TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**Ảnh có chứa văn bản, ký hiệu

Mô tả được tạo tự động**

**Tiểu luận môn nhập môn an toàn thông tin**

**Đề tài: Tấn công SQL Injection**

| Lớp                                 : | 157091 |
| --- | --- |
| Mã học phần                   : | IT4015 |
| Nhóm : | 13 |
| Giảng viên hướng dẫn    : | PGS. TS. Nguyễn Linh Giang |

Danh sách thành viên nhóm:

| Họ và tên | Mã số sinh viên |
| --- | --- |
| Lê Viết Đức Anh | 20226075 |
| Hồ Tuấn Anh | 20226100 |
| Nguyễn Tài Hoan | 20226106 |
| Đỗ Đình Trung | 20226128 |

*Hà Nội, tháng 6 năm 2025*

# **PHÂN CÔNG THÀNH VIÊN TRONG NHÓM**

| STT | Họ và tên | MSSV | Email | Công việc | Tỷ lệ % công việc làm |
| --- | --- | --- | --- | --- | --- |
| 1 | Hồ Tuấn Anh | 20226100 | Anh.HT226100@sis.hust.edu.vn | * Giới thiệu, tổng quan về tấn công SQL Injection * Cơ chế hoạt động của SQLi | 25% |
| 2 | Đỗ Đình Trung | 20226128 | Trung.DD226128@sis.hust.edu.vn | * Phát hiện SQL Injection * Công cụ phát hiện SQLi (Burp Suite, SQLMap,...) | 25% |
| 3 | Lê Viết Đức Anh | 20226075 | Anh.LVD226075@sis.hust.edu.vn | * Tìm kiếm và khai thác lỗi SQL Injection | 25% |
| 4 | Nguyễn Tài Hoan | 20226106 | Hoan.NT226106@sis.hust.edu.vn | * Phòng chống và khắc phục SQLi * Phân công nhiệm vụ | 25% |

# 

**MỤC LỤC**

[**PHÂN CÔNG THÀNH VIÊN TRONG NHÓM 1**](#_598q6h2im7bw)

[**I. Giới thiệu 4**](#_cgdtnqvxyarh)

[2. Mục tiêu và phạm vi đề tài 5](#_xq8sg7s7cyio)

[**II. Tổng quan về SQL Injection 6**](#_uudghfmnq0f2)

[1. Định nghĩa 7](#_frlk4iglw39p)

[1.1. Tấn công truy cập trái phép 7](#_76gksptwsvdp)

[1.2. Tấn công leo thang đặc quyền 8](#_a3di4jsttv1o)

[2. Cơ chế tấn công SQL Injection 8](#_tfytqqma6mxw)

[2.1. Giai đoạn trinh sát (Reconnaissance) 8](#_mk1mxabr6c3)

[2.2 Giai đoạn đánh giá lỗ hổng (Vulnerability Assessment) 8](#_l2g9vru5n18)

[2.3. Giai đoạn khai thác (Exploitation) 9](#_jpc55dw7m1ek)

[**III. Phát hiện SQL Injection 9**](#_emljytih47va)

[1. Phương pháp thủ công (manual testing) 9](#_r48q5ih7cbf7)

[2. Công cụ phát hiện SQLi (Burp Suite, SQLMap,...) 10](#_khdaaanfi77i)

[3. Dấu hiệu nhận biết trên logs và hành vi ứng dụng 10](#_xduunsswokxz)

[**IV. Tìm kiếm và khai thác lỗi SQL Injection 10**](#_obikvdy91p2g)

[1. Cách phát hiện input không kiểm soát 10](#_cca57m2t1nmg)

[2. Kỹ thuật fuzzing và khai thác query lỗi 10](#_61ddft7pkgtr)

[3. Demo thực tế: Tìm lỗi SQLi trong DVWA hoặc WebGoat 10](#_u1l33eq8oqr9)

[**V. Phòng chống và khắc phục 10**](#_s5ttd6t5pzjc)

[1. Tham số hóa truy vấn (prepared statements) 11](#_bqgstoz8as4)

[2. Xác thực đầu vào (Input validation) và làm sạch dữ liệu (Data Sanitization) 13](#_timdzg8ne00)

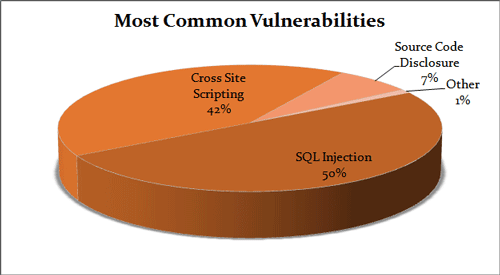
[3. Áp dụng Nguyên tắc đặc quyền tối thiểu 15](#_eiyahfecoc8k)

[4. Các biện pháp phòng chống bổ sung 15](#_yb9iranijw2z)

[**VI. Kết luận 16**](#_aouvkip16tcp)

# I. Giới thiệu

Các hệ thống dựa trên web thường đối mặt với nhiều mối đe dọa ảnh hưởng đến chức năng hoạt động của chúng. Do các hệ thống này cung cấp nhiều dịch vụ được lưu trữ trên cơ sở dữ liệu, chúng dễ trở thành mục tiêu của các cuộc tấn công SQL Injection nhằm đánh cắp hoặc xáo trộn dữ liệu, đặc biệt là các dữ liệu nhạy cảm, và trong trường hợp xấu nhất, kẻ tấn công có thể chiếm được quyền truy cập quản trị vào máy chủ cơ sở dữ liệu. Trong bối cảnh ngày càng gia tăng các cuộc tấn công mạng và sự phụ thuộc lớn vào các ứng dụng web, việc nghiên cứu SQL Injection không chỉ giúp hiểu rõ bản chất của lỗ hổng mà còn đóng vai trò quan trọng trong việc phát triển các biện pháp phòng chống hiệu quả.



***Biểu đồ tỷ lệ xảy ra các cuộc tấn công mạng (nguồn:*** [***https://developer.mozilla.org***](https://developer.mozilla.org/)***)***

Trong suốt khoảng thời gian kể từ khi các máy chủ web đầu tiên được triển khai đến thời điểm hiện tại, lịch sử ngành an ninh mạng đã chứng kiến không ít các cuộc tấn công SQL Injection nhắm vào vô số doanh nghiệp, tập đoàn lớn nhỏ, gây ra nhiều thiệt hại đáng kể cho nhiều doanh nghiệp và tổ chức trong những năm qua:

Vào tháng 4 năm 2011, Sony đã trải qua một trong những vụ tấn công dữ liệu nổi tiếng nhất trong lịch sử khi tin tặc xâm nhập vào PlayStation Network (PSN). Những kẻ tấn công đã sử dụng kết hợp các kỹ thuật lừa đảo và SQL Injection để truy cập trái phép vào máy chủ của Sony. Cuộc tấn công mạng này đã tiết lộ thông tin cá nhân của khoảng 77 triệu người dùng, bao gồm tên, địa chỉ và có thể là thông tin chi tiết về thẻ tín dụng, buộc Sony phải đóng cửa PSN trong gần một tháng, khiến hàng triệu game thủ không thể truy cập vào các dịch vụ của mạng. Thiệt hại này khiến Sony phải chịu tổn thất tài chính đáng kể do vụ vi phạm. Chi phí điều tra sự cố, cải thiện các biện pháp bảo mật và bồi thường cho người dùng bị ảnh hưởng ước tính lên tới 171 triệu đô la. Không chỉ vậy, vụ việc còn gây tổn hại nghiêm trọng đến danh tiếng của Sony. Khách hàng mất lòng tin vào khả năng bảo vệ dữ liệu của công ty, dẫn đến sự suy giảm lòng tin và lòng trung thành của người dùng.

Từ tháng 11 đến tháng 12 năm 2023, nhóm tin tặc ResumeLooters đã xâm phạm hơn 65 trang web, chủ yếu trong lĩnh vực tuyển dụng và bán lẻ, bằng cách sử dụng các cuộc tấn công SQL Injection và Cross-Site Scripting (XSS). Những kẻ tấn công đã thu thập hơn 2 triệu hồ sơ người dùng, bao gồm tên, email và số điện thoại. Dữ liệu bị đánh cắp sau đó được bán trên nhiều nền tảng tội phạm mạng khác nhau.

Ngày 29/07/2024, lỗ hổng SQL Injection trong sản phẩm OpManager được công bố với mã định danh CVE-2024-6748, được đánh giá mức cao (8.3/10), cho phép tiêm mã SQL độc hại trong quá trình giám sát URL.

## 2. Mục tiêu và phạm vi đề tài

Nghiên cứu về SQL Injection nhằm đạt được các mục tiêu dưới đây:

* Hiểu rõ bản chất và cơ chế của SQL Injection: Phân tích cách thức hoạt động của các cuộc tấn công SQL Injection, bao gồm các kỹ thuật tấn công phổ biến như Union-based, Blind SQL Injection, Time-based, và Error-based. Mục tiêu là làm rõ cách tin tặc khai thác lỗ hổng trong ứng dụng web để thao túng cơ sở dữ liệu.
* Đánh giá tác động của SQL Injection: Xác định mức độ ảnh hưởng của các cuộc tấn công này đến tính bảo mật, tính toàn vẹn và tính sẵn sàng của dữ liệu trong các hệ thống web, từ đó nhấn mạnh tầm quan trọng của việc phòng chống.
* Đề xuất giải pháp phòng chống hiệu quả: Phát triển hoặc cải tiến các phương pháp, kỹ thuật và công cụ nhằm phát hiện, ngăn chặn và giảm thiểu rủi ro từ SQL Injection. Các giải pháp này có thể bao gồm kỹ thuật lập trình an toàn (như sử dụng Prepared Statements, Parameterized Queries), cấu hình hệ thống bảo mật, và ứng dụng các công nghệ tiên tiến như máy học hoặc học sâu.
* Nâng cao nhận thức và đào tạo: Cung cấp thông tin và kiến thức để nâng cao nhận thức về bảo mật cho các nhà phát triển, quản trị viên, và tổ chức, từ đó giảm thiểu nguy cơ bị tấn công do thiếu hiểu biết hoặc sai sót trong thiết kế hệ thống.

Đề tài nghiên cứu SQL Injection sẽ tập trung vào các khía cạnh quan trọng sau:

* Chương 2 trình bày về cách thức thực hiện một cuộc tấn công SQL và ảnh hưởng của nó đến các hệ quản trị cơ sở dữ liệu nói riêng và dịch vụ web nói chung.
* Chương 3 trình bày về một số phương pháp phát hiện SQL Injection.
* Chương 4 trình bày về cách thức tìm kiếm và khai thác lỗi SQL Injection.
* Chương 5 trình bày về một số biện pháp phòng chống và khắc phục.

# II. Tổng quan về SQL Injection

## Định nghĩa

SQL Injection là quá trình kẻ tấn công thực thi các câu lệnh SQL độc hại trong máy chủ cơ sở dữ liệu thông qua việc “tiêm” các câu lệnh này vào các yêu cầu từ máy khách đến máy chủ web (như các yêu cầu HTTP Post, Put,..). Ví dụ sau đây mô tả một số hình thức lợi dụng tấn công SQL Injection nhằm kiểm soát hệ thống:

### 1.1. Tấn công truy cập trái phép

Ở hình thức tấn công này, tin tặc lợi dụng điểm yếu trong lời gọi thực thi câu lệnh SQL từ máy chủ web (back-end) mà không có quy trình “khử trùng” tham số đầu vào. Cụ thể, với câu lệnh:

```js

function checkUserAuth(username, password) {

let query = 'SELECT \* FROM Users WHERE username = ' + username + ‘AND password =’ + password

db. query(query)

// …

}

```

Nếu tham số password được truyền vào là “anything or 1= 1 “ (do kẻ tấn công đã tiêm chuỗi “or 1= 1” vào tham số được gửi từ máy khách), cơ sở dữ liệu sẽ luôn trả về bản ghi khớp với giá trị của username, bất kể tính hợp lệ của mật khẩu. Nhờ vậy kẻ tấn công dễ dàng vượt qua bộ lọc xác thực và xâm phạm vào hệ thống

### 1.2. Tấn công leo thang đặc quyền

Kẻ tấn công có thể tiêm mã độc tạo tài khoản quản trị viên trong hệ quản trị cơ sở dữ liệu (ví dụ như chuỗi truy vấn:

```pgsql

SELECT \* FROM …;CREATE ROLE attacker WITH SUPERUSER LOGIN PASSWORD 'hackme'; --');

```

Nếu thành công, kẻ tấn công có thể truy cập và điều khiển hệ quản trị cơ sở dữ liệu từ xa, thậm chí có thể thực thi các lệnh tương tác với hệ điều hành.

## Cơ chế tấn công SQL Injection

### 2.1. Giai đoạn trinh sát (Reconnaissance)

Xác định điểm nhập (Entry Points):

* Kẻ tấn công quét tất cả input fields: form đăng nhập, search box, URL parameters, cookies, HTTP headers
* Sử dụng các công cụ như SQLMap, Burp Suite để tự động hóa việc tìm kiếm
* Phân tích source code nếu có thể (white-box testing)

Fingerprinting Database:

* Gửi các payload đặc trưng để xác định loại database (MySQL, PostgreSQL, MSSQL, Oracle)
* Dựa vào error messages, response time, behavior khác nhau (Ví dụ: MySQL trả về error khác với PostgreSQL khi sai cú pháp)

### 2.2 Giai đoạn đánh giá lỗ hổng (Vulnerability Assessment)

Thử các ký tự đặc biệt: ', ", ;, --, /\*, \*/

Theo dõi phản ứng của ứng dụng: error messages, blank pages, timeout

Xác định xem input có được filter/escape hay không

Xác định loại SQL Injection:

* Error-based: Lợi dụng error messages để extract thông tin
* Blind SQLi: Không có error message, dựa vào boolean response hoặc time delay
* Union-based: Sử dụng UNION để extract data từ các table khác

### 2.3. Giai đoạn khai thác (Exploitation)

Schema Discovery:

* Extract thông tin về database structure: table names, column names, data types
* Sử dụng system tables như information\_schema (MySQL), sys.tables (MSSQL)
* Áp dụng các technique như binary search cho blind SQLi

Data Extraction:

* Systematically extract sensitive data: passwords, user info, business data
* Sử dụng các function như CONCAT(), GROUP\_CONCAT() để tối ưu hóa việc truy xuất dữ liệu
* Bypass các filter bằng encoding, comment injection, case variation

# III. Phát hiện SQL Injection

## Phương pháp thủ công (manual testing)

Phương pháp thủ công là cách kiểm tra SQLi bằng cách trực tiếp nhập các dữ liệu thử nghiệm vào ứng dụng mà không phụ thuộc hoàn toàn vào công cụ tự động. Phương pháp này yêu cầu người kiểm tra (tester) có kiến thức về SQL và cách ứng dụng tương tác với cơ sở dữ liệu. Dưới đây là các bước chính:

* **Xác định điểm nhập liệu (Entry Points):** Các điểm nhập liệu bao gồm các trường trong biểu mẫu (như đăng nhập, tìm kiếm), tham số URL (query string, ví dụ: http://example.com/index.php?id=1), hoặc cookie. Đây là những nơi kẻ tấn công có thể chèn mã SQL độc hại.
* **Thử nghiệm với ký tự đặc biệt:** Sử dụng các ký tự như dấu nháy đơn ('), dấu nháy kép ("), dấu gạch ngang kép (--), hoặc dấu thăng (#) để kiểm tra phản hồi của ứng dụng. Ví dụ:
  + Nhập ' OR 1=1-- vào ô tìm kiếm. Nếu ứng dụng trả về toàn bộ dữ liệu hoặc thông báo lỗi SQL (như "You have an error in your SQL syntax"), điều này cho thấy ứng dụng dễ bị tấn công SQLi.
  + Thử ' AND 1=2-- để kiểm tra phản hồi khác biệt so với ' AND 1=1--. Sự khác biệt này xác nhận ứng dụng đang thực thi câu lệnh SQL từ đầu vào.
* **Kiểm tra lỗi cú pháp SQL:** Nhập các ký tự không hợp lệ như ' hoặc " vào các trường nhập liệu. Nếu ứng dụng trả về thông báo lỗi liên quan đến cú pháp SQL (ví dụ: "Invalid query" hoặc "SQL syntax error"), điều này cho thấy đầu vào không được xử lý an toàn.
* **Kỹ thuật Union-Based:** Sử dụng câu lệnh UNION SELECT để truy xuất dữ liệu từ các bảng khác. Ví dụ: http://example.com/index.php?id=1 UNION SELECT 1,2,3--. Nếu trang web trả về dữ liệu bổ sung (như số 1, 2, 3) hoặc thông tin từ cơ sở dữ liệu, điều này xác nhận lỗ hổng SQLi.
* **Kỹ thuật Blind SQL Injection:** Khi ứng dụng không hiển thị lỗi hoặc dữ liệu trực tiếp, tester sử dụng các kỹ thuật như:
  + **Boolean-based:** So sánh phản hồi của ứng dụng khi nhập ' AND 1=1-- và ' AND 1=2--. Nếu phản hồi khác nhau, có khả năng tồn tại lỗ hổng.
  + **Time-based:** Sử dụng các hàm như SLEEP(10) (MySQL) hoặc pg\_sleep(10) (PostgreSQL). Ví dụ: http://example.com/index.php?id=1; SELECT CASE WHEN (1=1) THEN SLEEP(10) ELSE SLEEP(0) END--. Nếu trang web mất 10 giây để phản hồi, điều này xác nhận SQLi.
* **Ví dụ thực tế:** Giả sử bạn kiểm tra một form đăng nhập. Nhập admin' OR '1'='1 vào trường tên người dùng và mật khẩu. Nếu ứng dụng cho phép đăng nhập mà không cần mật khẩu đúng, điều này cho thấy lỗ hổng SQLi. Theo [OWASP](https://owasp.org/www-community/attacks/SQL_Injection), kỹ thuật này thường được sử dụng để xác định các lỗ hổng cơ bản.
* **Ưu điểm và hạn chế:** Phương pháp thủ công cho phép tester hiểu rõ cách ứng dụng xử lý dữ liệu, phát hiện các lỗ hổng phức tạp mà công cụ tự động có thể bỏ sót. Tuy nhiên, nó tốn thời gian và đòi hỏi kỹ năng cao.

## Công cụ phát hiện SQLi (Burp Suite, SQLMap,...)

## **Burp Suite**

## Burp Suite là nền tảng kiểm thử bảo mật web hàng đầu, được sử dụng rộng rãi để phát hiện SQLi.

## **Tính năng chính:**

## **Burp Scanner (Professional Edition):** Tự động quét các điểm nhập liệu (form, URL, cookie) với các payload như ' OR 1=1-- để phát hiện SQLi.

* + **Burp Repeater:** Cho phép chỉnh sửa và gửi lại yêu cầu HTTP, kiểm tra thủ công các payload như ' AND 1=1--.
  + **Burp Intruder:** Tự động gửi hàng loạt payload, hiệu quả cho Blind SQLi với chế độ "Cluster bomb".
  + **Tích hợp với SQLMap:** Sử dụng extension SQLiPy để kết hợp khai thác tự động.

1. **Cách sử dụng:** Cấu hình proxy để chặn lưu lượng HTTP, chạy Scanner hoặc sử dụng Repeater để kiểm tra payload như id=1' OR '1'='1. Ví dụ: Gửi yêu cầu qua Intruder với danh sách payload để kiểm tra Blind SQLi.
2. **Ưu điểm:** Linh hoạt, hỗ trợ cả kiểm tra thủ công và tự động, phù hợp cho tester chuyên nghiệp.
3. **Hạn chế:** Bản Community Edition hạn chế tính năng; bản Professional có chi phí cao (khoảng $399/năm, theo PortSwigger).
4. **Nguồn tham khảo:** PortSwigger Web Security Academy.

## **2. SQLMap**

SQLMap là công cụ mã nguồn mở mạnh mẽ, tự động hóa việc phát hiện và khai thác SQLi, hỗ trợ nhiều cơ sở dữ liệu (MySQL, PostgreSQL, Oracle, SQL Server).

* **Tính năng chính:**
  + **Phát hiện lỗ hổng:** Kiểm tra SQLi với lệnh như sqlmap -u "http://example.com/index.php?id=1" --batch.
  + **Khai thác dữ liệu:** Liệt kê cơ sở dữ liệu (--dbs), bảng (--tables), cột (--columns), hoặc trích xuất dữ liệu (--dump). Ví dụ: sqlmap -u "http://example.com/index.php?id=1" --dbs.
  + **Hỗ trợ Blind SQLi:** Sử dụng kỹ thuật Boolean-based và time-based (như SLEEP(10)).
  + **Tùy chỉnh payload:** Sử dụng tamper scripts để vượt qua WAF.
* **Cách sử dụng:** Chạy lệnh cơ bản như sqlmap -u <URL> --level=3 --risk=3 để tăng độ sâu kiểm tra. Thêm --tamper để vượt qua bảo vệ WAF.
* **Ưu điểm:** Miễn phí, mạnh mẽ, hỗ trợ nhiều loại tấn công và cơ sở dữ liệu.
* **Hạn chế:** Tạo lưu lượng lớn, dễ bị WAF chặn nếu không cấu hình cẩn thận.
* **Nguồn tham khảo:** SQLMap Official và GitHub SQLMap.

## **3. OWASP ZAP**

OWASP ZAP (Zed Attack Proxy) là công cụ mã nguồn mở, thân thiện với người mới, được sử dụng để quét SQLi.

* **Tính năng chính:**
  + **Active Scanning:** Tự động gửi payload để kiểm tra các điểm nhập liệu.
  + **Passive Scanning:** Phân tích lưu lượng HTTP để phát hiện dấu hiệu SQLi mà không gửi yêu cầu bổ sung.
  + **Spidering:** Thu thập thông tin cấu trúc ứng dụng để xác định điểm nhập liệu.
* **Cách sử dụng:** Cấu hình proxy, chạy Active Scan trên URL (ví dụ: http://example.com), và xem báo cáo lỗ hổng.
* **Ưu điểm:** Miễn phí, dễ sử dụng, phù hợp cho dự án nhỏ hoặc người mới học bảo mật.
* **Hạn chế:** Ít tính năng nâng cao so với Burp Suite hoặc SQLMap.
* **Nguồn tham khảo:** OWASP ZAP.

## **4. Acunetix**

Acunetix là công cụ thương mại, nổi bật với khả năng quét nhanh và phát hiện SQLi hiệu quả.

* **Tính năng chính:**
  + **Deep Scanning:** Kiểm tra các điểm nhập liệu phức tạp (form, cookie, AJAX).
  + **Hỗ trợ Blind SQLi:** Phát hiện lỗ hổng không hiển thị lỗi trực tiếp.
  + **Tích hợp WAF:** Kết nối với WAF để bảo vệ ngay sau khi phát hiện.
* **Cách sử dụng:** Nhập URL, chạy quét toàn diện, và phân tích báo cáo chi tiết.
* **Ưu điểm:** Giao diện trực quan, tốc độ quét nhanh, giảm dương tính giả.
* **Hạn chế:** Chi phí cao, yêu cầu tài nguyên hệ thống lớn.
* **Nguồn tham khảo:** Acunetix.

## **5. Netsparker**

Netsparker là công cụ thương mại, tập trung vào độ chính xác cao trong phát hiện SQLi.

* **Tính năng chính:**
  + **Proof-Based Scanning:** Xác minh lỗ hổng bằng cách khai thác thử, giảm dương tính giả.
  + **Hỗ trợ CI/CD:** Tích hợp vào quy trình phát triển phần mềm.
  + **Báo cáo chi tiết:** Cung cấp hướng dẫn khắc phục cụ thể.
* **Cách sử dụng:** Nhập URL, chạy quét, và xem báo cáo để xác định SQLi.
* **Ưu điểm:** Độ chính xác cao, phù hợp cho doanh nghiệp.
* **Hạn chế:** Chi phí cao, không lý tưởng cho cá nhân hoặc dự án nhỏ.
* **Nguồn tham khảo:** Netsparker.

## **6. Invicti**

Invicti (trước đây là Netsparker) là công cụ thương mại, chuyên về quét bảo mật web tự động.

* **Tính năng chính:**
  + **Dynamic Application Security Testing (DAST):** Quét toàn diện để phát hiện SQLi.
  + **Hỗ trợ xác minh:** Kiểm tra khai thác để đảm bảo kết quả chính xác.
  + **Tích hợp DevOps:** Phù hợp cho quy trình phát triển hiện đại.
* **Cách sử dụng:** Cấu hình URL, chạy quét, và phân tích báo cáo.
* **Ưu điểm:** Chính xác, phù hợp cho doanh nghiệp lớn.
* **Hạn chế:** Chi phí cao, không phù hợp cho cá nhân.
* **Nguồn tham khảo:** Invicti.

## **7. Lưu ý khi sử dụng công cụ**

* **Kết hợp công cụ:** Sử dụng Burp Suite cho kiểm tra thủ công và SQLMap cho khai thác tự động để tăng hiệu quả.
* **Tùy chỉnh payload:** Đối với ứng dụng có WAF, sử dụng tamper scripts trong SQLMap hoặc cấu hình Burp Intruder để vượt qua bảo vệ.
* **Giảm dương tính giả:** Các công cụ như Netsparker và Acunetix có tính năng xác minh, giúp giảm báo cáo sai.
* **Tài liệu tham khảo:** Theo Zenarmor, kết hợp mã nguồn mở (SQLMap, OWASP ZAP) với thương mại (Netsparker, Acunetix) mang lại kết quả tối ưu.

## **8. Kết luận**

Các công cụ như Burp Suite, SQLMap, OWASP ZAP, Acunetix, Netsparker và Invicti là những lựa chọn phổ biến và hiệu quả để phát hiện SQLi. Burp Suite và SQLMap nổi bật với tính linh hoạt và khả năng khai thác, trong khi Acunetix và Netsparker phù hợp cho doanh nghiệp nhờ độ chính xác cao. OWASP ZAP là lựa chọn tốt cho người mới hoặc dự án nhỏ. Kết hợp các công cụ này với kiến thức bảo mật sẽ giúp phát hiện SQLi toàn diện, bảo vệ ứng dụng web khỏi các cuộc tấn công nguy hiểm.

## Dấu hiệu nhận biết trên logs và hành vi ứng dụng

Phân tích logs và quan sát hành vi ứng dụng là cách phát hiện SQLi hiệu quả, đặc biệt trong các hệ thống giám sát bảo mật. Dưới đây là các dấu hiệu chính:

* **Dấu hiệu trong logs:**
  + **Lỗi cú pháp SQL:** Logs của máy chủ hoặc ứng dụng có thể ghi lại các thông báo lỗi như "You have an error in your SQL syntax" khi kẻ tấn công thử nhập ' hoặc ".
  + **Yêu cầu bất thường:** Logs HTTP ghi nhận các chuỗi SQL độc hại như ' OR 1=1--, UNION SELECT, hoặc SLEEP(10) trong tham số URL, trường POST, hoặc cookie.
  + **Thời gian phản hồi dài:** Trong Blind SQLi (time-based), logs có thể ghi nhận các yêu cầu mất thời gian bất thường (ví dụ: 10 giây) do sử dụng hàm SLEEP() hoặc pg\_sleep().
  + **Lưu lượng bất thường từ một IP:** Logs của WAF hoặc máy chủ web cho thấy nhiều yêu cầu liên tiếp từ cùng một IP với các payload SQLi, như information\_schema hoặc xp\_cmdshell.
* **Hành vi bất thường của ứng dụng:**
  + **Thông báo lỗi lộ liễu:** Ứng dụng trả về lỗi liên quan đến cơ sở dữ liệu (như "Internal Server Error") khi nhập các ký tự đặc biệt, cho thấy thiếu kiểm tra đầu vào.
  + **Trả về dữ liệu không mong muốn:** Hiển thị dữ liệu từ các bảng khác (như danh sách người dùng) khi thử payload như UNION SELECT.
  + **Phản hồi không nhất quán:** Khi nhập ' AND 1=1-- và ' AND 1=2--, ứng dụng trả về các phản hồi khác nhau, cho thấy nó đang xử lý logic SQL.
  + **Trang trống hoặc lỗi:** Ứng dụng không phản hồi hoặc hiển thị trang trống khi gặp mã SQL độc hại, như 1; DROP TABLE users--.
  + **Thay đổi dữ liệu bất thường:** Dữ liệu bị sửa đổi hoặc thêm mới (như tài khoản admin) là dấu hiệu của SQLi thành công.
* **Phân tích hành vi:** Sử dụng các công cụ SIEM (Security Information and Event Management) hoặc WAF để phát hiện các mẫu hành vi bất thường, như số lượng lớn yêu cầu đến một endpoint hoặc tăng đột biến lỗi cơ sở dữ liệu. Theo [Redgate](https://www.red-gate.com/hub/product-learning/redgate-monitor/detect-sql-injection-attacks-using-extended-events-sql-monitor), giám sát logs bằng các công cụ như SQL Monitor giúp phát hiện sớm các cuộc tấn công SQLi.

# IV. Tìm kiếm và khai thác lỗi SQL Injection

## Cách phát hiện input không kiểm soát

## Kỹ thuật fuzzing và khai thác query lỗi

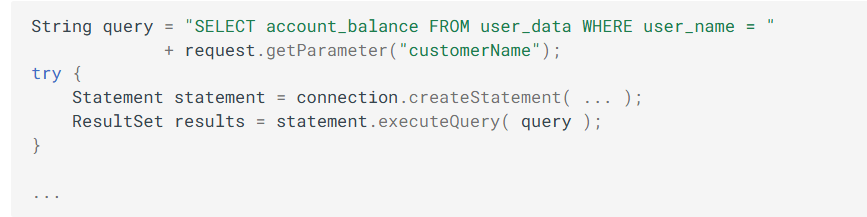
## Demo thực tế: Tìm lỗi SQLi trong DVWA hoặc WebGoat

# V. Phòng chống và khắc phục

Tấn công SQL Injection (SQLi) là một trong những mối đe dọa nguy hiểm nhất đối với bảo mật ứng dụng web, có thể dẫn đến rò rỉ dữ liệu nhạy cảm, thay đổi hoặc xóa bỏ thông tin, thậm chí kiểm soát toàn bộ máy chủ cơ sở dữ liệu. Mặc dù các kỹ thuật tấn công ngày càng phức tạp, nhưng các biện pháp phòng chống đã được thiết lập rõ ràng và tương đối đơn giản để triển khai. Việc phòng chống SQLi đòi hỏi một cách tiếp cận đa lớp và sự cảnh giác liên tục trong quá trình phát triển và vận hành ứng dụng

## Tham số hóa truy vấn (prepared statements)

Đây là một trong những cách hiệu quả và được khuyến nghị nhất để bảo vệ khỏi SQL Injection. Tham số hóa truy vấn (prepared statements) hoạt động bằng cách tách hoàn toàn mã SQL khỏi dữ liệu đầu vào của người dùng. Thay vì nối chuỗi trực tiếp, nhà phát triển định nghĩa cấu trúc truy vấn SQL trước với các phần giữ chỗ (placeholders) cho dữ liệu, và sau đó truyền dữ liệu này vào các tham số riêng biệt.



***Một câu truy vấn chưa được tham số hoá truy vấn, giá trị sẽ được chèn vào trực tiếp câu truy vấn. Nguồn SQL Injection Prevention***

Câu truy vấn trên là câu truy vấn thường, chưa được tham số hoá truy vấn. Nếu kẻ tấn công nhập vào customerName như sau: *' OR '1' = '1*. Truy vấn sẽ trở thành:

*SELECT account\_balance FROM user\_data WHERE user\_name = '' OR '1' = '1'*

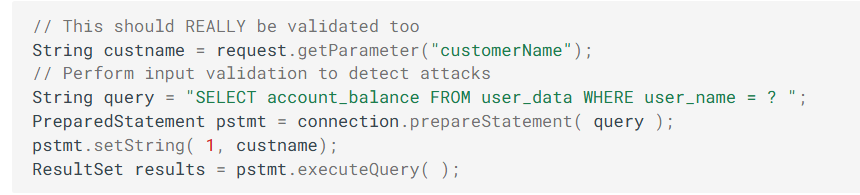
Điều này sẽ khiến truy vấn trả về tất cả các bản ghi trong cơ sở dữ liệu, thay vì chỉ bản ghi của người dùng cụ thể.

Khi sử dụng tham số hoá truy vấn, cơ sở dữ liệu sẽ luôn phân biệt rõ ràng giữa mã và dữ liệu. Dù kẻ tấn công có chèn mã SQL độc hại vào đầu vào người dùng, mã này sẽ được xử lý như một giá trị dữ liệu thông thường cho tham số đó, chứ không phải là một phần của lệnh SQL cần được thực thi. Điều này đảm bảo ý định của truy vấn không bị thay đổi.

Lợi ích của tham số hoá truy vấn

* *Tăng cường bảo mật:* Đây là lợi ích quan trọng nhất, giúp ngăn chặn SQL Injection một cách hiệu quả.
* *Giảm thời gian phân tích cú pháp*: Truy vấn chỉ được chuẩn bị (parse và compile) một lần duy nhất.
* *Giảm băng thông*: Chỉ cần gửi các tham số mỗi lần thực thi, không cần gửi lại toàn bộ truy vấn.

Hầu hết các ngôn ngữ lập trình và nền tảng phát triển đều hỗ trợ Prepared Statements, bao gồm Java (sử dụng *PreparedStatement*), C#/.NET (sử dụng *Parameters.Add()*), PHP (sử dụng PDO - PHP Data Object), Ruby, Perl, và Rust. PDO là một lớp trừu tượng cơ sở dữ liệu giúp đơn giản hóa và bảo mật việc tương tác với nhiều loại cơ sở dữ liệu khác nhau.



***Truy vấn sau khi được tham số hoá truy vấn. Nguồn SQL Injection Prevention***

Sau khi được tham số hoá truy vấn, vẫn với cách tấn công cũ *' OR '1' = '1,* bây giờ câu truy vấn sẽ trở thành

*SELECT account\_balance FROM user\_data WHERE user\_name = ''' OR ''1'' = ''1'''*

Đây là một chuỗi hợp lệ, nhưng không còn là đoạn SQL có điều kiện. Cơ sở dữ liệu sẽ tìm một dòng mà user\_name đúng bằng chuỗi ' OR '1' = '1, chứ không hiểu nó như điều kiện OR nữa.

## Xác thực đầu vào (Input validation) và làm sạch dữ liệu (Data Sanitization)

Năm 2008, Hacker Albert Gonzalez đã khai thác lỗ hổng SQLi để xâm nhập hệ thống của Heartland, đánh cắp 130 triệu thẻ tín dụng. Nguyên nhân là do thiếu xác thực đầu vào; dữ liệu được đưa trực tiếp vào truy vấn SQL mà không được kiểm soát. Việc không xác thực đầu vào người dùng trước khi sử dụng là một lỗ hổng bảo mật phổ biến, dẫn đến các cuộc tấn công như SQL Injection. Mọi đầu vào của người dùng phải được xác thực và làm sạch đúng cách trước khi tương tác với cơ sở dữ liệu. Cho câu truy vấn trước khi được xác thực đầu vào và làm sạch dữ liệu:  
*$username = $\_GET['username'];*

*$query = "SELECT \* FROM users WHERE username = '$username'";*

Nếu người dùng nhập *admin' --* ,câu truy vấn trở thành *SELECT \* FROM users WHERE username = 'admin' --'*

Câu truy vấn này đã bỏ qua phần còn lại và đăng nhập trái phép.

Xác thực đầu vào (Input validation): Đảm bảo rằng chỉ các đầu vào hợp lệ và mong muốn mới được chấp nhận.

* Allow-list (Danh sách cho phép): Đây là phương pháp được khuyến nghị. Nó bao gồm việc định nghĩa một danh sách các ký tự, định dạng, phạm vi hoặc giá trị được phép. Mọi đầu vào không khớp với danh sách này sẽ bị từ chối. Ví dụ: nếu một trường chỉ chấp nhận số điện thoại, nó chỉ nên cho phép các ký tự số.
* Deny-list (Danh sách từ chối): Phương pháp này không được khuyến nghị vì dễ bị vượt qua. Nó cố gắng chặn các mẫu độc hại đã biết nhưng luôn có khả năng bị bỏ qua bởi kẻ tấn công tinh vi.

Lưu ý: Xác thực đầu vào nên được thực hiện ở cả phía client (client-side) và server (server-side). Việc xác thực ở client-side không đủ vì kẻ tấn công có thể dễ dàng bỏ qua nó.

Tương tự với câu truy vấn trên ta có thể sử dụng allow-list

*if (!preg\_match('/^[a-zA-Z0-9\_]{3,20}$/', $username)) {*

*die("Invalid username");*

*}*

Chỉ cho phép các ký tự thuộc danh sách an toàn: chữ cái (a-z, A-Z), số (0-9) và dấu gạch dưới (\_). Bất kỳ ký tự nào không nằm trong danh sách này sẽ bị từ chối ngay lập tức. Hoặc có thể sử dụng deny-list  
*if (preg\_match('/[\'";]/', $username)) {*

*die("Invalid character in username");*

*}*

Loại bỏ một số ký tự nguy hiểm như ', ", ;,...

Làm sạch dữ liệu (Data Sanitization) là quá trình này điều chỉnh đầu vào của người dùng để đảm bảo nó hợp lệ và an toàn. Nó liên quan đến việc lọc bỏ bất kỳ ký tự nào không mong muốn hoặc có khả năng gây hại khỏi dữ liệu. Sử dụng biểu thức chính quy (regular expressions) là công cụ mạnh mẽ để phân tích chuỗi và đối sánh mẫu, giúp loại bỏ các ký tự "bất hợp pháp".

Việc làm sạch dữ liệu cần được áp dụng cho tất cả các loại trường đầu vào, bao gồm cả trường ẩn, trường bị vô hiệu hóa và cookie.

Ví dụ: một trường tên người dùng chỉ nên chứa các ký tự chữ và số, không nên chứa dấu nháy đơn hoặc các ký tự đặc biệt khác có thể bị lợi dụng

## Áp dụng Nguyên tắc đặc quyền tối thiểu

Nguyên tắc này yêu cầu mỗi tài khoản cơ sở dữ liệu chỉ được cấp quyền tối thiểu cần thiết để thực hiện công việc của nó. Điều này nhằm giảm thiểu thiệt hại tiềm tàng nếu một cuộc tấn công SQL Injection thành công.

Biện pháp cụ thể như sau:

* Phân quyền cho tài khoản ứng dụng: Tài khoản ứng dụng kết nối với cơ sở dữ liệu chỉ nên có quyền truy cập (đọc, ghi, cập nhật, xóa) vào các bảng và cột mà nó thực sự cần. Tuyệt đối không cấp quyền quản trị (DBA hoặc ADMIN) cho tài khoản ứng dụng.
* Phân tách người dùng DB: Mỗi ứng dụng web nên sử dụng một tài khoản người dùng cơ sở dữ liệu riêng biệt. Ví dụ, trang đăng nhập chỉ cần quyền đọc thông tin người dùng, trong khi trang đăng ký cần quyền chèn dữ liệu. Việc sử dụng các tài khoản khác nhau cho phép kiểm soát quyền truy cập chi tiết hơn.
* Giới hạn quyền OS của DBMS: Giảm thiểu đặc quyền của tài khoản hệ điều hành mà hệ quản trị cơ sở dữ liệu (DBMS) đang chạy dưới đó. Không nên chạy DBMS dưới quyền root (Linux) hoặc system (Windows).
* Sử dụng SQL Views: Để tăng cường chi tiết hóa quyền truy cập, có thể tạo các SQL Views để giới hạn quyền đọc chỉ trên các trường cụ thể của bảng hoặc các join của bảng. Điều này đặc biệt hữu ích để bảo vệ dữ liệu nhạy cảm như mật khẩu (chỉ hiển thị hash thay vì plaintext).
* Khi sử dụng Stored Procedures: Nếu ứng dụng chỉ tương tác với cơ sở dữ liệu thông qua stored procedures, hãy giới hạn tài khoản ứng dụng chỉ được phép thực thi các stored procedures cần thiết, mà không cấp quyền trực tiếp vào các bảng

## Các biện pháp phòng chống bổ sung

Tường lửa ứng dụng web (Web Application Firewall - WAF): WAF hoạt động như một lớp bảo vệ bổ sung bằng cách kiểm tra lưu lượng truy cập ở cấp ứng dụng và có thể phát hiện, chặn các cuộc tấn công SQL Injection. Tuy nhiên, WAF cần được bảo trì và cập nhật chữ ký thường xuyên để duy trì hiệu quả.

Hệ thống phát hiện/ngăn chặn xâm nhập (Intrusion Detection System/Intrusion Prevention System - IDS/IPS): Là các công cụ giúp phát hiện và ngăn chặn các hoạt động độc hại, bao gồm các dấu hiệu của SQL Injection trên nhật ký (logs) và hành vi ứng dụng.

Cập nhật bảo mật định kỳ: Đảm bảo rằng tất cả phần mềm, bao gồm hệ điều hành, DBMS và các thư viện ứng dụng, luôn được cập nhật với các bản vá bảo mật mới nhất. Điều này làm cho việc khai thác các lỗ hổng đã biết trở nên khó khăn hơn

Đào tạo nhân viên: Đào tạo đội ngũ phát triển và các bên liên quan về các thực hành mã hóa an toàn và nhận thức về các mối đe dọa như SQL Injection

# VI. Kết luận

SQL Injection (SQLi) được coi là một trong những mối đe dọa nguy hiểm nhất trong thế giới an ninh ứng dụng web. Cuộc tấn công này xảy ra khi kẻ tấn công chèn thành công mã độc vào một ứng dụng web và các mã được chèn này tương tác thành công với truy vấn cơ sở dữ liệu. Nguyên nhân gốc rễ của SQL Injection là do các nhà phát triển tạo ra các truy vấn SQL bằng cách nối chuỗi với dữ liệu đầu vào do người dùng cung cấp mà không được xác thực hoặc xử lý đúng cách. Khi dữ liệu độc hại này được chèn vào, nó có thể thay đổi cấu trúc của câu lệnh SQL, lừa cơ sở dữ liệu thực thi các lệnh ngoài ý muốn của nhà phát triển. Mặc dù đã tồn tại từ cuối những năm 1990, SQLi vẫn là một lỗ hổng phổ biến. Trong năm 2023, SQL Injection chiếm 23% các lỗ hổng nghiêm trọng được phát hiện trên các ứng dụng web toàn cầu, giảm từ 33% trong năm 2022. Mặc dù các kỹ thuật phòng chống SQLi hiện tại đã chứng minh được hiệu quả đáng kể, vẫn có những hướng nghiên cứu và phát triển tiềm năng để nâng cao khả năng bảo vệ như tự động hóa các phương pháp phòng chống, ứng dụng AI trong phát hiện và phòng chống SQLi,...