Basic SQL Timing Attacks
IN618 Security

Introduction

In the last lab we carried out some SQL injection attacks on a target application. We probed the remote database to learn table and column names, using the fact that the web site would show an error message if we requested a table or column that did not exist. But, to be honest, I wrote that application with the specific goal that it would be exploitable with SQL injection, including making sure that it gave helpful error messages. Some developers do that anyway, and attackers thank them for it. However, when I write real applications I generally avoid displaying wrrors that help attackers - and that usually don't help legitmate users anyway. That's what logs are for.

Suppose we wanted to exploit SQL injection, but we haven't been given the benefit of helpful error messages. We need a way to write queries that repond in an observably different way. It turns out that we can do this using an *SQL timing attack*. The idea is tht we write queries that return quickly on failure, but slowly on success (or vise-versa).

1 The exploit

We need an SQL statement that takes a long time to execute. The statement to use depends on what sort of database we are querying. For example, MySQL has a handy `SLEEP()` function that sleeps for the specified number of seconds. So first, we need to see if our target system is running MySQL.

Last time we injected queries like this one:

```
SELECT * FROM user_accounts WHERE email_addr = 'x'; SELECT COUNT(*) FROM possible_table; -'
```

Where the red text is what we entered into the HTML form and the black part is our estimation of the application's SQL code into which we are injecting our code. We will test if the `SLEEP()` function works by using it in place of our `SELECT COUNT...` query:

```
SELECT * FROM user_accounts WHERE email_addr = 'x'; SELECT SLEEP(20); -'
```

If the `SLEEP()` call works, then it will take over 20 seconds to get a response back. This means that we are almost certainly dealing with a MySQL database - or at least one that supports `SLEEP()`.

2 Using a timing attack to reconoiter the database

Now we can using timing in a query that references a possible table name to see if such a table exists in the database. For example, we can try the query

```
SELECT * FROM user_accounts WHERE email_addr = 'x'; SELECT COUNT(*) FROM users WHERE SLEEP(20);
-'
```

If there is a table named `users` (there is) then this query will take over 20 seconds to complete. On the other hand, if we reference a table that does not exist in the database, it will return an error message right away.

Here is a challenge for you: There is a table in the database whose name is the surname of somebody in the class. Can you use SQL timing to find out what it is?

You can use similar techniques to find column names like we did in the previous lab.

3 Conclusions

It turns out that any time a program responds more or less slowly depending on the outcome of an action it can be a source of information for an attacker. It the case of database systems it's inevitable that this happens because it's a necessary part of how the database works. We can, however, avoid sharing that information with an attacker be guarding against SQL injection in the first place.