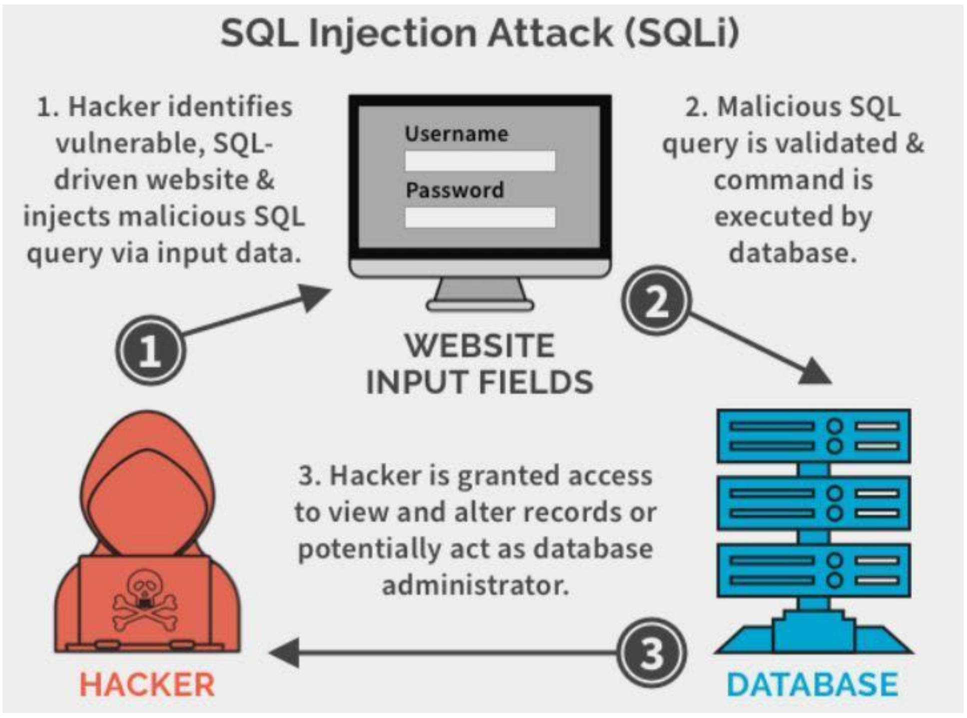
SQL Injection

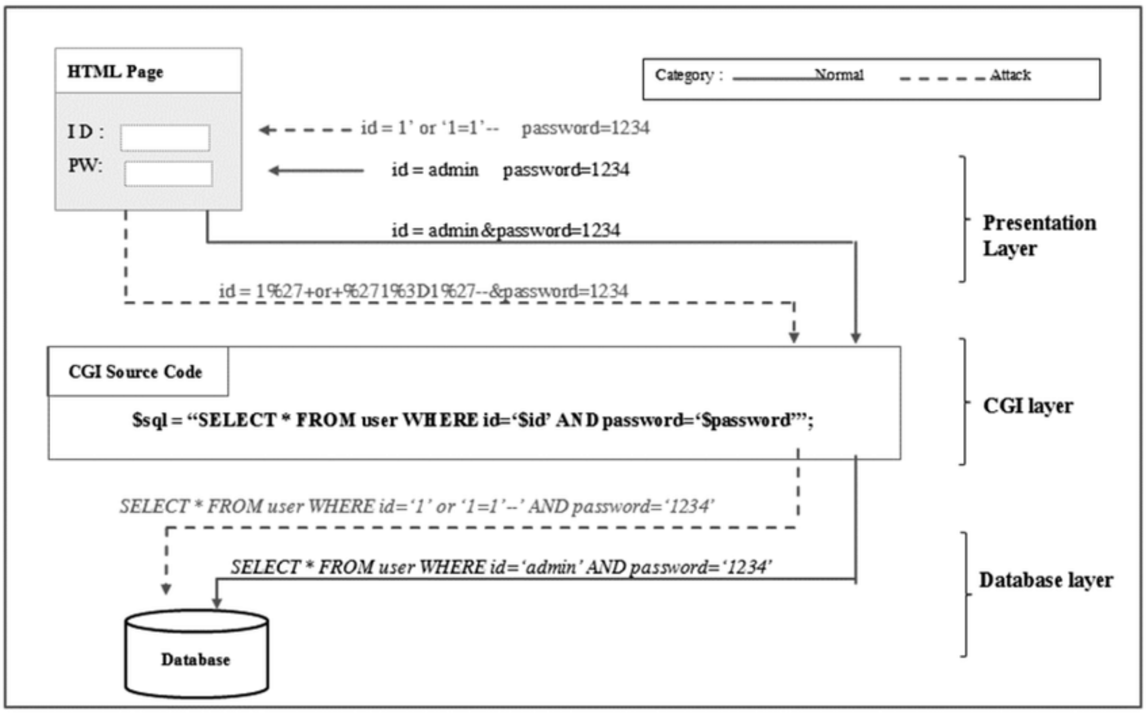
I. What is SQL injection ?

SQL injection, also known as SQLI, is a type of an injection attack that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.


SQL Injection Attack (SQLi)
1. Hacker identifies vulnerable, SQL-driven website & injects malicious SQL query via input data.
Username
Password
WEBSITE INPUT FIELDS
2. Malicious SQL query is validated & command is executed by database.
3. Hacker is granted access to view and alter records or potentially act as database administrator.
1
2
3
HACKER
DATABASE

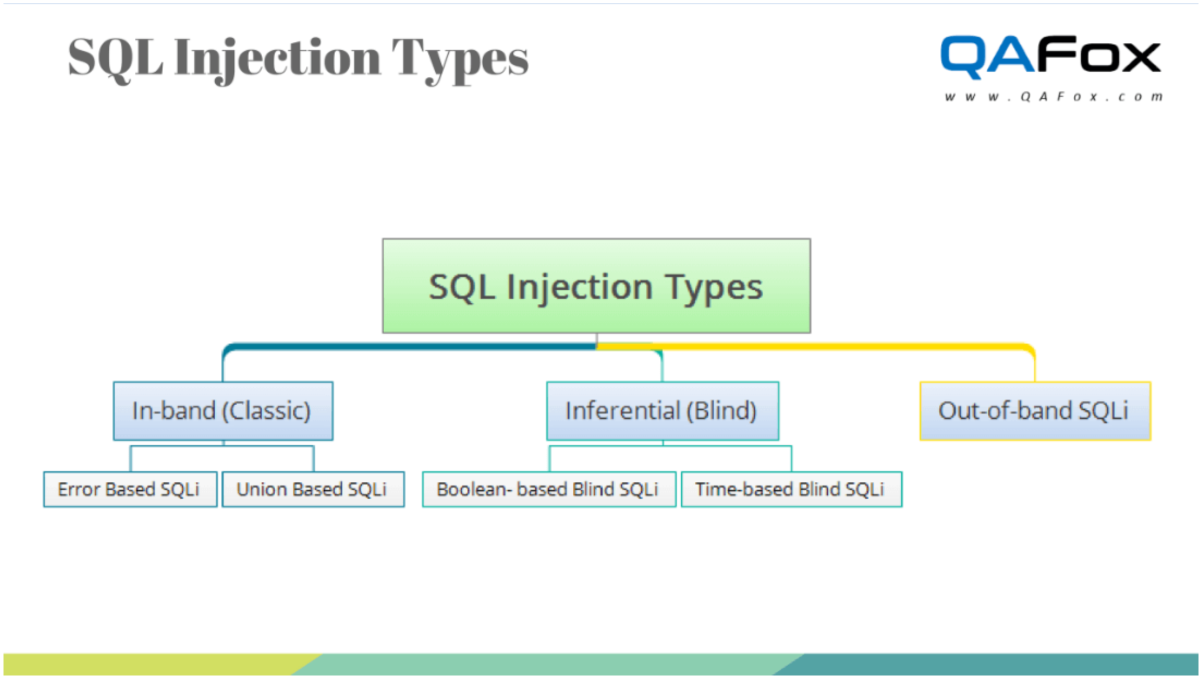
II. How is an SQL injection attack performed ?

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.


HTML Page
ID :
PW:
Category : Normal Attack
id = 1' or '1=1'-- password=1234
id = admin password=1234
id = admin&password=1234
id = 1%27+or+%271%3D1%27--&password=1234
Presentation Layer
CGI Source Code
$sql = "SELECT * FROM user WHERE id='$id' AND password='$password'";
CGI layer
SELECT * FROM user WHERE id='1' or '1=1'--' AND password='1234'
SELECT * FROM user WHERE id='admin' AND password='1234'
Database
Database layer

III. Types of SQL injetion.

SQL injections typically fall under three categories: In-band SQLi (Classic), Inferential SQLi (Blind) and Out-of-band SQLi. You can classify SQL injections types based on the methods they use to access backend data and their damage potential.



In-band SQLi

The attacker uses the same channel of communication to launch their attacks and to gather their results. In-band SQLi's simplicity and efficiency make it one of the most common types of SQLi attack. There are two sub-variations of this method:

- **Error-based SQLi**—the attacker performs actions that cause the database to produce error messages. The attacker can potentially use the data provided by these error messages to gather information about the structure of the database.
- **Union-based SQLi**—this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response. This response may contain data that can be leveraged by the attacker.

```
' UNION SELECT 'User',NULL,NULL,NULL --
Username = Admin
Password = Admin
```

Inferential (Blind) SQLi

The attacker sends data payloads to the server and observes the response and behavior of the server to learn more about its structure. This method is called blind SQLi because the data is not transferred from the website database to the attacker, thus the attacker cannot see information about the attack in-band.

Blind SQL injections rely on the response and behavioral patterns of the server so they are typically slower to execute but may be just as harmful. Blind SQL injections can be classified as follows:

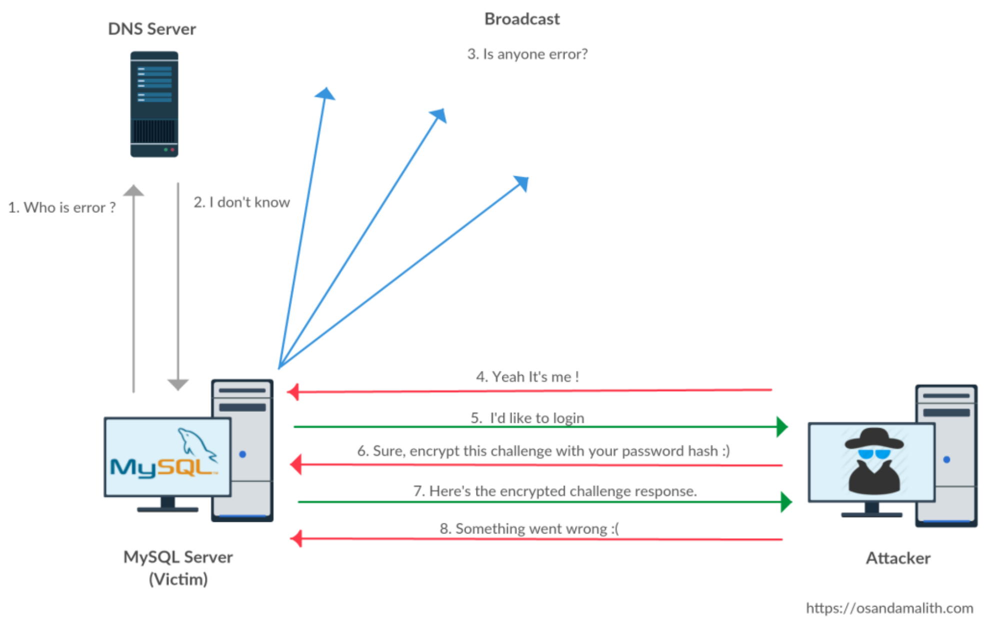
- **Boolean**—that attacker sends a SQL query to the database prompting the application to return a result. The result will vary depending on whether the query is true or false. Based on the result, the information within the HTTP response will modify or stay unchanged. The attacker can then work out if the message generated a true or false result.

- **Time-based**—attacker sends a SQL query to the database, which makes the database wait (for a period in seconds) before it can react. The attacker can see from the time the database takes to respond, whether a query is true or false. Based on the result, an HTTP response will be generated instantly or after a waiting period. The attacker can thus work out if the message they used returned true or false, without relying on data from the database.

Out-of-band SQLi

The attacker can only carry out this form of attack when certain features are enabled on the database server used by the web application. This form of attack is primarily used as an alternative to the in-band and inferential SQLi techniques.

Out-of-band SQLi is performed when the attacker can’t use the same channel to launch the attack and gather information, or when a server is too slow or unstable for these actions to be performed. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker.



How to detect SQL injection vulnerabilities

SQL injection can be detected manually by using a systematic set of tests against every entry point in the application. This typically involves:

- Submitting the single quote character ' and looking for errors or other anomalies.
- Submitting some SQL-specific syntax that evaluates to the base (original) value of the entry point, and to a different value, and looking for systematic differences in the resulting application responses.
- Submitting Boolean conditions such as OR 1=1 and OR 1=2, and looking for differences in the application's responses.
- Submitting payloads designed to trigger time delays when executed within an SQL query, and looking for differences in the time taken to respond.
- Submitting OAST payloads designed to trigger an out-of-band network interaction when executed within an SQL query, and monitoring for any resulting interactions.

IV. Impact of SQL injection

- **Extract sensitive information**, like Social Security numbers, or credit card details.
- **Enumerate the authentication details of users registered on a website,** so these logins can be used in attacks on other sites.

- **Delete data or drop tables**, corrupting the database, and making the website unusable.
- **Inject further malicious code** to be executed when users visit the site.

V. How to prevent SQL injection?

- **Parameterized Statements**

Programming languages talk to SQL databases using **database drivers.** A driver allows an application to construct and run SQL statements against a database, extracting and manipulating data as needed. **Parameterized statements** make sure that the parameters (i.e. inputs) passed into SQL statements are treated in a safe manner.

- **Escaping Inputs**

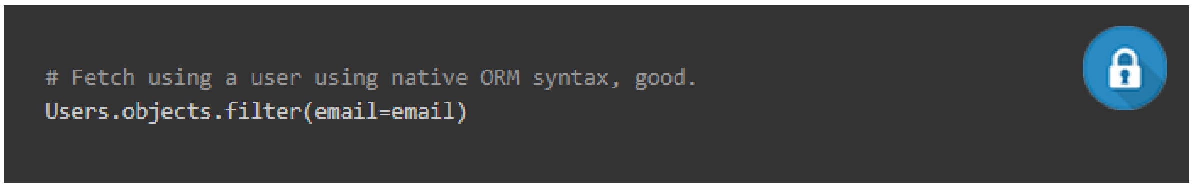
If you are unable to use parameterized statements or a library that writes SQL for you, the next best approach is to ensure proper escaping of special string characters in input parameters.

- **Sanitizing Inputs**

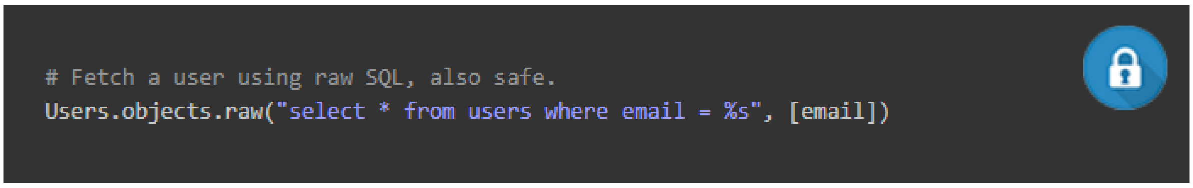
Sanitizing inputs is a good practice for all applications. In our example hack, the user supplied a password as ' or 1=1--, which looks pretty suspicious as a password choice.

- **Code Example in Python**

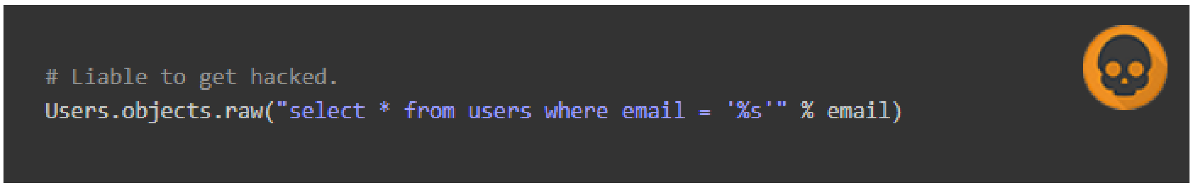
Django



```
# Fetch using a user using native ORM syntax, good.
Users.objects.filter(email=email)
```



```
# Fetch a user using raw SQL, also safe.
Users.objects.raw("select * from users where email = %s", [email])
```



```
# Liable to get hacked.
Users.objects.raw("select * from users where email = '%s'" % email)
```

Reference:

Hackplanning: https://www.hacksplaining.com

Web Security Academy: https://portswigger.net/web-security/sql-injection

Acunetix: https://www.acunetix.com/websitesecurity/sql-injection

Impreva: https://www.imperva.com/learn/application-security/sql-injection-sqli/

Netsparker: https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/