# Mã sinh viên: 22A1001D0363 Họ và tên: Đậu Huy Văn

# Báo cáo A03:2021 – Injection

## 1. Mô tả (Overview & Description)

### Tổng quan (Overview)

Lỗi **A03:2021 – Injection** (Tấn công Chèn) đã tụt xuống vị trí thứ ba trong danh sách **OWASP Top 10** (năm 2021). Đây là một trong những loại lỗ hổng bảo mật phổ biến và nghiêm trọng nhất.

**Tỷ lệ kiểm thử (Coverage):** 94% các ứng dụng được kiểm thử đã được xem xét về các hình thức Injection.

**Tỷ lệ xuất hiện (Incidence Rate):** Tỷ lệ xuất hiện tối đa là 19%, trung bình là 3.37%.

**Số lần xảy ra (Total Occurrences):** Ghi nhận 274,228 lần xuất hiện.

**Các lỗi CWEs liên quan đáng chú ý:**

**CWE-79:** Cross-site Scripting (XSS)

**CWE-89:** SQL Injection (SQLi)

**CWE-73:** External Control of File Name or Path.

### Mô tả Lỗ hổng (Description)

Một ứng dụng dễ bị tấn công Injection khi ứng dụng đó sử dụng **dữ liệu do người dùng cung cấp** (user-supplied data) mà không được **kiểm tra, lọc, hoặc làm sạch (sanitize)** đúng cách, khiến dữ liệu này được coi là một phần của lệnh hoặc truy vấn.

Các kịch bản làm ứng dụng dễ bị tấn công bao gồm:

**Dữ liệu đầu vào không được kiểm định (unvalidated):** Dữ liệu được cung cấp bởi người dùng không được xác thực, lọc, hoặc làm sạch bởi ứng dụng.

**Sử dụng truy vấn động/lời gọi không tham số (non-parameterized calls):** Sử dụng các truy vấn động (Dynamic queries) hoặc lời gọi không tham số mà không có cơ chế thoát ký tự (context-aware escaping) được sử dụng trực tiếp trong trình thông dịch (interpreter).

**Sử dụng dữ liệu độc hại (Hostile data) trong ORM:** Dữ liệu độc hại được sử dụng trong các tham số tìm kiếm của công cụ Ánh xạ Quan hệ Đối tượng (Object-Relational Mapping - ORM) để trích xuất các bản ghi nhạy cảm, bổ sung.

**Nối chuỗi dữ liệu độc hại:** Dữ liệu độc hại được sử dụng hoặc nối trực tiếp. Lệnh SQL hoặc lệnh shell chứa cả cấu trúc và dữ liệu độc hại trong các truy vấn động, lệnh hoặc thủ tục lưu trữ (stored procedures).

Các loại Injection phổ biến: **SQL**, **NoSQL**, lệnh **OS** (hệ điều hành), **ORM**, **LDAP**, và **Expression Language (EL)** hoặc **Object Graph Navigation Library (OGNL)** injection. Khái niệm cơ bản giữa tất cả các trình thông dịch là giống nhau.

## 2. Cơ chế Tấn công và Ví dụ Minh họa (Example Attack Scenarios)

### Cơ chế Tấn công (Attack Mechanism)

Kẻ tấn công chèn các lệnh (command) hoặc truy vấn (query) độc hại vào luồng dữ liệu của ứng dụng. Khi ứng dụng xử lý dữ liệu này và coi nó là một phần của lệnh hợp pháp, trình thông dịch (ví dụ: cơ sở dữ liệu, hệ điều hành) sẽ thực thi lệnh độc hại đó, dẫn đến:

**Lộ dữ liệu nhạy cảm:** Truy xuất thông tin không được phép (ví dụ: thông tin người dùng, mật khẩu băm).

**Sửa đổi hoặc xóa dữ liệu:** Thao túng hoặc phá hủy dữ liệu trong cơ sở dữ liệu.

**Từ chối dịch vụ (DoS):** Làm chậm hoặc ngừng hoạt động của ứng dụng/hệ thống.

**Thực thi mã từ xa (RCE):** Thực thi các lệnh trên máy chủ.

### Ví dụ Minh họa Tấn công (Example Attack Scenarios)

#### Kịch bản #1: SQL Injection Cơ bản (Vulnerable SQL Call)

Một ứng dụng sử dụng dữ liệu không đáng tin cậy để xây dựng truy vấn SQL sau:

SQL

String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";

**Hành vi Tấn công:** Kẻ tấn công thay đổi giá trị tham số 'id' trên trình duyệt thành: **' UNION SELECT SLEEP(10);--** *Ví dụ URL:* http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--

**Truy vấn sau khi bị chèn:**

SQL

SELECT \* FROM accounts WHERE custID='' UNION SELECT SLEEP(10);--'

Dấu ' đầu tiên đóng chuỗi ' ban đầu.

Lệnh **UNION SELECT SLEEP(10)** được chèn vào để buộc cơ sở dữ liệu tạm dừng (sleep) trong 10 giây (dùng để kiểm tra lỗ hổng Blind SQL Injection).

**--** là ký hiệu bình luận (comment) trong SQL, nó vô hiệu hóa dấu ' cuối cùng của truy vấn gốc.

**Hậu quả:** Kẻ tấn công có thể thay đổi ý nghĩa của truy vấn để trích xuất thêm bản ghi hoặc thực hiện các hành động nguy hiểm hơn như xóa/sửa dữ liệu hoặc gọi thủ tục lưu trữ.

#### Kịch bản #2: SQL Injection qua ORM (Hibernate Query Language - HQL)

Một ứng dụng sử dụng framework ORM như Hibernate nhưng vẫn bị lỗi do nối chuỗi trong HQL:

Java

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

Kẻ tấn công sử dụng payload tương tự như Kịch bản #1. Dù là framework, việc nối chuỗi trực tiếp vẫn tạo ra lỗ hổng tương tự.

## 3. Biện pháp Phòng chống (How to Prevent)

Phòng chống Injection đòi hỏi nguyên tắc cốt lõi là **tách biệt dữ liệu khỏi các lệnh và truy vấn**.

### Phương pháp Ưu tiên (Preferred Options)

**Sử dụng API an toàn (Safe API):**

Sử dụng các giao diện API an toàn, tránh sử dụng trực tiếp trình thông dịch (interpreter) nếu có thể.

Sử dụng giao diện **tham số hóa (parameterized interface)**.

Chuyển sang sử dụng **Công cụ Ánh xạ Quan hệ Đối tượng (ORMs)**.

*Lưu ý:* Ngay cả khi sử dụng thủ tục lưu trữ (stored procedures) đã được tham số hóa, lỗ hổng SQL Injection vẫn có thể xảy ra nếu mã PL/SQL hoặc T-SQL bên trong thủ tục vẫn nối chuỗi truy vấn với dữ liệu hoặc thực thi dữ liệu độc hại bằng **EXECUTE IMMEDIATE** hoặc **exec()**.

### Các Biện pháp Bổ sung

**Kiểm tra Đầu vào Phía Máy chủ (Server-Side Input Validation):**

Sử dụng **xác thực đầu vào dương tính (Positive Server-side input validation)**: chỉ chấp nhận các ký tự đã biết là an toàn (Whitelist).

*Lưu ý:* Đây không phải là biện pháp phòng thủ hoàn chỉnh vì nhiều ứng dụng vẫn cần các ký tự đặc biệt (ví dụ: các ô văn bản lớn, API ứng dụng di động).

**Thoát Ký tự Đặc biệt (Escape Special Characters):**

Đối với các truy vấn động còn lại, hãy **thoát các ký tự đặc biệt** bằng cách sử dụng cú pháp thoát cụ thể cho từng trình thông dịch.

*Lưu ý:* Các cấu trúc SQL như tên bảng, tên cột, v.v., không thể thoát được, do đó, việc sử dụng tên cấu trúc do người dùng cung cấp là rất nguy hiểm (thường gặp trong phần mềm báo cáo).

### Công cụ và Quy trình

**Đánh giá mã nguồn (Source Code Review):** Đây là phương pháp tốt nhất để phát hiện lỗi Injection.

**Kiểm thử tự động:** Khuyến khích kiểm thử tự động tất cả các đầu vào (tham số, tiêu đề, URL, cookie, JSON, SOAP, XML, v.v.).

**Tích hợp công cụ bảo mật vào CI/CD:** Sử dụng các công cụ kiểm thử bảo mật ứng dụng **Tĩnh (SAST)**, **Động (DAST)** và **Tương tác (IAST)** vào quy trình Tích hợp và Triển khai Liên tục (CI/CD) để xác định các lỗ hổng Injection trước khi triển khai sản xuất.