What Are Blind SQL Injections

SQL Injection (SQLi) (https://www.acunetix.com/websitesecurity/sql-injection/) vulnerabilities are one of the oldest and most common web security issues. The latest OWASP Top 10 (https://www.acunetix.com/blog/articles/owasp-top-10-2017/) list still features this type of attack at the number one spot as the biggest web application security risk.

The most common method used to check for a normal SQL Injection vulnerability is adding a single quote (' – ASCII value 39). If you use a single quote in a field or parameter that is passed directly to an SQL statement, the database server will report an error. If the database server is configured to show SQL errors, the web server will display the error in the web application. This way an attacker is certain that the field is vulnerable to SQL Injection attacks. The error could look similar to the following one (from Microsoft SQL Server):

```
Microsoft SQL Native Client error '80040e14'
Unclosed quotation mark after the character string ''.
/target.asp, line 9
```

After the attacker verifies the presence of an SQL Injection vulnerability, they can try different requests (often involving UNION SELECT statements) to receive information about the database in error responses. They can use it to fingerprint the database (find out if it's MySQL, PostgreSQL, Oracle, MSSQL, etc. and which version), build the database schema, retrieve data from any table in the database, and escalate the attack.

Web server administrators quickly realized that showing errors to the general public is not a wise thing to do, so they started suppressing detailed error messages. This is a flawed solution because it does not address the underlying problem. The SQL interpreter can still parse user input as part of an SQL query.

Attackers came up with methods to go around the lack of error messages and still know if the input is being interpreted as an SQL statement. This is how the *Blind SQL Injection* technique was born (sometimes called *Inferential SQL Injection*). There are two variants of this technique that are commonly used: *Content-based Blind SQL Injection* and *Time-based Blind SQL Injection*.

Content-based Blind SQL Injection

In the case of a Content-based Blind SQL Injection attack, the attacker makes different SQL queries that ask the database TRUE or FALSE questions. Then they analyze differences in responses between TRUE and FALSE statements.

This is an example of a web page of an online shop, which displays items that are for sale. The following link will display details about item 34, which are retrieved from a database.

```
http://www.shop.local/item.php?id=34
```

The SQL statement used for this request is:

```
SELECT column_name, column_name_2 FROM table_name WHERE id = 34
```

The attacker may manipulate the request to:

```
http://www.shop.local/item.php?id=34 and 1=2
```

The SQL statement changes to:

```
SELECT column_name_2 FROM table_name WHERE ID = 34 and 1=2SELECT name, description, price
FROM Store_table WHERE ID = 34 and 1=2
```

This will cause the query to return FALSE and no items are displayed in the list. The attacker then proceeds to change the request to:

```
http://www.shop.local/item.php?id=34 and 1=1
```

And the SQL statement changes to:

```
SELECT column_name, column_name_2 FROM table_name WHERE ID = 34 and 1=1SELECT name, descr
iption, price FROM Store_table WHERE ID = 34 and 1=1
```

This returns TRUE, and the details of item with ID 34 are shown. This is a clear indication that the page is vulnerable.

Time-based Blind SQL Injection

In the case of time-based attacks, the attacker makes the database perform a time-intensive operation. If the web site does not return a response immediately, the web application is vulnerable to Blind SQL Injection. A popular time-intensive operation is the *sleep* operation.

Based on the previous example, the attacker would first benchmark the web server response time for a regular query. They would then issue the following request:

```
http://www.shop.local/item.php?id=34 and if(1=1, sleep(10), false)
```

The web application is vulnerable if the response is delayed by 10 seconds.

Consequences of Blind SQL Injections

Blind SQL Injections are often used to build the database schema and get all the data in the database. This is done using brute force techniques and requires many requests but may be automated by attackers using SQL Injection tools.

Acunetix can detect Blind SQL Injection vulnerabilities. Acunetix also includes a Blind SQL Injector tool (https://www.acunetix.com/blog/docs/blind-sql-injector/), which allows the penetration tester to verify that the Blind SQL vulnerability exists and demonstrate the consequences of the vulnerability. Take a demo (https://www.acunetix.com/web-vulnerability-scanner/demo/) and find out more about running Blind SQLi scans against your website or web application.

(//www.acunetix.com/vulnerability-scanner/customers/)

(//www.acunetix.com/vulnerability-scanner/customers/)

(//www.acunetix.com/vulnerability-scanner/customers/)

(//www.acunetix.com/vulnerability-scanner/customers/)

(//www.acunetix.com/vulnerability-scanner/customers/)

(//www.acunetix.com/vulnerability-scanner/customers/)

Product Information

AcuSensor Technology (https://www.acunetix.com/vulnerability-scanner/acusensor-technology/)

AcuMonitor Technology (https://www.acunetix.com/vulnerability-scanner/acumonitor-technology/)

Network Security Scanner (https://www.acunetix.com/vulnerability-scanner/network-security-scanner/)

Acunetix Integrations (https://www.acunetix.com/vulnerability-scanner/acunetix-integrations/)

JavaScript Security (https://www.acunetix.com/vulnerability-scanner/javascript-html5-security/)

Use Cases

Penetration Testing Software (https://www.acunetix.com/vulnerability-scanner/penetration-testing-software/)

Website Security Scanner (https://www.acunetix.com/vulnerability-scanner/website-security-scanner/)

External Vulnerability Scanner (https://www.acunetix.com/vulnerability-scanner/external-vulnerability-scanner/)

Web Application Security (https://www.acunetix.com/vulnerability-scanner/web-application-security/)

Vulnerability Management Software (https://www.acunetix.com/vulnerability-scanner/vulnerability-management-software/)

Website Security

Cross-site Scripting (https://www.acunetix.com/websitesecurity/cross-site-scripting/)

SQL Injection (https://www.acunetix.com/websitesecurity/sql-injection/)

Reflected XSS (https://www.acunetix.com/websitesecurity/xss/)

CSRF Attacks (https://www.acunetix.com/websitesecurity/csrf-attacks/)

Directory Traversal (https://www.acunetix.com/websitesecurity/directory-traversal/)

Learn More

TLS Security (https://www.acunetix.com/blog/articles/tls-security-what-is-tls-ssl-part-1/)

WordPress Security (https://www.acunetix.com/vulnerability-scanner/wordpress-

security-scan/)

Acunetix Alternatives (https://www.acunetix.com/comparisons/why-choose-acunetix-over-alternatives/)

Web Service Security (https://www.acunetix.com/websitesecurity/web-services-wp/)

Prevent SQL Injection (https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/)

Company

About Us (https://www.acunetix.com/company/)

Customers (https://www.acunetix.com/vulnerability-scanner/customers/)

Become a Partner (/apply)

Jobs (https://www.acunetix.com/jobs/)

Contact (https://www.acunetix.com/company/contact/)

Documentation

Case Studies (https://www.acunetix.com/vulnerability-scanner/case-studies/)

Support (https://www.acunetix.com/support/)

Videos (https://www.acunetix.com/support/videos/)

Web Vulnerabilities (//www.acunetix.com/vulnerabilities/web/)

Webinars (https://www.acunetix.com/webinars/)

Whitepapers (https://www.acunetix.com/websitesecurity/whitepapers/)

 Acunetix Online Login (https://online.acunetix.com)

(https://www.acunetix.com)

Privacy Policy (https://www.acunetix.com/company/privacy/)

Terms and Conditions (https://www.acunetix.com/company/terms_conditions/)

Sitemap (https://www.acunetix.com/sitemap/)