**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

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

**─────── \* ───────**

****

**BÁO CÁO TUẦN 4**

Môn học: Project II (IT - 3931)

Đề tài: Network Security with IDS/IPS: Detection and Prevention of Network Attacks in a Virtualized Environment

|  |  |
| --- | --- |
| **Sinh viên thực hiện:** | **Ngô Trung Hiếu - 20225316**  **Kỹ thuật máy tính 04 – K67** |
| **Giảng viên hướng dẫn:** | **Nguyễn Quốc Khánh** |

**Hà Nội – 2025**

**Mục lục**

[I. Tổng quan về SQL injection 4](#_Toc211011699)

[1. SQL injection (SQLi) là gì? 4](#_Toc211011700)

[2. Tác hại của một cuộc tấn công SQL injection thành công 4](#_Toc211011701)

[3. Cách phát hiện lỗ hổng SQL injection 4](#_Toc211011702)

[4. SQL injection xảy ra trong các bộ phận của câu query 4](#_Toc211011703)

[5. Ví dụ SQL injection 4](#_Toc211011704)

[a. Truy xuất dữ liệu ẩn 4](#_Toc211011705)

[b. Làm sai lệch logic ứng dụng 4](#_Toc211011706)

[c. Tấn công UNION 4](#_Toc211011707)

[d. Blind SQL injection 4](#_Toc211011708)

[6. Second-Order SQL injection 4](#_Toc211011709)

[a. Phân biệt First-Order và Second-Order SQL Injection 4](#_Toc211011710)

[b. Quá trình hoạt động của Second-Order SQL Injection 4](#_Toc211011711)

[c. Tại sao Second-Order SQLi lại phổ biến? 5](#_Toc211011712)

# Tổng quan về SQL injection

## SQL injection (SQLi) là gì?

SQL injection (SQLi) là một lỗ hổng bảo mật trên web cho phép kẻ tấn công can thiệp vào các câu truy vấn mà ứng dụng gửi tới cơ sở dữ liệu. Từ đó kẻ tấn công có thể xem dữ liệu mà lẽ ra họ không được truy xuất — ví dụ dữ liệu của người dùng khác hoặc bất kỳ dữ liệu nào ứng dụng có quyền truy cập. Trong nhiều trường hợp, kẻ tấn công còn có thể sửa đổi hoặc xoá dữ liệu, gây ra thay đổi tồn tại lâu dài trên nội dung hoặc hành vi của ứng dụng.

Trong một số tình huống, kẻ tấn công có thể leo thang từ SQLi để xâm phạm máy chủ nền tảng hoặc các thành phần backend khác. SQLi cũng có thể được lợi dụng để thực hiện các cuộc tấn công từ chối dịch vụ (DoS).

## Tác hại của một cuộc tấn công SQL injection thành công

SQLi thành công có thể dẫn tới truy cập trái phép dữ liệu nhạy cảm, ví dụ: mật khẩu, thông tin thẻ tín dụng, thông tin cá nhân của người dùng. SQLi là nguyên nhân của nhiều vụ rò rỉ dữ liệu lớn, gây tổn thất danh tiếng và phạt hành chính; trong một số vụ, kẻ tấn công thiết lập được backdoor bền vững dẫn tới xâm phạm lâu dài mà khó phát hiện.

## Cách phát hiện lỗ hổng SQL injection

Bạn có thể phát hiện SQLi thủ công bằng một bộ kiểm thử hệ thống trên mọi điểm nhập dữ liệu của ứng dụng — thường bằng cách gửi vào:

* Ký tự nháy đơn ' và quan sát lỗi hoặc các bất thường.
* Các cấu trúc SQL đặc thù để so sánh kết quả đầu vào về giá trị gốc và giá trị khác, rồi đối chiếu khác biệt trong phản hồi.
* Điều kiện Boolean như OR 1=1 và OR 1=2, quan sát khác biệt trong phản hồi của ứng dụng.
* Payload gây trễ thời gian (time-based) khi được thực thi trong câu SQL, rồi so sánh thời gian phản hồi.

Ngoài ra, bạn có thể tìm phần lớn lỗ hổng SQLi nhanh hơn và đáng tin cậy hơn bằng công cụ quét tự động như Burp Scanner.

## SQL injection xảy ra trong các bộ phận của câu query

Hầu hết lỗ hổng SQL injection xảy ra trong mệnh đề WHERE của câu SELECT. Đây là loại mà nhiều người kiểm thử quen thuộc.  
Tuy nhiên, SQLi có thể xuất hiện ở bất kỳ vị trí nào trong câu SQL và trong các loại câu khác nhau. Một vài vị trí phổ biến khác là:

* Trong câu UPDATE, ở phần giá trị được cập nhật hoặc trong mệnh đề WHERE.
* Trong câu INSERT, ở phần giá trị được chèn.
* Trong SELECT, ở tên bảng hoặc tên cột.
* Trong SELECT, ở mệnh đề ORDER BY.

## Ví dụ SQL injection

Có nhiều kiểu lỗ hổng, phương pháp tấn công và kỹ thuật SQLi xuất hiện trong các tình huống khác nhau. Một số ví dụ phổ biến:

* **Truy xuất dữ liệu ẩn:** bạn sửa câu SQL để trả về nhiều kết quả hơn.
* **Làm sai lệch logic ứng dụng:** thay đổi truy vấn để can thiệp logic ứng dụng.
* **Tấn công UNION:** sử dụng UNION để lấy dữ liệu từ các bảng khác.
* **Blind SQL injection:** kết quả truy vấn do bạn điều khiển không được trả trực tiếp trong phản hồi — bạn dò bằng thời gian/boolean/OOB.

### Truy xuất dữ liệu ẩn

Giả sử ứng dụng mua sắm hiển thị sản phẩm theo category. Khi người dùng bấm category Gifts, trình duyệt gọi:

https://insecure-website.com/products?category=Gifts

Ứng dụng tạo câu SQL:

SELECT \* FROM products WHERE category = 'Gifts' AND released = 1

Ý nghĩa: lấy tất cả cột (\*) từ products mà category='Gifts' và released=1 (chỉ sản phẩm đã phát hành). Giả sử sản phẩm chưa phát hành có released=0.

Nếu **không** có biện pháp bảo vệ, attacker có thể gửi:

https://insecure-website.com/products?category=Gifts'--

Thành câu SQL:

SELECT \* FROM products WHERE category = 'Gifts'--' AND released = 1

-- là comment trong SQL → phần còn lại bị bỏ qua → điều kiện AND released = 1 bị xóa. Kết quả: tất cả sản phẩm (kể cả chưa phát hành) được hiển thị.

Hoặc:

https://insecure-website.com/products?category=Gifts'+OR+1=1--

Thành:

SELECT \* FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

Vì 1=1 luôn đúng, truy vấn trả về tất cả hàng — tức toàn bộ sản phẩm, bất kể category hay released.

**Cảnh báo (Warning)**  
Cẩn thận khi chèn OR 1=1 — dữ liệu từ một request có thể được dùng ở nhiều câu truy vấn. Nếu cùng input tới được UPDATE hoặc DELETE, payload như OR 1=1 có thể gây mất dữ liệu (ví dụ làm cho DELETE WHERE ... OR 1=1 xóa toàn bộ bảng).

### Làm sai lệch logic ứng dụng

Giả sử một ứng dụng cho phép đăng nhập bằng username và password. Nếu người dùng gửi username = wiener và password = bluecheese, ứng dụng kiểm tra bằng câu SQL:

SELECT \* FROM users WHERE username = 'wiener' AND password = 'bluecheese'

Nếu câu truy vấn trả về thông tin người dùng thì đăng nhập thành công, ngược lại bị từ chối.

Trong ví dụ này, kẻ tấn công có thể đăng nhập như bất kỳ người dùng nào mà không cần biết mật khẩu — bằng cách dùng chuỗi comment -- để loại bỏ phần kiểm tra mật khẩu trong mệnh đề WHERE. Ví dụ, gửi username = administrator'-- và password rỗng tạo ra câu:

SELECT \* FROM users WHERE username = 'administrator'--' AND password = ''

Phần sau -- bị coi là comment, do đó điều kiện kiểm tra password bị loại bỏ — truy vấn trả về user administrator và kẻ tấn công được đăng nhập dưới danh nghĩa user đó.

### Tấn công UNION

Truy xuất dữ liệu từ các bảng khác: Khi ứng dụng trả kết quả của một câu SQL trong phản hồi, kẻ tấn công có thể lợi dụng SQLi để lấy dữ liệu từ các bảng khác trong cơ sở dữ liệu. Từ khóa UNION được dùng để chạy một SELECT bổ sung và ghép (append) kết quả của nó vào kết quả ban đầu.

Ví dụ, ứng dụng chạy câu (với input là Gifts):

SELECT name, description FROM products WHERE category = 'Gifts'

Kẻ tấn công có thể gửi input:

' UNION SELECT username, password FROM users--

Khi ghép vào, truy vấn trả về cả name, description của products và username, password của users — tức là lộ danh sách username/password.

### Blind SQL injection

## Second-Order SQL injection

### Phân biệt First-Order và Second-Order SQL Injection

|  |  |  |
| --- | --- | --- |
| Loại Tấn Công | Đặc Điểm | Quá Trình (Dòng chảy dữ liệu) |
| First-Order SQLi (Cấp độ một) | Xảy ra ngay lập tức tại điểm nhập liệu. | Yêu cầu HTTP → Truy vấn SQL (thực thi) |
| Second-Order SQLi (Cấp độ hai) | Xảy ra trễ, tại một điểm khác sau khi dữ liệu đã được lưu trữ. | Yêu cầu HTTP → Lưu trữ DB → Yêu cầu HTTP khác → Truy vấn SQL (thực thi) |

Tên gọi khác: Second-order SQL injection còn được gọi là Stored SQL Injection (SQL Injection được lưu trữ), vì dữ liệu độc hại được lưu trữ trong cơ sở dữ liệu trước khi được kích hoạt.

### Quá trình hoạt động của Second-Order SQL Injection

Second-order SQLi là một cuộc tấn công hai giai đoạn:

**Giai đoạn 1:** Chèn (Insertion)

* Kẻ tấn công gửi dữ liệu độc hại (payload) thông qua một yêu cầu HTTP (ví dụ: đăng ký tài khoản, gửi bình luận).
* Ứng dụng xử lý dữ liệu này và lưu trữ nó vào cơ sở dữ liệu (DB).
* Lưu ý quan trọng: Tại thời điểm lưu trữ này, ứng dụng không hề có lỗ hổng. Dữ liệu được xử lý một cách an toàn (ví dụ: sử dụng câu lệnh Prepared Statement) nên không có truy vấn SQL độc hại nào được thực thi.

**Giai đoạn 2:** Kích hoạt (Activation)

* Sau đó, khi xử lý một yêu cầu HTTP khác (có thể là một trang quản trị, trang báo cáo, hoặc một chức năng khác), ứng dụng truy xuất dữ liệu đã lưu trữ trước đó.
* Ứng dụng kết hợp dữ liệu đã truy xuất này vào một truy vấn SQL theo cách không an toàn.
* Lỗ hổng xảy ra tại đây: Payload độc hại được lấy ra từ DB và kích hoạt thành công trong truy vấn mới này.

### Tại sao Second-Order SQLi lại phổ biến?

Lỗ hổng này thường xảy ra khi các nhà phát triển:

* **Chỉ tập trung bảo mật ở điểm nhập liệu đầu tiên:** Họ đã cẩn thận khi lưu dữ liệu vào DB, khiến họ tin rằng dữ liệu đó đã "sạch" và an toàn.
* **Mặc định coi dữ liệu đã lưu trữ là đáng tin cậy:** Khi dữ liệu được truy xuất ra, nó được xem là trusted data (dữ liệu đáng tin cậy) vì nó đến từ cơ sở dữ liệu nội bộ.
* **Xử lý không an toàn ở điểm truy xuất:** Do tin tưởng, họ sử dụng phương pháp **nối chuỗi** (string concatenation) thay vì các phương pháp an toàn (như parameterized queries) khi tạo truy vấn SQL mới bằng dữ liệu đã lưu.

Lỗ hổng này khai thác sự mất cảnh giác và giả định sai lầm của nhà phát triển về tính an toàn của dữ liệu đã được lưu trữ.