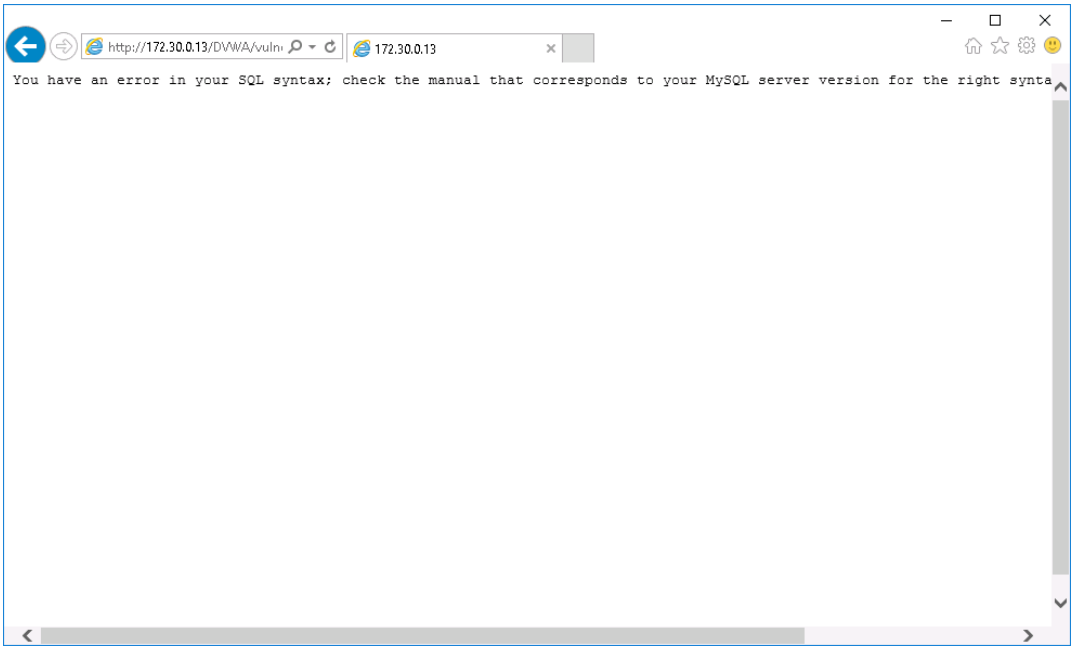
Part 2: SQL Injection Attacks

Note: Poorly designed or improperly secured web forms can be exploited to output passwords, credit card information, and other data. In the next steps, you will insert a series of SQL statements into a web form to find and then exploit a SQL injection vulnerability. SQL Injection attacks are used to extract privacy data elements out of a database by inserting real SQL query commands into web forms. Many of these commands will not return any errors or produce any results, but determined hackers will continue probing until they find the data they were seeking.

1. **Navigate** to the **SQL Injection** page.

2. In the web form, **submit** `O'Malley` to determine if the web server is vulnerable to SQL injection attacks.

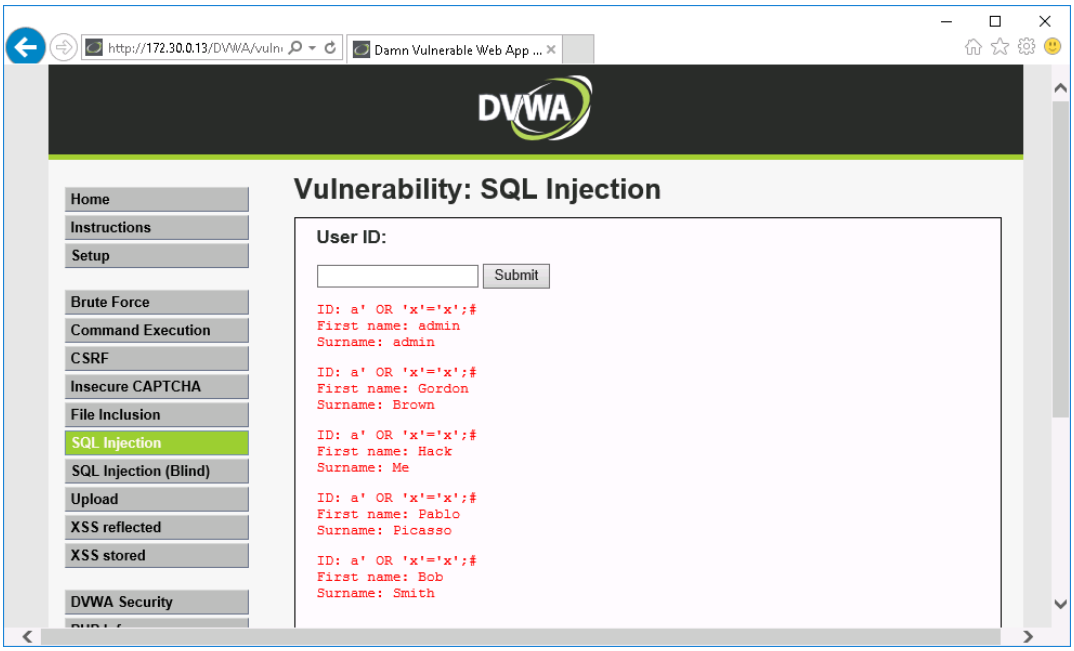
   The web form was unable to handle the special character of an apostrophe. Often, programmers forget to include script handling for special characters in data input forms. This type of error can make an application vulnerable to SQL injection.



SQL error

3. In the web form, **submit** `a' OR 'x'='x';#`.

   This script will return the first and last names of everyone in the application's database.



SQL injection test

4. **Submit** the following scripts in the web form and **make a screenshot** showing any **submissions that return a response** and **paste** it into the Lab Report file.

   These scripts are used to determine the number of columns in a table.

   - **`a' ORDER BY 1;#`**
   - **`a' ORDER BY 2;#`**
   - **`a' ORDER BY 3;#`**

5. **Submit** the following scripts in the web form and **make a screenshot** showing any **submissions that return a response** and **paste** it into the Lab Report file.

   These scripts are seeking the correct spelling of a field name.

   - **`a' OR firstname IS NULL;#`**
   - **`a' OR first_name IS NULL;#`**

6. **Submit** the following scripts in the web form and **make a screenshot** showing any

submissions that return a response and **paste** it into the Lab Report file.

These scripts search for a possible hit on the database's characters. The percent sign (%) is used to split the returned fields.

- `a' OR database() LIKE 'DB';#`
- `a' OR database() LIKE 'd%';#`

7. In the web form, **submit** `a' UNION SELECT table_schema, table_name FROM information_schema.tables;#` to return all of the table and column names in the database.

8. In the web form, **submit** `a' UNION ALL SELECT 1, @@version;#` to return information about the version of SQL being used on the server.

9. In the web form, **submit** `a' UNION ALL SELECT system_user(), user();#` to return information about the user name that you are using as you make queries on the server.

10. In the web form, **submit** `a' UNION ALL SELECT 1, @@datadir;#` to return the location of the database on the server.

11. In the web form, **submit** `a' UNION ALL SELECT user, password FROM mysql.user;# priv;#` to display a hash to the backend database for the current user.

12. **Make a screen capture** showing the **hash information** and **paste** it into the Lab Report file.

13. In the web form, **submit** `'UNION SELECT 'test', '123' INTO OUTFILE 'test2.txt`.

Note: This script, if successful, indicates that data can be written to a file, and later downloaded. How do you know that it is successful? If the file can be created, there will be no error message and the script will pass through the Submit button and clear the User ID box. If the file cannot be created, either because the file already exists or because the script was faulty or the developer prevented it, you will receive an error or the script will remain in the User ID box. Together with the information you gathered in earlier tests, you now have a user with elevated permissions, user IDs, passwords, and the table structure where all of this data is being held—in other words, an injectable database.

14. **Close** the **browser** to exit DVWA.

Note: This completes Section 2 of this lab. There are no deliverable files for this section.

Section 3: Lab Challenge and Analysis

Note: The following questions are provided to allow you the opportunity for independent, unguided research, similar to what you will encounter in a real situation. Some questions will challenge you to find command line syntax for tasks you performed in the lab, others may ask you to extend your learning from the lab. Use screen captures where possible to illustrate your answers.

Part 1: Analysis and Discussion

Research best practices for preventing SQL injection attacks on the network. Identify at least three common practices and cite your sources.

Part 2: Tools and Commands

Create a SQL injection attack that will determine the correct field name that holds the user's surname.

Part 3: Challenge Exercise

In the lab, you discovered that the password hash was included in the mysql.user database; however, there is another database that contains user data. What is the name of that database, and what content can you uncover within it?