What is SQL Injection (SQLi) and How to Prevent It

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists injections in their OWASP Top 10 2017 document as the number one threat to web application security.

How and Why Is an SQL Injection Attack Performed

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

- Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
- SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
- SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
- You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.
- In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

There are several types of SQL Injection attacks: in-band SQLi (using database errors or UNION commands), blind SQLi, and out-of-band SQLi. You can read more about them in the following articles: Types of SQL Injection (SQLi), Blind SQL Injection: What is it.

To follow step-by-step how an SQL Injection attack is performed and what serious consequences it may have, see: Exploiting SQL Injection: a Hands-on Example.

Simple SQL Injection Example

The first example is very simple. It shows, how an attacker can use an SQL Injection vulnerability to go around application security and authenticate as the administrator.

The following script is pseudocode executed on a web server. It is a simple example of authenticating with a username and a password. The example database has a table named `users` with the following columns: `username` and `password`.

```
# Define POST variables
uname = request.POST['username']
```

```
passwd = request.POST['password']

# SQL query vulnerable to SQLi
sql = "SELECT id FROM users WHERE username='" + uname + "' AND password='"
+ passwd + "'"

# Execute the SQL statement
database.execute(sql)
```

These input fields are vulnerable to SQL Injection. An attacker could use SQL commands in the input in a way that would alter the SQL statement executed by the database server. For example, they could use a trick involving a single quote and set the `passwd` field to:

```
password' OR 1=1
```

As a result, the database server runs the following SQL query:

```
SELECT id FROM users WHERE username='username' AND password='password' OR
1=1'
```

Because of the `OR 1=1` statement, the `WHERE` clause returns the first `id` from the `users` table no matter what the `username` and `password` are. The first user `id` in a database is very often the administrator. In this way, the attacker not only bypasses authentication but also gains administrator privileges. They can also comment out the rest of the SQL statement to control the execution of the SQL query further:

```
-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite
' OR '1'='1' --
' OR '1'='1' /*
-- MySQL
' OR '1'='1' #
-- Access (using null characters)
' OR '1'='1' %00
' OR '1'='1' %16
```

Example of a Union-Based SQL Injection

One of the most common types of SQL Injection uses the UNION operator. It allows the attacker to combine the results of two or more SELECT statements into a single result. The technique is called *union*-based SQL Injection.

The following is an example of this technique. It uses the web page **testphp.vulnweb.com**, an intentionally vulnerable website hosted by Acunetix.

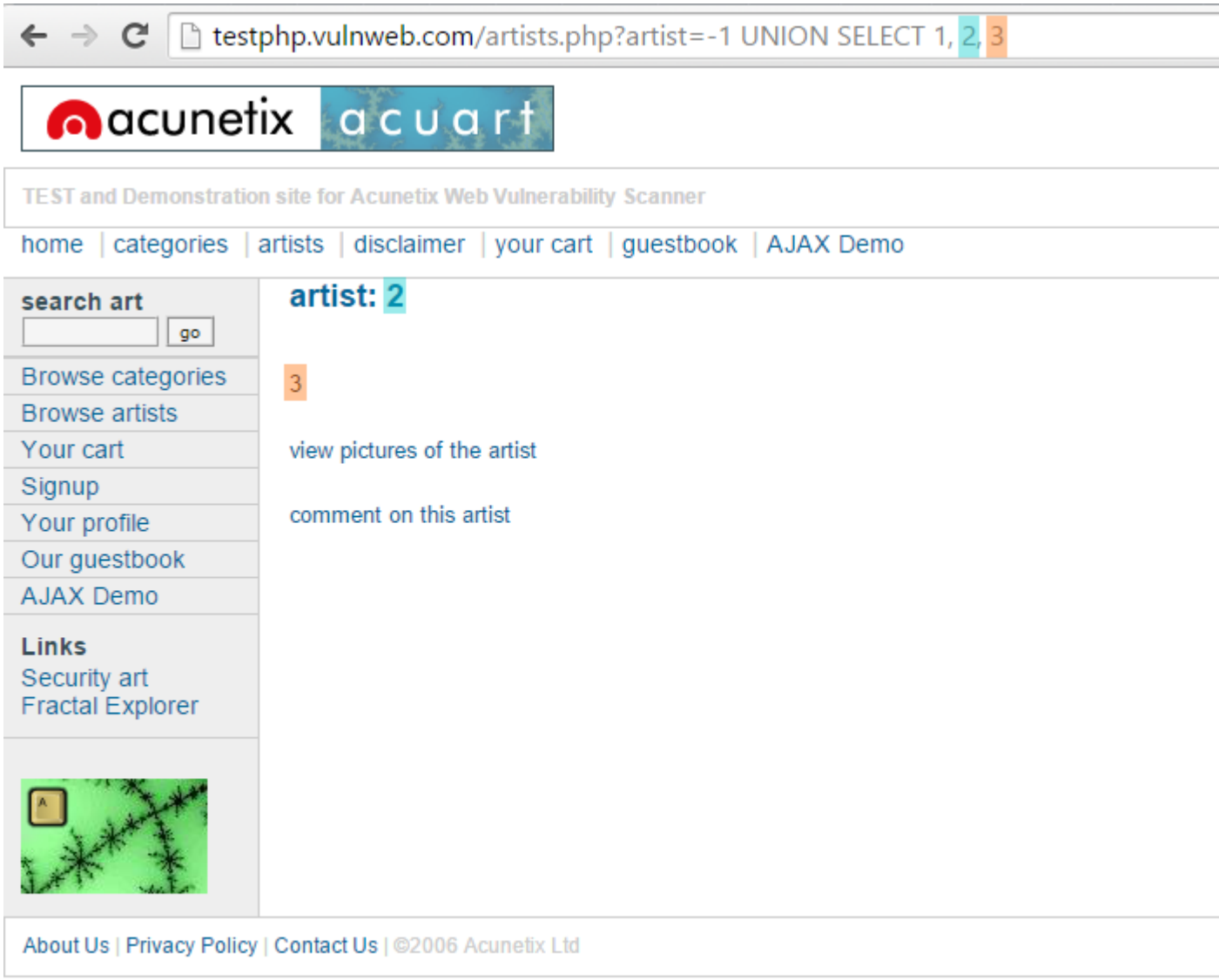
The following HTTP request is a normal request that a legitimate user would send:

```
GET http://testphp.vulnweb.com/artists.php?artist=1 HTTP/1.1
Host: testphp.vulnweb.com
```

The `artist` parameter is vulnerable to SQL Injection. The following payload modifies the query to look for an inexistent record. It sets the value in the URL query string to `-1`. Of course, it could be any other value that does not exist in the database. However, a negative value is a good guess because an identifier in a database is rarely a negative number.

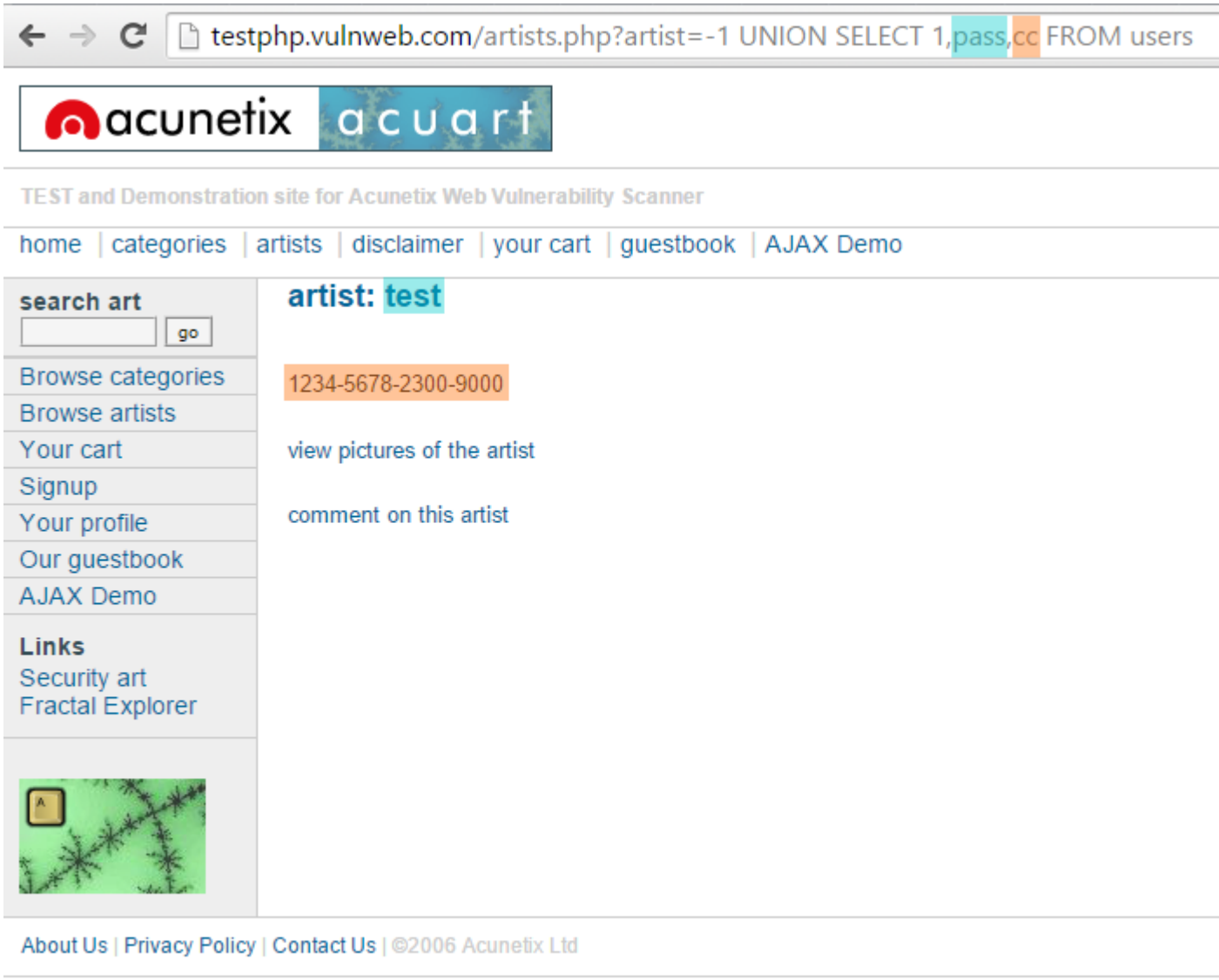
In SQL Injection, the `UNION` operator is commonly used to attach a malicious SQL query to the original query intended to be run by the web application. The result of the injected query will be joined with the result of the original query. This allows the attacker to obtain column values from other tables.

```
GET http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, 2, 3
HTTP/1.1
Host: testphp.vulnweb.com
```



The following example shows how an SQL Injection payload could be used to obtain more meaningful data from this intentionally vulnerable site:

```
GET http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,pass,cc
FROM users WHERE uname='test' HTTP/1.1
Host: testphp.vulnweb.com
```



How to Prevent an SQL Injection

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.

To learn how to prevent SQL Injection attacks in the PHP language, see: Preventing SQL Injection Vulnerabilities in PHP Applications and Fixing Them. To find out how to do it in many other different programming languages, refer to the Bobby Tables guide to preventing SQL Injection.

How to Prevent SQL Injections (SQLi) – Generic Tips

Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.

Step 1: Train and maintain awareness

To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page.

Step 2: Don’t trust any user input

Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input.

Step 3: Use whitelists, not blacklists

Don’t filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only.

Step 4: Adopt the latest technologies

Older web development technologies don’t have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi.

Step 5: Employ verified mechanisms

Don't try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures.

Step 6: Scan regularly (with Acunetix)

SQL Injections may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build.

Types of SQL Injection (SQLi)

SQL Injection can be used in a range of ways to cause serious problems. By levering SQL Injection, an attacker could bypass authentication, access, modify and delete data within a database. In some cases, SQL Injection can even be used to execute commands on the operating system, potentially allowing an attacker to escalate to more damaging attacks inside of a network that sits behind a firewall.

SQL Injection can be classified into three major categories – *In-band SQLi*, *Inferential SQLi* and *Out-of-band SQLi*.

In-band SQLi (Classic SQLi)

In-band SQL Injection is the most common and easy-to-exploit of SQL Injection attacks. In-band SQL Injection occurs when an attacker is able to use the same communication channel to both launch the attack and gather results.

The two most common types of in-band SQL Injection are *Error-based SQLi* and *Union-based SQLi*.

Error-based SQLi

Error-based SQLi is an in-band SQL Injection technique that relies on error messages thrown by the database server to obtain information about the structure of the database. In some cases, error-based SQL injection alone is enough for an attacker to enumerate an entire database. While errors are very useful during the development phase of a web application, they should be disabled on a live site, or logged to a file with restricted access instead.

Union-based SQLi

Union-based SQLi is an in-band SQL injection technique that leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result which is then returned as part of the HTTP response.

Inferential SQLi (Blind SQLi)

Inferential SQL Injection, unlike in-band SQLi, may take longer for an attacker to exploit, however, it is just as dangerous as any other form of SQL Injection. In an inferential SQLi attack, no data is actually transferred via the web application and the attacker would not be able to see the result of an attack in-band (which is why such attacks are commonly referred to as "blind SQL Injection attacks"). Instead, an attacker is able to reconstruct the database structure by sending payloads, observing the web application's response and the resulting behavior of the database server.

The two types of inferential SQL Injection are *Blind-boolean-based SQLi* and *Blind-time-based SQLi*.

Boolean-based (content-based) Blind SQLi

Boolean-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result.

Depending on the result, the content within the HTTP response will change, or remain the same. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database, character by character.

Time-based Blind SQLi

Time-based SQL Injection is an inferential SQL Injection technique that relies on sending an SQL query to the database which forces the database to wait for a specified amount of time (in seconds) before responding. The response time will indicate to the attacker whether the result of the query is TRUE or FALSE.

Depending on the result, an HTTP response will be returned with a delay, or returned immediately. This allows an attacker to infer if the payload used returned true or false, even though no data from the database is returned. This attack is typically slow (especially on large databases) since an attacker would need to enumerate a database character by character.

Out-of-band SQLi

Out-of-band SQL Injection is not very common, mostly because it depends on features being enabled on the database server being used by the web application. Out-of-band SQL Injection occurs when an attacker is unable to use the same channel to launch the attack and gather results.

Out-of-band techniques, offer an attacker an alternative to inferential time-based techniques, especially if the server responses are not very stable (making an inferential time-based attack unreliable).

Out-of-band SQLi techniques would rely on the database server's ability to make DNS or HTTP requests to deliver data to an attacker. Such is the case with Microsoft SQL Server's `xp_dirtree` command, which can be used to make DNS requests to a server an attacker controls; as well as Oracle Database's UTL_HTTP package, which can be used to send HTTP requests from SQL and PL/SQL to a server an attacker controls