Blind SQL Injection

Source:

https://owasp.org/www-community/attacks/Blind_SQL_Injection

Description

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible.

Threat Modeling

Same as for SQL Injection

Risk Factors

Same as for SQL Injection

Examples

An attacker may verify whether a sent request returned true or false in a few ways:

Content-based

Using a simple page, which displays an article with given ID as the parameter, the attacker may perform a couple of simple tests to determine if the page is vulnerable to SQL Injection attacks.

Example URL:

```
http://newspaper.com/items.php?id=2
```

sends the following query to the database:

```
SELECT title, description, body FROM items WHERE ID = 2
```

The attacker may then try to inject a query that returns ‘false’:

```
http://newspaper.com/items.php?id=2 and 1=2
```

Now the SQL query should looks like this:

```
SELECT title, description, body FROM items WHERE ID = 2 and 1=2
```

If the web application is vulnerable to SQL Injection, then it probably will not return anything. To make sure, the attacker will inject a query that will return ‘true’:

```
http://newspaper.com/items.php?id=2 and 1=1
```

If the content of the page that returns ‘true’ is different than that of the page that returns ‘false’, then the attacker is able to distinguish when the executed query returns true or false.

Once this has been verified, the only limitations are privileges set up by the database administrator, different SQL syntax, and the attacker’s imagination.

Time-based

This type of blind SQL injection relies on the database pausing for a specified amount of time, then returning the results, indicating successful SQL query executing. Using this method, an attacker enumerates each letter of the desired piece of data using the following logic:

If the first letter of the first database's name is an 'A', wait for 10 seconds.

If the first letter of the first database's name is an 'B', wait for 10 seconds. etc.

Microsoft SQL Server

```
http://www.site.com/vulnerable.php?id=1' waitfor delay '00:00:10'--
```

MySQL

```
SELECT IF(expression, true, false)
```

Using some time-taking operation e.g. BENCHMARK(), will delay server responses if the expression is True.

```
BENCHMARK(5000000,ENCODE('MSG','by 5 seconds'))
```

will execute the ENCODE function 5000000 times.

Depending on the database server's performance and load, it should take just a moment to finish this operation. The important thing is, from the attacker's point of view, to specify a high-enough number of BENCHMARK() function repetitions to affect the database response time in a noticeable way.

Example combination of both queries:

```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) =
CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM
users WHERE user_id = 1;
```

If the database response took a long time, we may expect that the first user password character with user_id = 1 is character '2'.

```
(CHAR(50) == '2')
```

Using this method for the rest of characters, it's possible to enumerate entire passwords stored in the database. This method works even when the attacker injects the SQL queries and the content of the vulnerable page doesn't change.

Obviously, in this example, the names of the tables and the number of columns was specified. However, it's possible to guess them or check with a trial and error method.

Databases other than MySQL also have time-based functions which allow them to be used for time-based attacks:

- MS SQL: `'WAIT FOR DELAY '0:0:10''`
- PostgreSQL: `pg_sleep()`

Conducting Blind SQL Injection attacks manually is very time consuming, but there are a lot of tools which automate this process. One of them is SQLMap partly developed within OWASP grant program. On the other hand, tools of this kind are very sensitive to even small deviations from the rule. This includes:

- scanning other website clusters, where clocks are not ideally synchronized,
- WWW services where argument acquiring method was changed, e.g. from `/index.php?ID=10` to `/ID,10`

Remote Database Fingerprinting

If the attacker is able to determine when their query returns True or False, then they may fingerprint the RDBMS. This will make the whole attack much easier. If the time-based approach is used, this helps determine what type of database is in use. Another popular methods to do this is to call functions which will return the current date. MySQL, MSSQL, and Oracle have different functions for that, respectively `now()`, `getdate()`, and `sysdate()`.