CYBER SECURITY & CRYPTOGRAPHY (8)

techworldthink • March 11, 2022

19. Briefly explain any four Application Security Risks.

Injection

Injection or SQL injection is a type of security attack in which the malicious attacker inserts or injects a query via input data (as simple as via filling a form on the website) from the client-side to the server. If it is successful, the attacker can read data from the database, add new data, update data, delete some data present in the database, issue administrator commands to carry out privileged database tasks, or even issue commands to the operating system in some cases.

Broken Authentication

It is a case where the authentication system of the web application is broken and can result in a series of security threats. This is possible if the adversary carries out a brute force attack to disguise itself as a user, permitting the users to use weak passwords that are either dictionary words or common passwords like “12345678”, “password” etc. This is so common because shockingly 59% of the people use the same passwords on all websites they use. Moreover, 90% of the passwords can be cracked in close to 6 hours! Therefore, it is important to permit users to use strong passwords with a combination of alphanumeric and special characters. This is also possible due to credential stuffing, URL rewriting, or not rotating session IDs.

Sensitive Data Exposure

As the name suggests, this means that sensitive data stored is leaked to malicious attackers. This information can include personal data like name, address, gender, date of birth, personal identification numbers like Aadhar card number or SSN, etc., financial data like account number, credit card numbers, health-related information, etc. This can result in a monetary loss if the attacker uses the financial information of users to carry out online payments (in most cases to cryptocurrency), identity theft, and reputation loss.

XML External Entities

This type is common to web applications that parse XML input. It is carried out when the input in the form of XML references an external entity but is processed by a weak XML parser. It can cause a huge loss to the brand as it can in turn allow distributed denial of service, port scanning, server-side request forgery, disclosure of sensitive information, etc.

Broken Access Control

Access control specifies limits or boundaries in which a user is allowed to operate. For example, the root privileges are usually given to the administrator and not the actual users. Having a broken or leaking access control system can result in unintended information leaks, modifying details of other user accounts, manipulating metadata, acting as the admin, unauthorized API access, etc.

20. Explain the attack scenarios of any four web application security vulnerabilities.

Injection flaws

Injection flaws result from a classic failure to filter untrusted input. It can happen when you pass unfiltered data to the SQL server (SQL injection), to the browser (XSS), to the LDAP server (LDAP injection), or anywhere else. The problem here is that the attacker can inject commands to these entities, resulting in loss of data and hijacking clients' browsers.

The good news is that protecting against injection is "simply" a matter of filtering your input properly and thinking about whether an input can be trusted. But the bad news is that *all* input needs to be properly filtered, unless it can unquestionably be trusted

Broken Authentication

few reasons are...

1. The URL might contain the session id and leak it in the referer header to someone else.

2. The passwords might not be encrypted either in storage or transit.
3. The session ids might be predictable, thus gaining access is trivial.
4. Session fixation might be possible.
5. Session hijacking might be possible, timeouts not implemented right or using HTTP (no SSL security), etc…

Cross Site Scripting (XSS)

This is a fairly widespread input sanitization failure . An attacker gives your web application JavaScript tags on input. When this input is returned to the user unsanitized, the user's browser will execute it. It can be as simple as crafting a link and persuading a user to click it, or it can be something much more sinister. On page load the script runs and, for example, can be used to post your cookies to the attacker.

Prevention: There's a simple web security solution: don't return HTML tags to the client. This has the added benefit of defending against HTML injection, a similar attack whereby the attacker injects plain HTML content (such as images or loud invisible flash players) – not high-impact but surely annoying ("please make it stop!"). Usually, the workaround is simply converting all HTML entities, so that `<script>` is returned as `&lt;script&gt;`. The other often employed method of sanitization is using regular expressions to strip away HTML tags using regular expressions on `<` and `>`, but this is dangerous as a lot of browsers will interpret severely broken HTML just fine. Better to convert all characters to their escaped counterparts.

Security misconfiguration

few reasons are...

1. Running the application with debug enabled in production.
2. Having directory listing enabled on the server, which leaks valuable information.
3. Running outdated software (think WordPress plugins, old PhpMyAdmin).
4. Having unnecessary services running on the machine.

5. Not changing default keys and passwords. (Happens way more frequently than you'd believe!)
6. Revealing error handling information to the attackers, such as stack traces.