Part II: Understanding XSS and SQLi

Cross-Site Scripting (XSS)

Definition and Overview

Cross-Site Scripting or XSS is an enabling security weakness identified on web applications that permits attackers to inject malicious scripts into commonly viewed Web pages. These scripts run in the victim's browser environment and results in unauthorized actions or exposure of other data.

How XSS Attacks Work

Cross site scripting vulnerabilities happen when an application has user input and then uses it for generating the HTML output without providing input validation. For instance, if a comment form on a site accepts <script>alert('Hacked!');</script> as input, then this script may be executed on other users visiting the comment which they or some program posted.

Types of XSS

1. **Stored XSS**: The malicious script is stored on the web server (e.g., in a database). It is delivered to users when they access the compromised page.
   Example: An attacker posts a malicious script in a forum that executes when other users visit the page.
2. **Reflected XSS**: The script is part of a URL or request and is reflected in the web page's response.
   Example: An attacker sends a crafted link to a user, and clicking it executes the script.
3. **DOM-based XSS**: The vulnerability resides in client-side scripts that manipulate the DOM without proper sanitization.
   Example: A script dynamically updates the page based on unsanitized URL parameters.

Impacts of XSS

- **Data theft**: Attackers can steal cookies, session tokens, or personal information.
- **Account hijacking**: Malicious scripts can impersonate users or hijack their sessions.
- **Phishing attacks**: Scripts can inject fake login forms to steal credentials.
- **Defacement**: Attackers can alter the content of web pages, damaging the brand's reputation.

Prevention of XSS

- **Input validation**: Ensure user inputs are checked for acceptable content and reject suspicious inputs.
- **Output encoding**: Sanitize data before displaying it in a browser. Use functions like `textContent` in JavaScript or libraries such as DOMPurify.
- **Content Security Policy (CSP)**: Define allowed sources of scripts, reducing the impact of injected scripts.

- **Avoid `eval` and `innerHTML`**: These functions increase vulnerability to XSS attacks. Use safer alternatives.

SQL Injection (SQLi)

Definition and Overview
SQL Injection (SQLi) is a web security vulnerability where attackers inject malicious SQL statements into application queries. These statements exploit weaknesses in the application's interaction with the database, allowing attackers to manipulate the database's behavior.

How SQLi Attacks Work
SQLi occurs when an application takes user inputs and directly incorporates them into SQL queries without proper sanitization. For example, the following login query is vulnerable:

```
sql
Copy code
SELECT * FROM users WHERE username = 'user' AND password = 'pass';
```

If the `username` input is `' OR '1'='1`, the query becomes:

```
sql
Copy code
SELECT * FROM users WHERE username = '' OR '1'='1' AND password = 'pass';
```

This query always returns true, bypassing authentication.

Types of SQLi

1. **Classic SQLi**: Injected SQL affects queries in predictable ways.
2. **Blind SQLi**: The attacker learns information through observing responses or behaviors without seeing the data directly.
3. **Time-based Blind SQLi**: Exploits delays in database responses to infer results.
4. **Error-based SQLi**: Relies on database error messages to extract information.

Impacts of SQLi

- **Data leakage**: Access to sensitive information like user credentials or business records.
- **Data manipulation**: Attackers can modify or delete data.
- **Unauthorized access**: Bypass authentication to gain admin privileges.
- **System compromise**: Escalation to backend system control through database exploitation.

Prevention of SQLi

- **Parameterized queries**: Use prepared statements to separate SQL code from user input. *Example (PHP)*:

```
php
Copy code
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ? AND
password = ?");
$stmt->execute([$username, $password]);
```

- **Input sanitization**: Validate and escape user inputs using functions appropriate to the language and database.
- **Use ORM tools**: Tools like Hibernate abstract database queries, reducing direct interaction with SQL.
- **Limit privileges**: Restrict database user permissions to the minimum necessary.
- **Error handling**: Avoid exposing detailed database errors to users. Use generic error messages instead.

How XSS and SQLi Impact Web Applications

Web applications without proper security measures are prime targets for XSS and SQLi.

- **XSS** affects users by stealing personal data, while **SQLi** directly impacts the application's integrity and data security.
- Both vulnerabilities can lead to reputational damage, legal implications, and financial losses.
- Addressing these vulnerabilities requires a combination of secure coding practices, regular vulnerability assessments, and educating developers on secure application design.

References

- OWASP Foundation. (n.d.). *Cross-site scripting (XSS)*. Retrieved from https://owasp.org
- OWASP Foundation. (n.d.). *SQL injection*. Retrieved from https://owasp.org
- Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. *Proceedings of the IEEE International Symposium on Secure Software Engineering*, 13–15.
- Mitre. (n.d.). *CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*. Retrieved from https://cwe.mitre.org
- Mitre. (n.d.). *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. Retrieved from https://cwe.mitre.org