SQL Injection to Help You Sleep at Night

If there's anything to be learned from Gitlab's recent downtime (which they handled amazingly well), it's that production databases need to be pampered. They aren't something to play around with and as penetration testers that responsibility extends to us.

Many companies will allow testing in production, it can be argued that it is the responsible thing to do; production is where a company is most likely to get hit and it's important to test those servers. While everything said in this blog should be followed in non-prod environments, it isn't a catastrophe if non-prod data is modified. As a penetration tester it starts becoming catastrophic when one mistake in production can lead to outages and having to restore from backups, if they even exist. There has to be a way to test SQL Injection without the risk of modifying production data accidentally.

A google search for "Safe SQL Injection" will return 0 relevant results. Surely others have written on this topic and other NetSPI employees have mentioned how they go about this, but the goal of this blog is to make this subject visible and easily accessible.

Setup

Starting with setting up the databases, 3 popular Relational Database Management Systems and their associated syntaxes will be used.

RDBMS	Create Table

MySQL 5.7.12	`CREATE TABLE USERS (`
	`username VARCHAR(100)  NOT NULL,`
	`password VARCHAR(100) NOT NULL,`
	`email VARCHAR(100)    NOT NULL`
	`)`
	`;`

MSSQL Server 2014 Express Edition	`CREATE TABLE USERS`
	`(username varchar(100),`
	`password varchar(100),`
	`email varchar(100))`
	`;`

Oracle SQL 12c	`CREATE TABLE USERS`
	`("username" VARCHAR2(100),`
	`"password" VARCHAR2(100),`
	`"email" VARCHAR2(100)`
	`)`
	`/`

Go ahead and add some users as well.

RDBMS	Add Users
MySQL 5.7.12	`INSERT INTO USERS (username, password, email) values ('jake','reynolds','jreynoldsdev@gmail.com'), ('net','spi','alex@netspi.com'), ('johnjacob','jingle','heimer@schmidt.com');`
MSSQL Server 2014 Express Edition	`INSERT INTO USERS (username, password, email) VALUES ('jake','reynolds','jreynoldsdev@gmail.com'), ('net','spi','alex@netspi.com'), ('johnjacob','jingle','heimer@schmidt.com');`
Oracle SQL 12c	`INSERT into USERS ("username", "password", "email") values ('jake','reynolds','jreynoldsdev@gmail.com'), ('net','spi','alex@netspi.com'), ('johnjacob','jingle','heimer@schmidt.com') /`

Pen Tester's First Day at Work

Now every database has a table called USERS with the structure:

	username	**password**	**email**
1	jake	reynolds	jreynoldsdev@gmail.com
2	net	spi	alex@netspi.com
3	johnjacob	jingle	heimer@schmidt.com

This is usually the first table any pen tester would test against since it is called from every login form. A simple query is used here:

```
SELECT username FROM USERS WHERE username='$username' and password='$password';
```

There's not much harm to this query, aside from being vulnerable to SQLi. As a tester tossing in a ' or 1=1 — here or there won't hurt anybody. How about the next time this table comes into play? When a user wants to update their email address the query looks somewhat like:

```
UPDATE USERS set email='$email' where username='$username';
```

Now here's a weekend ruiner if the test is in production. Giving this input form the simple test of '; — can ruin the entire Users table.

```
UPDATE USERS set email=''; -- where username = '$username';
```

	username	password	email
1	jake	reynolds	
2	net	spi	
3	johnjacob	jingle	

CRAP.

Every email in the company's database has been deleted. Maybe they have backups, but it's not St. Patrick's Day so luck is a little short. What happens now? Dust off that resume and hope to not make the same mistake with future employers.

How to Keep Future Jobs

There are a couple ways to avoid this mistake and they come down to taking an extra second to think about the query format before inserting injection strings. Going back to the update query, look at it from another angle.

```
UPDATE USERS set email='$email' where username='$username';
```

This would be blind to testers, but it would be behind a request similar to:

```
POST /updateEmail HTTP/1.1
Host: jakereynolds.co
Connection: close
Content-Length: 165
Content-Type: application/x-www-form-urlencoded

username=jake&email=jreynoldsdev@gmail.com
```

It's clear that an email parameter is going to be inserted into a query. Our goal is to find some strings that can be inserted without ruining everyone's weekend.

The first attempt is string concatenation, breaking out of the query and appending something to our string. This allows the rest of the query to still be valid and shows if the parameter is vulnerable.

MSSQL	MySQL	Oracle
`'+'concat`	`con' 'cat'`	`'||'concat`

These strings all result in the query looking similar to:

```
UPDATE USERS set email=''+'concat' where username='jake';
```

	username	password	email

1	jake	reynolds	concat
2	net	spi	alex@netspi.com
3	johnjacob	jingle	heimer@schmidt.com

Now everyone is hunky-dory, but none of the queries are the same across the 3 RDBMS'. What other options are available for these 3? MySQL and Oracle allow arithmetic operators on numeric strings. If the injection does not need to escape a quote, MSSQL can be used as well with integers.

MSSQL	**MySQL**	**Oracle**

`1+1`
`1-1`
`1/1`
`1*1`
	`'='test`
	`1'+'1`
	`1'-'1`
	`1'/'1`
	`1'*'1`
		`1'+'1`
		`1'-'1`
		`1'/'1`
		`1'*'1`

Using addition from MySQL shows this is possible with strings and numbers.

```
UPDATE USERS set email='1'+'1' where username='jake';
```

	username	**password**	**email**
1	jake	reynolds	2
2	net	spi	alex@netspi.com
3	johnjacob	jingle	heimer@schmidt.com

So all 3 of the RDBMS' have some options to use, but this is operating under the assumption that it doesn't matter what database is being tested. What option is there to safely inject a string blindly into any of these 3 databases?

The Blind Leading the Blind

It was difficult to find any operators, functions, etc… that executed in the same way across all 3 databases. Although, coming up from behind for a cheap 2nd is one operator that works on all 3, just doing different things.

In MSSQL the + character acts as a form of string concatenation, as presented above. MySQL and Oracle initially failed any tests for this operator until it came clear that they are for integer arithmetic. That gives the magical injection string of:

MSSQL	**MySQL**	**Oracle**
`1'+'1`	`1'+'1`	`1'+'1`

```
UPDATE USERS set email='1'+'1' where username='jake';
```

	username	**password**	**email**
1	jake	reynolds	2
2	net	spi	alex@netspi.com
3	johnjacob	jingle	heimer@schmidt.com

In MSSQL the output will become 11, due to string concatenation.

There it is! We now have an option that will allow us to inject blindly into queries for 3 major RDBMS', without potentially destroying their tables. The challenge going forward is expanding this to fit more RDBMS' and to fit more complicated scenarios. That will be left as a challenge to the user, but if you have any other ideas or comments please let us know below!