SQL Injection in INSERT Query

AMol NAik
http://amolnaik4.blogspot.com
http://twitter.com/amolnaik4

SQL injection is being one of the mostly exploited issues in web application security and has found a place in OWASP Top 10 since 2004. There are many blog posts, papers available on SELECT query injection exploiting WHERE or HAVING clauses. Today I'm going to discuss SQL injection in INSERT query.

The Basics:
INSERT query followed by VALUES inserts rows into an existing table based on explicitly specified values. The syntax of INSERT query is: (source: http://dev.mysql.com/doc/refman/5.5/en/insert.html)

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

`tbl_name` is the table into which rows should be inserted. A comma-separated list of column names can be provided following the table name. In this case, a value for each named column must be provided by the `VALUES` list.

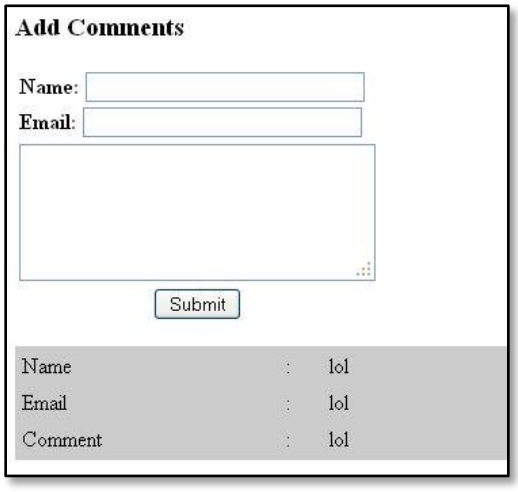
To insert a record in a table, following query will be used:

```
INSERT INTO tbl_name (a,b,c) VALUES('data','data','data');
```

I hope this is enough to introduce INSERT query.

The Injection:

One of the examples of usage of INSERT query in web application is comment page.



The page requests for name, email address and the comment and inserts this data into database using following query:

```
INSERT INTO comments (name, email, comment) VALUES ('lol','lol','lol');
```

In this query, an attacker can inject arbitrary data if the inputs are not sanitized. Let's check this by placing single quote (') in name field.

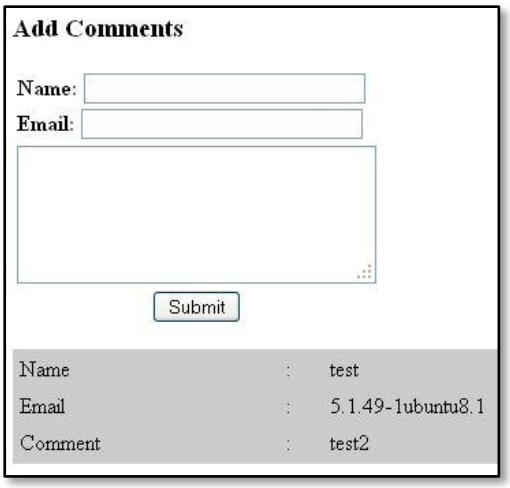
This results in SQL Error as expected:

```
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 'asd', 'asd')' at line 1
```

Now we can inject any data in all fields with comment string at the end.

To get the result of injected query, we need a place where the injected data is reflected back by the application. In this example, the comment details are printed back on the page.

So let's start injecting something which will give information about the database server. We can insert sql subquery in place of parameter value. We will insert subquery '**(select version())**' without single quotes in 'email' parameter.



We get the mysql version ☺. In this way we can get the other database details as well like current user, current database, etc.

Let try to get the password of user 'root' from mysql.user table.

Injected Data:

```
test',(select password from mysql.user where user='root'),'test2')-- -
```

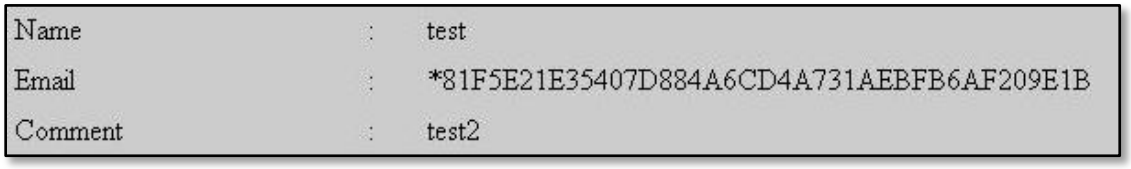
This gives error:

```
Subquery returns more than 1 row
```

Hmm, only one row. We will use LIMIT to fetch 1 row at a time. Let's craft the payload with LIMIT:

```
test',(select password from mysql.user where user='root' limit 0,1),'test2')-- -
```

This works and we now have password hash for user 'root':

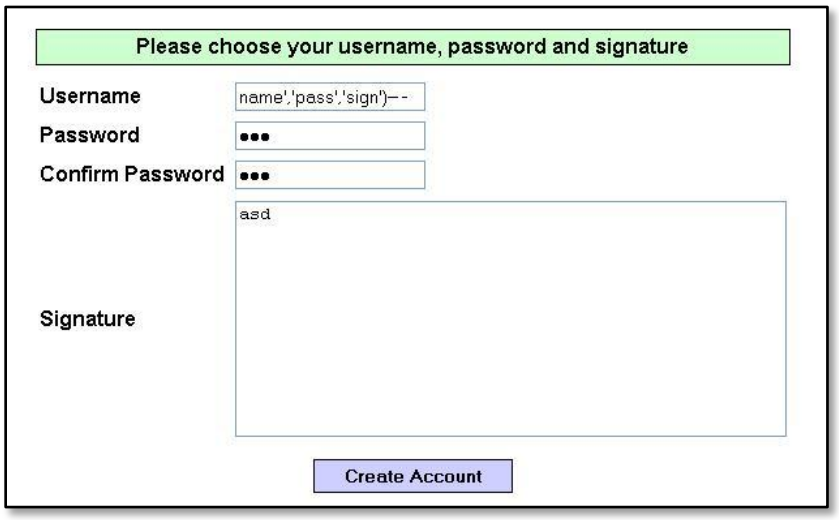


In this way, we can mine the database with SQL injection in INSERT Query using sql subquery.

The INSERT query is used by register user pages as well in web applications. Let's analyze an example. This example is taken from "Mutillidae", a well-known web application to learn security. A new user provides details such as username, password & signature in order to create an account with the forum/application. This data is then inserted into database using this query. The background SQL query looks like this:

```
INSERT INTO accounts (username, password, mysignature) VALUES ('data', 'data', 'data');
```

It's same as the last example and we will be able to inject arbitrary values into database using single quote (') and comment string (-- -):



And the result is:

Great!! The user is added. Now we can use the same sql subquery technique to inject sql queries and to get the data. But the question is where will be the returned data?

There are 2 places where the injected data is being reflected by the app.

One place is "Account created" message as shown in above snapshot. Here 'username' value is being reflected. As we cannot control the first single quote (') for 'username' field, we will not be able to inject subquery which will successfully give us the returned data.

We need to look for other places to see if our injected data is being reflected back. As we have registered an account, let's login and check reflected data.

We can see 'signature' parameter is getting reflected in status message. So we need to inject subqueries into 'signature' parameter and we will get returned data in status message.

Let's start with mysql version(). The payload in username field for this will be:

```
test','test',(select version()))-- -
```

And user added successfully.

Now login with username & password as 'test' and check status. It should have mysql version info.

Yes, it's there and the injection is successful. Let's try to get password hash for user 'root'.

Payload:

```
test1','test1',(select password from mysql.user where user='root'))-- -
```

And Error:

We again need to use LIMIT to get only one row.

Payload:

```
test1','test1',(select password from mysql.user where user='root' LIMIT 0,1))-- -
```

And we get password hash for user ‘root’.

That’s all. The same way other data can be mined from the database.

Conclusion:
1. Identify the injection point.
2. Check where the injected data is visible.
3. Use subquery to insert sql queries.
4. Use LIMIT to get one row at a time.

Hope you like this post. Suggestions, queries are welcome.