

**SQL Injection**

**Final Chuyên đề bảo mật – Team 15 K16T1**



[1. Thành viên 1](#_Toc375828623)

[2. Giới thiệu SQL Injection 1](#_Toc375828624)

[3. Nhận diện điểm yếu SQL injection trong ứng dụng web 1](#_Toc375828625)

[3.1. Thăm dò dựa trên phản hồi 1](#_Toc375828626)

[*3.1.1.* Xác định các điểm nhận input từ client 2](#_Toc375828627)

[*3.1.2.* Các hình thức trả thông báo lỗi thường gặp 2](#_Toc375828628)

[3.2. Cơ chế sinh truy vấn SQL bên trong ứng dụng và các phương pháp chèn truy vấn SQL 3](#_Toc375828629)

[*3.2.1.* Cơ chế sinh truy vấn SQL bên trong ứng dụng 3](#_Toc375828630)

[*3.2.2.* Các phương pháp chèn tham số 4](#_Toc375828631)

[4. Các dạng lỗi thường gặp 6](#_Toc375828632)

[4.1. Không kiểm tra ký tự thoát truy vấn 6](#_Toc375828633)

[4.2. Xử lý không đúng kiểu 6](#_Toc375828634)

[4.3. Lỗi bảo mật bên trong máy chủ cơ sở dữ liệu 7](#_Toc375828635)

[4.4. Blind SQL injection 7](#_Toc375828636)

[*4.4.1.* Thay đổi giá trị điều kiện truy vấn 7](#_Toc375828637)

[*4.4.2.* Điều kiện lỗi 7](#_Toc375828638)

[*4.4.3.* Thời gian trễ 7](#_Toc375828639)

[5. Một số dạng tấn công thường gặp với các ứng dụng web 8](#_Toc375828640)

[5.1. Dạng tấn công vượt qua kiểm tra lúc đăng nhập 8](#_Toc375828641)

[5.2. Dạng tấn công sử dụng câu lệnh SELECT 9](#_Toc375828642)

[5.3. Dạng tấn công sử dụng câu lệnh INSERT 9](#_Toc375828643)

[5.4. Dạng tấn công sử dụng stored-procedures 10](#_Toc375828644)

[6. Phòng chống SQL injection 10](#_Toc375828645)

[6.1. Kiểm soát chặt chẽ dữ liệu nhập vào 10](#_Toc375828646)

[6.2. Thiết lập cấu hình an toàn cho hệ quản trị cơ sở dữ liệu 11](#_Toc375828647)

[6.3. Phòng chống từ mức xây dụng mã nguồn ứng dụng 11](#_Toc375828648)

[*6.3.1.* Làm sạch dữ liệu đầu vào 11](#_Toc375828649)

[*6.3.2.* Xây dựng truy vấn theo mô hình tham số hóa 13](#_Toc375828650)

**MỤC LỤC**

|  |  |  |  |
| --- | --- | --- | --- |
| Ngày | Phiên bản | Người viết | Mô tả |
| 20/12/2013 | 1.0 | Lê Ngọc Châu | Lập tài liệu |
| 26/12/2013 | 1.1 | BigFiveTeam | Cập nhật nội dung tài liệu |

**LỊCH SỬ TÀI LIỆU**

# Thành viên

|  |  |
| --- | --- |
| T094054 | Trịnh Thái Anh |
| T103573 | Lê Ngọc Châu |
| T105026 | Khấu Thành Đạo |
| T104898 | Huỳnh Trọng Khang |
| T101184 | Tạ Ngọc Thiên Phú |
| T103569 | Nguyễn Hoàng Fa Thứ |

# Giới thiệu SQL Injection

SQL injection (còn gọi là SQL Insertion) cho phép những kẻ tấn công lợi dụng lỗ hổng của việc kiểm tra dữ liệu đầu vào trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu trả về để inject (tiêm vào) và thi hành các câu lệnh SQL bất hợp pháp, Sql injection có thể cho phép những kẻ tấn công thực hiện các thao tác, delete, insert, update,… trên cơ sỡ dữ liệu của ứng dụng, thậm chí là server mà ứng dụng đó đang chạy, lỗi này thường xãy ra trên các ứng dụng web có dữ liệu được quản lý bằng các hệ quản trị cơ sở dữ liệu như SQL Server, MySQL, Oracle, DB2, Sysbase..

Hình thái chính của SQL injection bao gồm việc chèn trực tiếp mã vào các tham số mà sẽ được ghép vào các câu lệnh SQL (quá trình này gọi là sinh truy vấn SQL động) để tạo thành truy vấn của ứng dụng gửi tới máy chủ database. Một cách tấn công khác ít trực tiếp hơn, đó là chèn mã độc vào các xâu mà đích đến là việc lưu trữ trong các bảng hoặc từ điển dữ liệu (metadata). Khi các chuỗi đó được ghép vào các câu lệnh SQL thì đoạn mã đó sẽ được chạy.

Khi ứng dụng Web thất bại trong việc lọc các tham số đầu vào (được dùng làm nguyên liệu cho quá trình sinh SQL động), ngay cả khi dùng hình thức tham số hóa (parameterize) thì kẻ tấn công có thể dễ dàng điều chỉnh quá trình xây dựng truy vấn SQL. Một khi kẻ tấn công có thể sửa câu truy vấn SQL, thì những truy cấn SQL anh ta muốn sẽ được thực thi với quyền của người sở hữu ứng dụng, và thiệt hại anh ta có thể gây ra sẽ tùy theo quyền hạn được cấp.

SQL injection là một dạng tấn công dễ thực hiện, hầu hết mọi thao tác người tấn công cần được thực hiện với một trình duyệt web, có thể kèm theo một ứng dụng proxy server. Chính vì đơn giản như vậy cho nên bất cứ ai cũng có thể học cách tiến hành một cuộc tấn công. Lỗi bắt nguồn từ mã nguồn của ứng dụng web chứ không phải từ phía database, chính vì thế bất cứ thành phần nào của ứng dụng mà người dùng có thể tương tác được để điều khiển nội dung (ví dụ: các form, tham số URL, cookie, tham số referrer, user-agent,…) đều có thể được sử dụng để tiến hành chèn truy vấn có hại.

# Nhận diện điểm yếu SQL injection trong ứng dụng web

Công việc nhận diện điểm yếu này là công việc đầu tiên trong chuỗi các thao tác cần để khắc phục điểm yếu SQL injection trong ứng dụng. Công việc này được thực hiện tương tự các thao tác hacker tiến hành thăm dò lỗi SQL injection của ứng dụng. Chúng ta xét một số công việc cần thực hiện trong quá trình thăm dò lỗi SQL injection.

## Thăm dò dựa trên phản hồi

Thăm dò dựa trên phản hồi là phương pháp tự nhiên nhất. Chúng ta cần tối thiểu là một trình duyệt web, có thể trang bị thêm một ứng dụng Proxy (ví dụ Burp proxy, Web Scarab proxy,…) , tiến hành các phép thử SQL injection ngầu nhiên và tiến hành phân tích, thống kê kết quả. Các bước tiến hành gồm có:

- Xác định tất cả các điểm nhận input từ client.

- Thử và xác định đặc điểm chung của những request có phát sinh kết quả bất thường.

- Xác định nguyên nhân xác điểm bất thường đó

### Xác định các điểm nhận input từ client

Phía client trong mô hình Client/ Server trong môi trường Web chính là trình duyệt Web. Những điểm nhận input phổ biến nhất từ client là đường dẫn (link), khung nhập dữ liệu (form), cookie, ….Sau khi thực hiện input, trình duyệt Web sẽ sinh một request phổ biến nhất là GET và POST.

Cấu trúc thông điệp GET và POST có nhiều điểm khác nhau, xong khi tiến hành sửa đổi và chèn nội dung (inject) chúng ta cần chú ý tới vị trí của chuỗi truy vấn (query string). Chuỗi truy vấn này chứa các chuỗi tham số được gửi lên web server, chuỗi này có dạng sau:

|  |
| --- |
| ?var\_1=val\_1&var\_2=val\_2& … &var\_n=val\_n |

Trong thông điệp GET chuỗi truy vấn nằm ở đầu thông điệp, trong khi ở POST nó nằm ở cuối thông điệp.

Xét một trang thông tin có đường dẫn: <http://www.site.com/categories_index.php>

Nội dung trang trên có các đường liên kết (link), khi click chuột vào từng liên kết đó sẽ dẫn tới các địa chỉ dạng như:

|  |
| --- |
| <http://www.site.com/categories_index.php?cat_name=tin_nong>  <http://www.site.com/categories_index.php?cat_name=tin_the_gioi>  <http://www.site.com/categories_index.php?cat_name=tin_dia_phuong> |

Trong trường hợp này thông điệp request là GET bởi chuỗi truy vấn (query string) được hiển thị ngay trên trình duyệt. Tham số xuất hiện trong trường họp này là cat\_name, ứng với mỗi giá trị cat\_name thì nội dung trả về sẽ khác nhau. Thực hiện sửa nội dung cat\_name rồi gửi, với đường dẫn:

<http://www.site.com/categories_index.php?cat_name=nothing>

Kết quả trả về sẽ có thể là một thông báo lỗi dạng sau:

|  |
| --- |
| Warning: mysqli\_fetch\_object() expects parameter 1 to be mysqli\_result, Boolean given in **/home1/thangmom/public\_html/includes/functions.php** on line **225** |

Thử thêm dấu nháy đơn (‘) vào cuối giá trị tham số cat\_name, ta có kết quả trả về cho đường dẫn:

<http://www.site.com/categories_index.php?cat_name=nothing>’

|  |
| --- |
| 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 |

Như vậy chúng ta nhận thấy có những dấu hiệu bất thường trong phản hồi ứng với các giá trị tham số được chỉnh sửa khác nhau.

### Các hình thức trả thông báo lỗi thường gặp

Những thông báo lỗi trả về ở trên những thông báo lỗi chi tiết, chúng là “trợ giúp đắc lực” cho hacker trong việc khai thác thông tin từ database của ứng dụng. Ngoài cách hiển thị chi tiết này ra Webserver sẽ có những lựa chọn sau:

* Nội dung lỗi được giấu đi nhằm mục đích gỡ lỗi trong mã nguồn.
* Trả về một mã lỗi HTTP, ví dụ 500 (internal server error), 302 (redirect)…
* Ứng dụng bắt lỗi, xử lý nó bằng cách không trả về kết quả gì, hoặc trả về một trang thông báo lỗi tổng quát. Trang này được cấu hình, ví dụ trong Apache2 là file conf.d/localized-error-pages.

Trường hợp ứng dụng cấu hình một trang mặc định được trả về trong trường hợp sinh lỗi là trường hợp khó nhận diện điểm yếu hơn cả, bởi có nhiều lý do có thể sinh lỗi, không chỉ riền trường hợp chúng ta chèn tham số.

Trong các trường hợp điểm yếu SQL injection tồn tại, có một trường hợp khó phát hiện hơn cả, đó là trường hợp Blind SQL injection. Thông thường, các tham số từ chuỗi truy vấn được dùng để xây dựng câu truy vấn SQL, ví dụ với đoạn URL university.php?searchkey=’vnu’. Có thể được sử dụng để xây dựng truy vấn ví dụ như:

|  |
| --- |
| SELECT \* FROM university WHERE name like ‘%vnu%’; |

Blind SQP injection là một dạng tấn công mà không thể dựa vào các thông báo lỗi thông thường, mà chỉ có thể dựa vào sự khác nhau trong phản hồi giữa hai trường hợp đúng/ sai của mệnh đề WHERE. Ví dụ, hai truy vấn ứng với hai trường hợp tham số nhập vào sau của trang university.php sau:

|  |
| --- |
| Tham số **vnu%’ or 1=1--**  ta có truy vấn:  SELECT \* FROM university WHERE name like ‘%vnu%’ or 1=1--%”  Tham số **vnu’ and 1=0—**ta có truy vấn:  SELECT \* FROM university WHERE name like ‘%vnu%’ and 1=0--%” |

Hai truy vấn trên khác nhau ở chỗ truy vấn thứ nhất có mệnh đề WHERE luôn đúng, còn truy vấn thứ hai có mệnh đề luôn sai. Nếu kết quả trả về có sự khác biệt giữa hai trường hợp tham số này với nhau và với trường hợp tham số không bị chỉnh sửa thì rất có thể tồn tại điểm yếu dạng blind SQL injection.

## Cơ chế sinh truy vấn SQL bên trong ứng dụng và các phương pháp chèn truy vấn SQL

### Cơ chế sinh truy vấn SQL bên trong ứng dụng

Tham số được nhập vào sẽ được sử dụng để xây dựng các truy vấn SQL nên nó sẽ cần thỏa mãn các ràng buộc cú pháp với thành phần trước và sau trong truy vấn gốc. Xét đoạn mã PHP xử lý đăng nhập sau:

|  |
| --- |
| <?php  $uname = isset($\_POST[‘uname’]) ? $\_POST[‘uname’] : “”;  $passwd= isset($\_POST[‘passwd’]) ? $\_POST[‘passwd’] : “”;  $query = “SELECT \* FROM tbl\_users WHERE username = “” + $uname + “’ AND password = ‘”+ $passwd + “”;  $qr = @mysql\_query(@query);  …  ?> |

Xâu truy cấn SQL được sinh ra trong trường hợp trên sử dụng trực tiếp giá trị input được người đăng nhập vào, do đó mô hình xây dựng truy vấn dạng này được gọi chung là xây dựng truy vấn động (dynamic query). Truy vấn thu được sẽ có dạng như sau:

SELECT \* FROM tbl\_users WHERE username = ‘$uname’AND password = ‘$passwd’;

Trong đó hai giá trị $name và $passwd được nhập từ người dùng. Khi thực hiện nhập giá trị username là **admin’ or ‘1’=’1** truy vấn động thu được sẽ nhu sau:

SELECT \* FROM tbl\_users WHERE username=’admin’or ‘1’=’1 và password=”;

Truy vấn này tuy có cụm luôn đúng, nhưng do toán tử AND có độ ưu tiên cao hơn OR do đó truy vấn trên tương đương với:

SELECT \* FROM tbl\_users WHERE username=”admin’ AND password=”;

Trường hợp này rõ ràng đăng nhập thất bại. Tiếp tục thử với việc thêm cả cụm **‘or ‘1’=’1** vào cả password, ta có truy vấn được sinh ra:

SELECT \* FROM tbl\_users WHERE username’=admin’ AND password=” or ‘1’=’1;

Truy vấn trên tương đương với:

SELECT \* FROM tbl\_users WHERE username=’admin’ AND password=” or ‘1’=’1’;

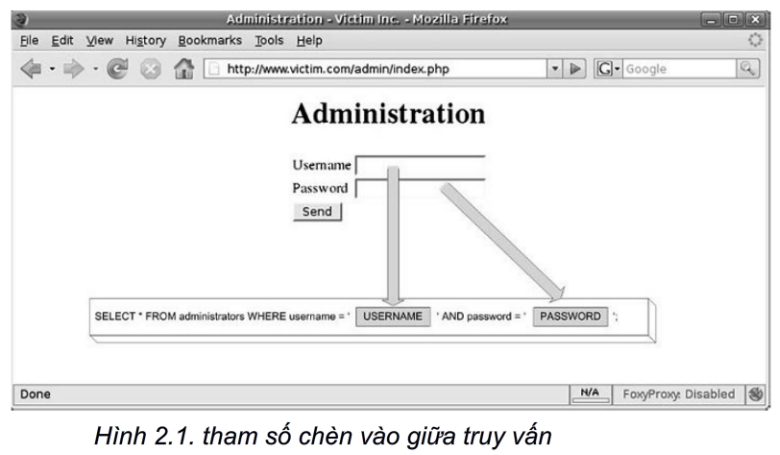
Trường hợp này việc xác thực đã thành công do mệnh đề WHERE luôn đúng. Ngoài cách trên ta có thể thực hiện chèn thêm một đoạn **or ‘1’=’1**  vào username, tức là **admin’ or ‘1’=’1’ or ‘1’=’1**  vào, kết quả thu được cũng tương tự, do toán tử AND đã được “khủ” trước các toán tử OR.

### Các phương pháp chèn tham số

Tùy thuộc vào câu truy vấn gốc mà các tham số được chèn vào sẽ có vị trí khác nhau trong truy vấn đó. Ứng với từng trường hợp đó, chúng ta có các mô hình chèn tham số sau:

#### Chèn vào giữa truy vấn

Chèn vào giữa truy vấn là mô hình chỉ đơn thuần thao tác với tham số, không hề tác động đến cấu trúc và các thành phần của truy vấn gốc. Việc chèn như minh họa ở phần a, chính là chèn vào giữa truy vấn. Mô hình này có thể khái quát như sau:



#### Chèn và ngắt truy vấn

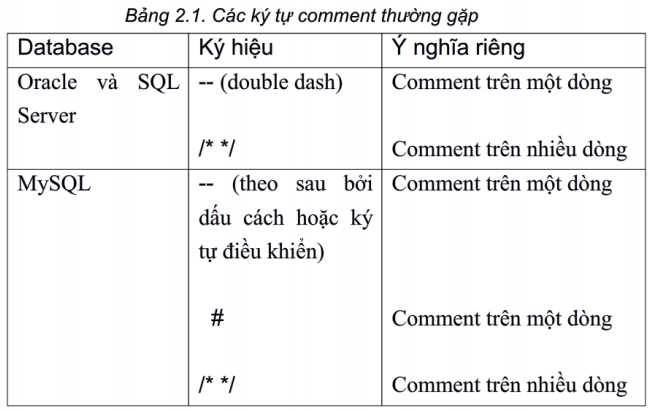
Đây là mô hình chèn truy vấn phổ biến nhất, truy vấn được chèn vào sẽ bao gồm thêm ở cuối các ký tự comment nhằm ngắt truy vấn tại đó, vô hiệu hóa các phần tử trong truy vấn gốc nằm phía sau vị trí tham số. Đoạn mã PHP đã nêu được cải tiến như sau:

|  |
| --- |
| <?php  $uname = isset($\_POST[‘uname’]) ? $\_POST[‘uname’]: “”;  $passwd= isset($\_POST[‘passwd’]) ? $\_POST[‘passwd’]: “”;  if($uname = = ‘” | | password is missing”;  }else{  if($passwd = = “”)  echo “password is missing”;  else if($uname = = “”)  echo “username is missing”;  else{  $query = “SELECT \* FROM tbl\_users WHERE username = “” + $uname  + ‘” AND password = “” + $passwd + “”;  $qr = @mysql\_query($query);  …  ?> |

Với đoạn xử lý trên, các trường username và password không thể để trống, tuy nhiên không nhất thiết phải chèn nhiều mệnh đề OR, chúng ta chỉ cần đảm bảo có giá trị trong hai trường đó và sử dụng comment để ngắt truy vấn sau khi xuất hiện mệnh đề **OR 1=1**  đầu tiên, ví dụ với username là **admin’ or 1=1--**  và password bất kì (khác rỗng) thì truy vấn sẽ có dạng:

SELECT \* FROM tbl\_users WHERE username = ‘admin’ or 1=1;

Và với truy vấn này hacker qua mặt được xác thực do mệnh đề WHERE luôn đúng. Một số ký tự comment hay dùng:



# Các dạng lỗi thường gặp

## Không kiểm tra ký tự thoát truy vấn

Đây là dạng lỗi SQL injection xảy ra khi thiếu đoạn mã kiểm tra dữ liệu đầu vào trong câu truy vấn SQL. Kết quả là người dùng cuối có thể thực hiện một số truy vấn không mong muốn đối với cơ sở dữ liệu của ứng dụng. Dòng mã sau sẽ minh họa lỗi này:

|  |
| --- |
| statement = "SELECT \* FROM users WHERE name = '" + userName + "';" |

Câu lệnh này được thiết kế để trả về các bản ghi tên người dùng cụ thể từ bảng những người dùng. Tuy nhiên, nếu biến "userName" được nhập chính xác theo một cách nào đó bởi người dùng ác ý, nó có thể trở thành một câu truy vấn SQL với mục đích khác hẳn so với mong muốn của tác giả đoạn mã trên. Ví dụ, ta nhập vào giá trị của biến **userName** như sau:

|  |
| --- |
| a’ or ‘t’=’t |

Khiến câu truy vấn có thể được hiểu như sau:

|  |
| --- |
| **SELECT** \* **FROM** users **WHERE** name = 'a' **OR** 't'='t'; |

Nếu đoạn mã trên được sử dụng trong một thủ tục xác thực thì ví dụ trên có thể được sử dụng để bắt buộc lựa chọn một tên người dùng hợp lệ bởi 't'='t' luôn đúng. Trong khi hầu hết các SQL server cho phép thực hiện nhiều truy vấn cùng lúc chỉ với một lần gọi, tuy nhiên một số SQL API như mysql\_query của php lại không cho phép điều đó vì lý do bảo mật. Điều này chỉ ngăn cản tin tặc tấn công bằng cách sử dụng các câu lệnh riêng rẽ mà không ngăn cản tin tặc thay đổi các từ trong cú pháp truy vấn. Các giá trị của biến "userName" trong câu truy vấn dưới đây sẽ gây ra việc xoá những người dùng từ bảng người dùng cũng tương tự như việc xóa tất cả các dữ liệu được từ bảng dữ liệu (về bản chất là tiết lộ các thông tin của mọi người dùng), ví dụ này minh họa bằng một API cho phép thực hiện nhiều truy vấn cùng lúc:

|  |
| --- |
| a';DROP TABLE users; SELECT \* FROM data WHERE 't' = 't |

Điều này đưa tới cú pháp cuối cùng của câu truy vấn trên như sau

|  |
| --- |
| **SELECT** \* **FROM** users **WHERE** name = 'a';**DROP** **TABLE** users; **SELECT** \* **FROM** **DATA** **WHERE** 't' = 't'; |

## Xử lý không đúng kiểu

Lỗi SQL injection dạng này thường xảy ra do lập trình viên hay người dùng định nghĩa đầu vào dữ liệu không rõ ràng hoặc thiếu bước kiểm tra và lọc kiểu dữ liệu đầu vào. Điều này có thể xảy ra khi một trường số được sử dụng trong truy vấn SQL nhưng lập trình viên lại thiếu bước kiểm tra dữ liệu đầu vào để xác minh kiểu của dữ liệu mà người dùng nhập vào có phải là số hay không. Ví dụ như sau:

|  |
| --- |
| statement := "SELECT \* FROM data WHERE id = " + a\_variable + ";" |

Ta có thể nhận thấy một cách rõ ràng ý định của tác giả đoạn mã trên là nhập vào một số tương ứng với trường id - trường số. Tuy nhiên, người dùng cuối, thay vì nhập vào một số, họ có thể nhập vào một chuỗi ký tự, và do vậy có thể trở thành một câu truy vấn SQL hoàn chỉnh mới mà bỏ qua ký tự thoát. Ví dụ, ta thiết lập giá trị của biến a\_variable là:

|  |
| --- |
| 1;DROP TABLE users |

khi đó, nó sẽ thực hiện thao tác xóa người dùng có id tương ứng khỏi cơ sở dữ liệu, vì câu truy vấn hoàn chỉnh đã được hiểu là:

|  |
| --- |
| **SELECT** \* **FROM** **DATA** **WHERE** id=1;**DROP** **TABLE** users; |

## Lỗi bảo mật bên trong máy chủ cơ sở dữ liệu

Đôi khi lỗ hổng có thể tồn tại chính trong phần mềm máy chủ cơ sở dữ liệu, như là trường hợp hàm mysql\_real\_escape\_string() của các máy chủ MySQL. Điều này sẽ cho phép kẻ tấn công có thể thực hiện một cuộc tấn công SQL injection thành công dựa trên những ký tự Unicode không thông thường ngay cả khi đầu nhập vào đang được thoát.

## Blind SQL injection

Lỗi SQL injection dạng này là dạng lỗi tồn tại ngay trong ứng dụng web nhưng hậu quả của chúng lại không hiển thị trực quan cho những kẻ tấn công. Nó có thể gây ra sự sai khác khi hiển thị nội dung của một trang chứa lỗi bảo mật này, hậu quả của sự tấn công SQL injection dạng này khiến cho lập trình viên hay người dùng phải mất rất nhiều thời gian để phục hồi chính xác từng bit dữ liệu. Những kẻ tấn công còn có thể sử dụng một số công cụ để dò tìm lỗi dạng này và tấn công với những thông tin đã được thiết lập sẵn.

### Thay đổi giá trị điều kiện truy vấn

Dạng lỗi này khiến cho kẻ tấn công có thể thay đổi giá trị điều kiện trong câu truy vấn, làm sai lệch sự hiển thị của một ứng dụng chứa lỗi này.

|  |
| --- |
| **SELECT** booktitle **FROM** booklist **WHERE** bookId = 'OOk14cd' **AND** 1=1; |

Sẽ hiển thị một trang một cách bình thường, trong khi:

|  |
| --- |
| **SELECT** booktitle **FROM** booklist **WHERE** bookId = 'OOk14cd' **AND** 1=2 |

sẽ hiển thị một nội dung khác, hoặc không hiển thị gì nếu ứng dụng web có chứa lỗi SQL injection dạng này. Lỗ hổng dạng này còn cho phép tin tặc không chỉ gây ảnh hưởng tới bảng hay dữ liệu hiện tại mà còn ảnh hưởng tới những dữ liệu hay bảng khác phụ thuộc vào nội dung của dữ liệu hay bảng hiện tại.

### Điều kiện lỗi

Lỗi SQL injection dạng này dẫn tới việc buộc cơ sở dữ liệu chỉ được phép đánh giá khi mà giá trị của câu lệnh WHERE là đúng. Ví dụ

|  |
| --- |
| **SELECT** 1/0 **FROM** users **WHERE** username='Ralph'; |

Phép chia cho 0 chỉ được đánh giá là lỗi khi mà người dùng có tên "Ralph" tồn tại trong cơ sở dữ liệu.

### Thời gian trễ

Lỗi SQL injection dạng này tồn tại khi thời gian xử lý của một hay nhiều truy vấn SQL phụ thuộc vào dữ liệu logic được nhập vào hoặc quá trình xử lý truy vấn của SQL engine cần nhiều thời gian. Tin tặc có thể sử dụng lỗi SQL injection dạng này để xác định thời gian chính xác mà trang cần tải khi giá trị nhập vào là đúng

# Một số dạng tấn công thường gặp với các ứng dụng web

## Dạng tấn công vượt qua kiểm tra lúc đăng nhập

Với dạng tấn công này, tin tặc có thể dễ dàng vượt qua các trang đăng nhập nhờ vào lỗi khi dùng các câu lệnh SQL thao tác trên cơ sở dữ liệu của ứng dụng web. Thông thường để cho phép người dùng truy cập vào các trang web được bảo mật, hệ thống thường xây dựng trang đăng nhập để yêu cầu người dùng nhập thông tin về tên đăng nhập và mật khẩu. Sau khi người dùng nhập thông tin vào, hệ thống sẽ kiểm tra tên đăng nhập và mật khẩu có hợp lệ hay không để quyết định cho phép hay từ chối thực hiện tiếp. Ví dụ, trong trường hợp sử dụng ASP, người ta có thể dùng 2 trang: 1 trang HTML để hiển thị Form nhập liệu và 1 trang ASP để xử lý thông tin nhập vào từ phía người dùng như sau:

|  |
| --- |
| - Trang nhập liệu: login.htm |
| <form action="ExecLogin.asp" method="post">  Username: <input type="text" name="fUSRNAME"><br />  Password: <input type="password" name="fPASSWORD"><br />  <input type="submit">  </form> |
| - Trang xử lý nhập liệu: execlogin.asp |
| <%  Dim vUsrName, vPassword, objRS, strSQL  vUsrName = Request.Form("fUSRNAME")  vPassword = Request.Form("fPASSWORD")  strSQL = "SELECT \* FROM T\_USERS " & \_  "WHERE USR\_NAME=' " & vUsrName & \_  " ' and USR\_PASSWORD=' " & vPassword & " ' "  **Set** objRS = Server.CreateObject("ADODB.Recordset")  objRS.Open strSQL, "DSN=..."  **If** (objRS.EOF) Then  Response.Write "Invalid login."  **Else**  Response.Write "You are logged in as " & objRS("USR\_NAME")  End **If**  **Set** objRS = Nothing %> |

Chỗ sơ hở trong đoạn mã xử lý nhập liệu trên nằm ở chỗ dữ liệu nhập vào từ người dùng được dùng để xây dựng trực tiếp câu lệnh SQL. Chính điều này cho phép tin tặc có thể điều khiển câu truy vấn sẽ được thực hiện. Ví dụ, nếu người dùng nhập chuỗi trong ngoặc sau vào trong cả 2 ô nhập liệu username/password của trang login.htm là:('OR*='). Lúc này, câu truy vấn sẽ được gọi thực hiện là:*

|  |
| --- |
| **SELECT** \* **FROM** T\_USERS **WHERE** USR\_NAME =''**OR**''='' **AND** USR\_PASSWORD= ''**OR**''='' |

Câu truy vấn này là hợp lệ và sẽ trả về tất cả các bản ghi của T\_USERS và đoạn mã tiếp theo xử lí người dùng đăng nhập bất hợp pháp này như là người dùng đăng nhập hợp lệ.

## Dạng tấn công sử dụng câu lệnh SELECT

Dạng tấn công này phức tạp hơn. Để thực hiện được kiểu tấn công này, kẻ tấn công phải có khả năng hiểu và lợi dụng các sơ hở trong các thông báo lỗi từ hệ thống để dò tìm các điểm yếu khởi đầu cho việc tấn công. Ví dụ, trong các trang tìm kiếm. Các trang này cho phép người dùng nhập vào các thông tin tìm kiếm như Họ, Tên, … Đoạn mã thường gặp là:

|  |
| --- |
| <%  Dim vAuthorName, objRS, strSQL  vAuthorName = Request("fAUTHOR\_NAME")  strSQL = "SELECT \* FROM T\_AUTHORS WHERE AUTHOR\_NAME =' " & \_ vAuthorName & " ' "  Set objRS = Server.CreateObject("ADODB.Recordset")  objRS.Open strSQL, "DSN=..."  …  Set objRS = Nothing %> |

Tương tự như trên, tin tặc có thể lợi dụng sơ hở trong câu truy vấn SQL để nhập vào trường tên tác giả bằng chuỗi giá trị:

|  |
| --- |
| ' UNION SELECT ALL SELECT OtherField FROM OtherTable WHERE ' '=' (\*) |

Lúc này, ngoài câu truy vấn đầu không thành công, chương trình sẽ thực hiện thêm lệnh tiếp theo sau từ khóa UNION nữa. Giả sử đoạn mã nhập vào là:

|  |
| --- |
| ' DROP TABLE T\_AUTHORS -- |

Câu truy vấn sẽ thực hiện việc xóa bảng.

## Dạng tấn công sử dụng câu lệnh INSERT

Thông thường các ứng dụng web cho phép người dùng đăng kí một tài khoản để tham gia. Chức năng không thể thiếu là sau khi đăng kí thành công, người dùng có thể xem và hiệu chỉnh thông tin của mình. SQL injection có thể được dùng khi hệ thống không kiểm tra tính hợp lệ của thông tin nhập vào. Ví dụ, một câu lệnh INSERT có thể có cú pháp dạng:

|  |
| --- |
| INSERT INTO TableName VALUES('Value One', 'Value Two', 'Value Three') |

Nếu đoạn mã xây dựng câu lệnh SQL có dạng:

|  |
| --- |
| <%  strSQL = "INSERT INTO TableName VALUES(' " & strValueOne & " ', ' " \_ & strValueTwo & " ', ' " & strValueThree & " ') "  Set objRS = Server.CreateObject("ADODB.Recordset")  objRS.Open strSQL, "DSN=..."  …  Set objRS = Nothing %> |

Thì chắc chắn sẽ bị lỗi SQLi, bởi vì nếu ta nhập vào trường thứ nhất ví dụ như:

|  |
| --- |
| ' + (SELECT TOP 1 FieldName FROM TableName) + ' |

Lúc này câu truy vấn sẽ là:

|  |
| --- |
| INSERT INTO TableName VALUES(' ' + (SELECT TOP 1 FieldName FROM TableName) + ' ', 'abc', 'def') |

Khi đó, lúc thực hiện lệnh xem thông tin, xem như bạn đã yêu cầu thực hiện thêm một lệnh nữa đó là:

|  |
| --- |
| SELECT TOP 1 FieldName FROM TableName |

## Dạng tấn công sử dụng stored-procedures

Việc tấn công bằng stored-procedures sẽ gây tác hại rất lớn nếu ứng dụng được thực thi với quyền quản trị hệ thống 'sa'. Ví dụ, nếu ta thay đoạn mã tiêm vào dạng: ' ; EXEC xp\_cmdshell ‘cmdd.exe dir C: '. Lúc này hệ thống sẽ thực hiện lệnh liệt kê thư mục trên ổ đĩa C:\ cài đặt server. Việc phá hoại kiểu nào tuỳ thuộc vào câu lệnh đằng sau cmd.exe.

# Phòng chống SQL injection

Như vậy, có thểthấy lỗi SQL injection khai thác những bất cẩn của các lập trình viên phát triển ứng dụng web khi xử lí các dữ liệu nhập vào để xây dựng câu lệnh SQL. Tác hại từ lỗi SQL injection tùy thuộc vào môi trường và cách cấu hình hệ thống. Nếu ứng dụng sử dụng quyền dbo (quyền của người sởhữu cơ sở dữ liệu - owner) khi thao tác dữ liệu, nó có thể xóa toàn bộ các bảng dữ liệu, tạo các bảng dữ liệu mới, … Nếu ứng dụng sử dụng quyền sa (quyền quản trị hệ thống), nó có thể điều khiển toàn bộ hệ quản trị cơ sở dữ liệu và với quyền hạn rộng lớn như vậy nó có thể tạo ra các tài khoản người dùng bất hợp pháp để điều khiển hệ thống của bạn. Để phòng tránh, ta có thể thực hiện:

## Kiểm soát chặt chẽ dữ liệu nhập vào

Để phòng tránh các nguy cơ có thể xảy ra, hãy bảo vệ các câu lệnh SQL là bằng cách kiểm soát chặt

chẽ tất cả các dữ liệu nhập nhận được từ đối tượng Request (Request, Request.QueryString,

Request.Form, Request.Cookies, and Request.ServerVariables). Ví dụ, có thể giới hạn chiều dài của

chuỗi nhập liệu, hoặc xây dựng hàm EscapeQuotes để thay thế các dấu nháy đơn bằng 2 dấu nháy

đơn như:

|  |
| --- |
| <%  Function EscapeQuotes(sInput)  sInput = replace(sInput, " ' ", " ' ' ")  EscapeQuotes = sInput  5  End Function  %> |

Trong trường hợp dữ liệu nhập vào là số, lỗi xuất phát từ việc thay thế một giá trị được tiên đoán là dữ liệu số bằng chuỗi chứa câu lệnh SQL bất hợp pháp. Để tránh điều này, đơn giản hãy kiểm tra dữ liệu có đúng kiểu hay không bằng hàm IsNumeric(). Ngoài ra có thể xây dựng hàm loại bỏ một số kí tự và từ khóa nguy hiểm như: ;, --, select, insert,

xp\_, … ra khỏi chuỗi dữliệu nhập từphía người dùng đểhạn chếcác tấn công dạng này:

|  |
| --- |
| <%  Function KillChars(sInput)  dim badChars  dim newChars  badChars = array("select", "drop", ";", "--", "insert", "delete", "xp\_")  newChars = strInput  for i = 0 to uBound(badChars)  newChars = replace(newChars, badChars(i), "")  next  KillChars = newChars  End Function  %> |

## Thiết lập cấu hình an toàn cho hệ quản trị cơ sở dữ liệu

Cần có cơ chế kiểm soát chặt chẽ và giới hạn quyền xử lí dữ liệu đến tài khoản người dùng mà ứng dụng web đang sử dụng. Các ứng dụng thông thường nên tránh dùng đến các quyền như dbo hay sa. Quyền càng bị hạn chế, thiệt hại càng ít.

Ngoài ra để tránh các nguy cơ từ SQL Injection attack, nên chú ý loại bỏ bất kì thông tin kĩ thuật nào chứa trong thông điệp chuyển xuống cho người dùng khi ứng dụng có lỗi. Các thông báo lỗi thông thường tiết lộ các chi tiết kĩ thuật có thể cho phép kẻ tấn công biết được điểm yếu của hệ thống.

## Phòng chống từ mức xây dụng mã nguồn ứng dụng

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

### Làm sạch dữ liệu đầu vào

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

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

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

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

* Username: chỉ chứa các ký tự chữ cái, chữ số và dấu gạch dưới, độ dài tối đa 30 ký tự, tối thiểu 3 ký tự: “^([a-zA-Z0-9] |\_){3,30}$”
* Password: chỉ chứa ký tự chữ cái, chữ số, dấu gạch dưới, độ dài tối thiểu 4, tối đa 50: “^([a-zA-Z0-9] |\_) {4,50}$”
* Email: chỉ chứa ký tự chữ cái, chữ số, dấu gạch dưới, dấu chấm và kí tự @ trong teenm sẽ có dạng như sau: “( |^) [a-zA-Z0-9] |\_)\*@([a-z0-9]+.){1, }[a-z]+( |$)”

#### Mô hình danh sách cấm – blacklist

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

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

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

“ ‘ | % | -- | ; | / \ \* | \ \ \ \* | \_ | \ [ | @ | xp\_”

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

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

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

##### Trong PHP

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

* + *is\_<type>(input)*: type được thay bằng kiểu dữ liệu muốn kiểm tra, ví dụ *is\_numeric($\_GET[‘price’])*; hàm này kiểm tra kiểu dữ liệu và trả về true/ false.
  + *Strlen(input)*: trả về độ dài input. Ví dụ *strlen($keyword\_search);*

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

##### Trong C#

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

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

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

##### Trong Java

Thực hiện cài đặt từ giao tiếp javax.faces.validator.Validator. Giao tiếp này nằm trong framework có tên là Java Server Faces (JSF).

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

#### Khái niệm

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

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

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

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

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

|  |
| --- |
| SELECT \* FROM ? WHERE username = ‘nam’  SELECT ? FROM students WHERE studentid = 21  SELECT \* FROM students WHERE address LIKE ‘Hanoi%’ ORDER BY ? |

Trong đó các dấu ? là các biến giữ chỗ (placeholder), tùy vào từng database, biến giữ chỗ sẽ khác nhau.

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

String sql = “SELECT \* FROM” + tbl\_Name = “WHERE column\_Name = ?”

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

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

#### Tham số hóa truy vấn trong PHP

Một prepared query thường có dạng như sau:

SELECT \* FROM tbl\_Name WHERE col\_Name =?

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

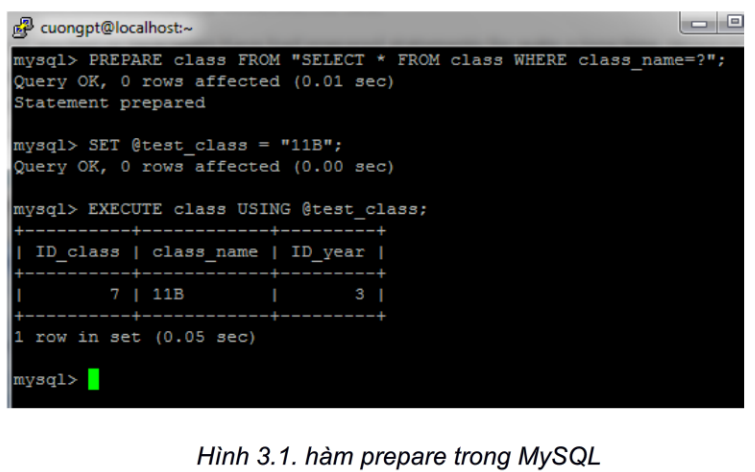
Bản thân MySQL cũng hỗ trợ hàm PREPARE để sinh các truy vấn tham số hóa. Ví dụ với truy vấn đơn giản sau:

PREPARE class FROM “SELECT \* FROM class WHERE class\_name =?”;

Khi thực thi:

SET @test\_class = “11B”;

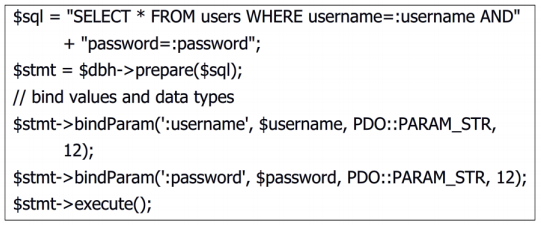
EXECUTE class USING @test\_class;



Xét trường hợp PHP sử dụng sqli để kết nối tới MySQL, ta có thể sử dụng cả 2 hình thức tham số hóa (kiểu hướng đối tượng và kiểu thủ tục) như sau:

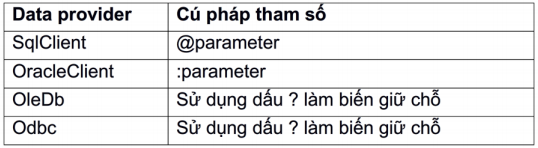
|  |
| --- |
| /\*-----------------------======---------------------------------=====-------------------  Source from: <http://www.php.net/manual/en/mysqli.prepare.php>  OPP-style  /\*----------------------=======--------------------------------=====-------------------  <?php  $mysqli = new mysqli(“localhost”,”my\_user”,”my\_password”,”world”);  …  $city = “Amersfoort”;  /\* create a prepared statement \*/  if($stmt = $mysqli->prepare(“SELECT District FROM City WHERE Name = ?”)){  /\*blind parameters for markers \*/  $stmt->bind\_param(“s”,$city);  /\* execute query \*/  $stmt->execute();  /\* bind result variables \*/  $stmt->bind\_result($district);  /\* fetch value \*/  $stmt->fetch();  Printf(“%s is in district %\n”, $city, $district);  /\* close statement \*/  $stmt->close();  }  /\* close connection \*/  $mysqli->close();  ?> |
| /\*-----------======---------------------=======--------------------------------  Procedural style \*/  /\*----------======---------------------=======---------------------------------  <?php  $link = mysqli\_connect(“localhost”, “my\_user”, “my\_password”, “world”);  …  $city = “Amersfoort”;  /\* create a prepared statement \*/  if($stmt = mysqli\_prepare($link, “SELECT District FROM City WHERE Name = ?”)){  /\* bind parameters for markers \*/  mysqli\_stmt\_bind\_param($stmt, “s”, $city);  /\* execute query \*/  mysqli\_stmt\_execute($stmt);  /\* bind result variables \*/  mysqli\_stmt\_bind\_result($stmt, $district);  /\* fetch value \*/  mysqli\_stmt\_fetch($stmt);  printf(“%s is district %s\n”, $city, $district);  /\* close statement \*/  mysqli\_stmt\_close($stmt);  }  /\* close connection \*/  mysqli\_close($link);  ?> |

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



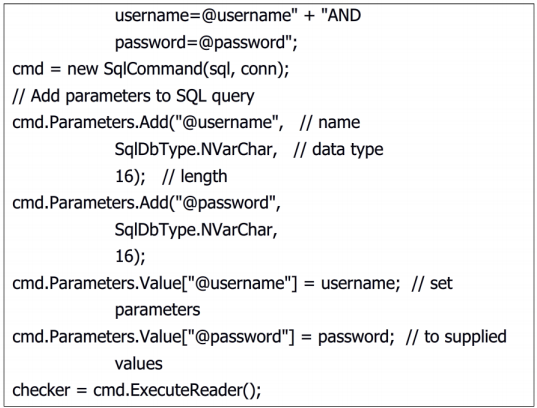
#### Tham số hóa truy vấn trong C#

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

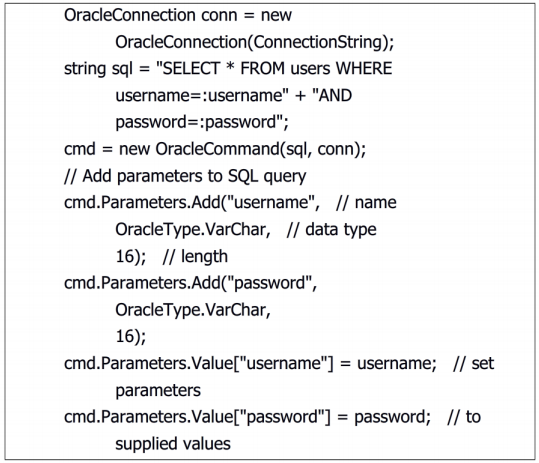


Bảng 6.1 Cú pháp đại diện tham số trong truy vấn trong C#

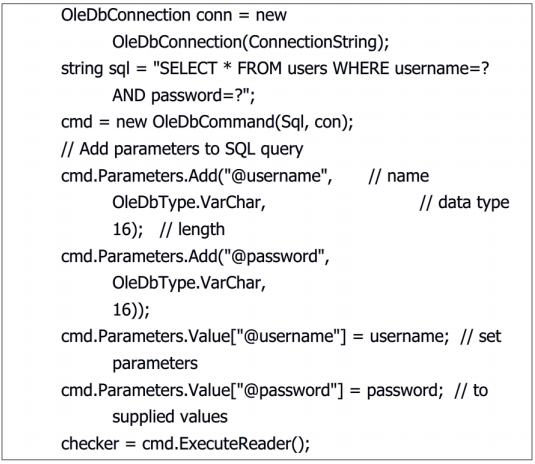
Xét đoạn mã sau xây dựng truy vấn tham số hóa trên provider là SqlClient



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



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

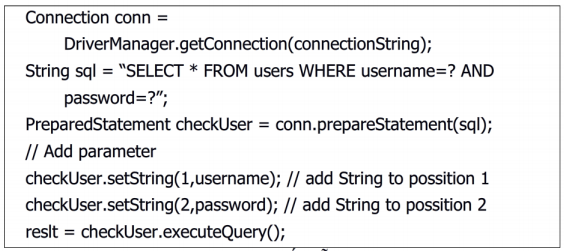


Trong framework ADO.NET, chúng ta có thể chỉ định nhiều thông tin hơn về tham số, càng chi tiết thì việc tối ưu và kiểm tra tham số sẽ chi tiết hơn. Để đảm bảo an ninh, tối thiểu cần chỉ định thêm thông số về kích thước dữ liệu và kiểu dữ liệu cho tham số.

#### Tham số hóa truy vấn trong Java

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

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



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

