What is SQLi(Injection)?

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 unauthorised 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.

Types of SQLi:

There are several types of SQL Injection attacks: in-band SQLi (using database errors or UNION commands), blind SQLi, and out-of-band SQLi.

Example of SQLi:

In this simple SQL injection example, the code is part of a web server application that handles user authentication. Let's break down what's happening:

1. **User Input Retrieval:**
   ```python
   uname = request.POST['username']
   passwd = request.POST['password']
   ```
   The code retrieves the username and password from the user input in a web form.

2. **SQL Query Building:**
   ```python
    sql = "SELECT id FROM users WHERE username='" + uname + "' AND password='" + passwd + "'"
   ```
    The application constructs an SQL query to check if there's a match in the `users` table for the provided username and password.

3. **SQL Injection Vulnerability:**
    The vulnerability lies in how the SQL query is constructed. If an attacker provides malicious input for the password field, they can manipulate the SQL query. For example:
   ```sql
   password' OR 1=1
   ```
   This input, when injected, changes the SQL query to:
   ```sql
   ```

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

4. **Exploiting the Vulnerability:**
The injected SQL statement includes `OR 1=1`, which always evaluates to true. As a result, the query returns the first user ID from the `users` table, effectively bypassing the need for a correct username and password. This is particularly dangerous in cases where the first user is often an administrator.

5. **Consequences:**
The attacker not only bypasses the authentication but also gains administrator privileges since the query effectively becomes:
```sql
SELECT id FROM users WHERE username='username' AND (password='password' OR 1=1)'
```

Prevention Steps to Prevent SQLi(Injection)

1.Temporary Protection:

If an SQL Injection vulnerability is discovered, using a web application firewall can temporarily sanitise input while a more permanent fix is implemented.

2.Preventive Measures:

Input validation and parameterized queries (prepared statements) are the most reliable ways to prevent SQL Injection attacks.
Developers should avoid using user input directly in SQL queries and must sanitise all inputs, not just those from web forms like login forms.

3.Visibility of Errors:

Turning off the visibility of database errors on production sites is recommended to prevent attackers from gaining information about the database through error messages.

4. Training and Awareness:

- All individuals involved in building the web application, including developers, QA staff, DevOps, and SysAdmins, should undergo security training to be aware of SQL Injection risks.

*5. **Untrusted User Input:***

- Treat all user input as untrusted, regardless of the source. Authenticated and internal user inputs should be treated with the same level of caution as public inputs.

*6. **Whitelists over Blacklists:***
```

- Use whitelists instead of blacklists to filter and verify user input. Blacklists may be circumvented by clever attackers.

*7. **Adopting Latest Technologies:***

- Use the latest versions of development environments and languages, along with associated technologies that provide better protection against SQL Injection. For example, preferring PDO over MySQLi in PHP.

*8. **Employing Verified Mechanisms:***

- Instead of building SQL Injection protection from scratch, leverage mechanisms provided by modern development technologies. This includes using parameterized queries or stored procedures.

*9. **Regular Scanning:***

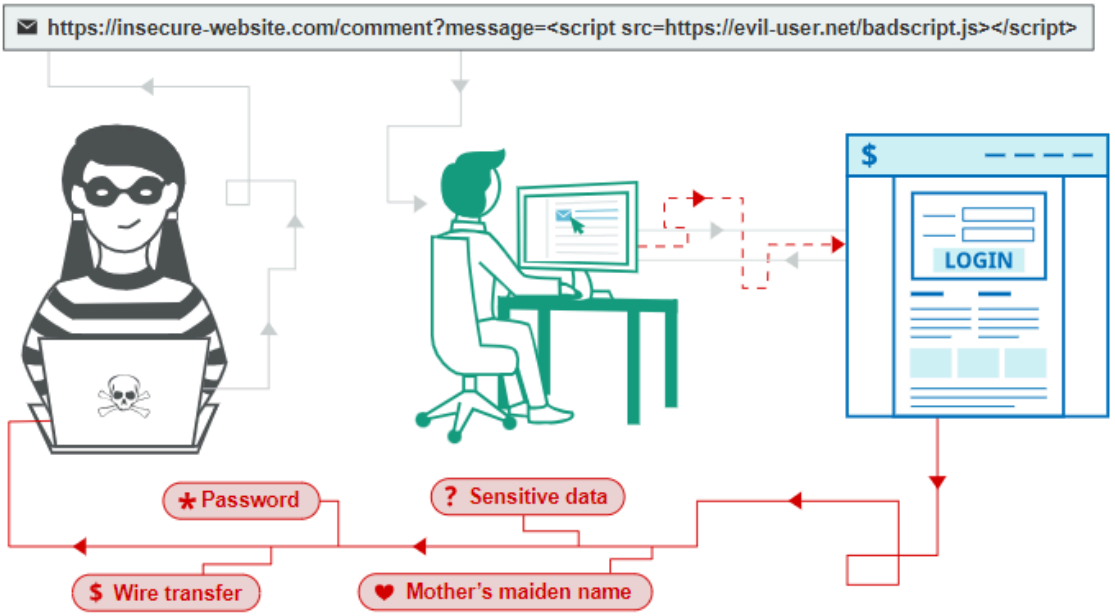
- Regularly scan web applications for vulnerabilities using a web vulnerability scanner like Acunetix. Automated scans can be integrated into the development process, for example, using the Acunetix plugin with Jenkins.

What is cross-site scripting (XSS)?

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. Cross-Site Scripting (XSS) is a web security issue that lets a bad actor mess with how users interact with a vulnerable website. It helps the attacker bypass a security rule called the "same origin policy," which usually keeps different websites separate. With XSS, the attacker can pretend to be a regular user, do whatever that user can, and peek into their data. If the user has special access, the attacker might even take over the whole website, gaining control over everything.

How does XSS work?

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.



What are the types of XSS attacks?

There are three main types of XSS attacks. These are:

Reflected XSS, where the malicious script comes from the current HTTP request.

Stored XSS, where the malicious script comes from the website's database.

DOM-based XSS, where the vulnerability exists in client-side code rather than server-side code.

How to prevent XSS attacks?

Preventing cross-site scripting is insignificant in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data. In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.

Example of Reflected XSS:

Reflected XSS is the most straightforward type of cross-site scripting. It happens when a website takes data from an incoming request and puts it directly into the response, without doing any safety checks.

Here's a simple example to illustrate a reflected XSS vulnerability:

Suppose you visit a website, https://insecure-website.com, and it has a feature where you can check the status by adding a message in the URL. For example:

- Normal URL: https://insecure-website.com/status?message=All+is+well.

- Response: `<p>Status: All is well.</p>`

Now, if the website doesn't properly handle the input, an attacker can create a harmful link like this:

- Attacker's URL: https://insecure-website.com/status?message=<script>/* Bad stuff here... */</script>

- Response: `<p>Status: <script>/* Bad stuff here... */</script></p>`

If a user clicks on the link created by the attacker, the malicious script runs in the user's browser, using that user's session with the website. At this point, the script can do anything the user can do and access any data available to that user, posing a security risk.

What is Transport Layer Security (TLS)?

Transport Layer Security (TLS) is a widely adopted security protocol aimed at ensuring privacy and data security in Internet communications. It is commonly used to encrypt communication between web applications and servers, securing activities like browsing websites. TLS extends its encryption capabilities to other forms of communication, including email, messaging, and voice over IP (VoIP). Proposed by the Internet Engineering Task Force (IETF), TLS has evolved since its inception in 1999, with the latest version being TLS 1.3, introduced in 2018.

What does TLS do?

There are three main components to what the TLS protocol accomplishes: Encryption, Authentication, and Integrity.

Encryption: hides the data being transferred from third parties.
Authentication: ensures that the parties exchanging information are who they claim to be.
Integrity: verifies that the data has not been forged or tampered with.

What is the difference between TLS and SSL?

TLS evolved from a previous encryption protocol called Secure Sockets Layer (SSL), which was developed by Netscape. TLS version 1.0 actually began development as SSL version 3.1, but the name of the protocol was changed before publication in order to indicate that it was no longer associated with Netscape. Because of this history, the terms TLS and SSL are sometimes used interchangeably.

Why Businesses and Web Applications Should Use TLS:

Using the TLS protocol, particularly with HTTPS, is crucial for businesses and web applications due to the following reasons:

1. **Data Protection:**

- TLS encryption safeguards web applications against data breaches and various cyber attacks, providing a secure environment for transmitting sensitive information.

2. **Industry Standard:**

- TLS-protected HTTPS has become a standard practice for websites. Major browsers, starting with Google Chrome, enforce stricter security measures for non-HTTPS sites. Users are increasingly cautious about websites lacking the HTTPS padlock icon.

3. **User Trust:**

- Everyday Internet users now associate the HTTPS padlock with a secure connection. Utilizing TLS helps build trust with users who are more likely to engage with and trust websites that prioritize secure communication.

What is a TLS Certificate:

For a website or application to implement TLS, it must have a TLS certificate (sometimes referred to as an "SSL certificate") installed on its origin server. Key points about TLS certificates include:

1. Issued by Certificate Authorities (CAs):

- TLS certificates are issued by Certificate Authorities to the individual or business that owns a domain. CAs are trusted entities responsible for verifying the legitimacy of the certificate holder.

2. Ownership and Identity Verification:

- The certificate contains crucial information about the domain owner and the server's public key. This information is vital for validating the identity of the server and ensuring that users are interacting with the legitimate entity associated with the domain.