TẤN CÔNG KIỂU SQL INJECTION - TÁC HẠI VÀ PHÒNG TRÁNH

Lê Đình Duy

Khoa Công Nghệ Thông Tin, Trường ĐH Khoa Học Tự Nhiên Tp. HCM.

Email: ldduy@fit.hcmuns.edu.vn

1. SQL Injection là gì?

Khi triển khai các ứng dụng web trên Internet, nhiều người vẫn nghĩ rằng việc đảm bảo an toàn, bảo mật nhằm giảm thiểu tối đa khả năng bị tấn công từ các tin tặc chỉ đơn thuần tập trung vào các vấn đề như chọn hệ điều hành, hệ quản trị cơ sở dữ liệu, webserver sẽ chạy ứng dụng, ... mà quên mất rằng ngay cả bản thân ứng dụng chạy trên đó cũng tiềm ẩn một lỗ hổng bảo mật rất lớn. Một trong số các lỗ hổng này đó là SQL injection. Tại Việt Nam, đã qua thời kì các quản trị website lơ là việc quét virus, cập nhật các bản vá lỗi từ các phần mềm hệ thống, nhưng việc chăm sóc các lỗi của các ứng dụng lại rất ít được quan tâm. Đó là lí do tại sao trong thời gian vừa qua, không ít website tại Việt Nam bị tấn công và đa số đều là lỗi SQL injection [1]. Vậy SQL injection là gì ?

SQL injection là một kĩ thuật cho phép những kẻ tấn công lợi dụng lỗ hổng trong việc kiểm tra dữ liệu nhập trong các ứng dụng web và các thông báo lỗi của hệ quản trị cơ sở dữ liệu để "tiêm vào" (inject) và thi hành các câu lệnh SQL bất hợp pháp (không được người phát triển ứng dụng lường trước). Hậu quả của nó rất tai hại vì nó cho phép những kẻ tấn công có thể thực hiện các thao tác xóa, hiệu chỉnh, … do có toàn quyền trên cơ sở dữ liệu của ứng dụng, thậm chí là server mà ứng dụng đó đang chạy. Lỗi này thường xảy ra trên các ứng dụng web có dữ liệu được quản lí bằng các hệ quản trị cơ sở dữ liệu như SQL Server, MySQL, Oracle, DB2, Sysbase.

2. Các dạng tấn công bằng SQL Injection

Có bốn dạng thông thường bao gồm: vượt qua kiểm tra lúc đăng nhập (authorization bypass), sử dụng câu lện SELECT, sử dụng câu lệnh INSERT, sử dụng các stored-procedures [2], [3].

2.1. Dạng tấn công vượt qua kiểm tra 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.

Xét một ví dụ điển hình, 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.

Trong trường hợp này, người ta có thể dùng hai trang, một trang HTML để hiển thị form nhập liệu và một trang ASP dùng để xử lí thông tin nhập từ phía người dùng. Ví dụ:

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>
```

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
%>
```

Thoạt nhìn, đoạn mã trong trang execlogin.asp dường như không chứa bất cứ một lỗ hổng về an toàn nào. Người dùng không thể đăng nhập mà không có tên đăng nhập và mật khẩu hợp lệ. Tuy nhiên, đoạn mã này thực sự không an toàn và là tiền đề cho một lỗi SQL injection. Đặc biệt, chỗ sơ hở 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 những kẻ tấn công có thể điều khiển câu truy vấn sẽ được thực hiện. Ví dụ, nếu người dùng nhập chuỗi 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.

2.2. 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.

Xét một ví dụ rất thường gặp trong các website về tin tức. Thông thường, sẽ có một trang nhận ID của tin cần hiển thị rồi sau đó truy vấn nội dung của tin có ID này. Ví dụ: http://www.myhost.com/shownews.asp?ID=123. Mã nguồn cho chức năng này thường được viết khá đơn giản theo dạng

```
<%
  Dim vNewsID, objRS, strSQL
  vNewsID = Request("ID")

  strSQL = "SELECT * FROM T_NEWS WHERE NEWS_ID =" & vNewsID
```

```
  Set objRS = Server.CreateObject("ADODB.Recordset")
  objRS.Open strSQL, "DSN=..."

  Set objRS = Nothing
%>
```

Trong các tình huống thông thường, đoạn mã này hiển thị nội dung của tin có ID trùng với ID đã chỉ định và hầu như không thấy có lỗi. Tuy nhiên, giống như ví dụ đăng nhập ở trước, đoạn mã này để lộ sơ hở cho một lỗi SQL injection khác. Kẻ tấn công có thể thay thế một ID hợp lệ bằng cách gán ID cho một giá trị khác, và từ đó, khởi đầu cho một cuộc tấn công bất hợp pháp, ví dụ như: **0 OR 1=1** (nghĩa là, http://www.myhost.com/shownews.asp?ID=0 or 1=1).

Câu truy vấn SQL lúc này sẽ trả về tất cả các article từ bảng dữ liệu vì nó sẽ thực hiện câu lệnh:
SELECT * FROM T_NEWS WHERE NEWS_ID=0 or 1=1

Một trường hợp khác, ví dụ như trang tìm kiếm. 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.

Tất nhiên các ví dụ nói trên, dường như không có gì nguy hiểm, nhưng hãy thử tưởng tượng kẻ tấn công có thể xóa toàn bộ cơ sở dữ liệu bằng cách chèn vào các đoạn lệnh nguy hiểm như lệnh DROP TABLE. Ví dụ như: **' DROP TABLE T_AUTHORS --**

Chắc các bạn sẽ thắc mắc là làm sao biết được ứng dụng web bị lỗi dạng này được. Rất đơn giản, hãy nhập vào chuỗi *(*)* như trên, nếu hệ thống báo lỗi về cú pháp dạng: Invalid object name "OtherTable"; ta có thể biết chắc là hệ thống đã thực hiện câu SELECT sau từ khóa UNION, vì như vậy mới có thể trả về lỗi mà ta đã cố tình tạo ra trong câu lệnh SELECT.

Cũng sẽ có thắc mắc là làm thế nào có thể biết được tên của các bảng dữ liệu mà thực hiện các thao tác phá hoại khi ứng dụng web bị lỗi SQL injection. Cũng rất đơn giản, bởi vì trong SQL Server, có hai đối tượng là sysobjects và syscolumns cho phép liệt kê tất cả các tên bảng và cột có trong hệ thống. Ta chỉ cần chỉnh lại câu lệnh SELECT, ví dụ như:

' UNION SELECT name FROM sysobjects WHERE xtype = 'U' là có thể liệt kê được tên tất cả các bảng dữ liệu.

2.3. 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 SQL injection, 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**

2.4. 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 'cmd.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.

3. Cách phòng tránh

Như vậy, có thể thấy 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 sở hữu 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 ở hai mức:

3.1. 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
```

```
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 dữ liệu nhập từ phía người dùng để hạn chế các 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
%>
```

3.2. 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.

Tham chiếu

[1]. Danh sách các website bị lỗi SQL injection: http://www.security.com.vn/

[2]. SQL Injection FAQ: http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=2&tabid=3

[3]. Advanced SQL Injection : http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[4]. Preventing SQL Injection: http://www.owasp.org/asac/input_validation/sql.shtml

[5]. SQL Injection Attacks - Are You Safe? http://www.sitepoint.com/article/794