Let us discuss some basic recommendation points for common OWASP Top 10 vulnerabilities and some others (assuming that for almost all major vulnerabilities, OWASP has Wiki documents that can be used for Referencing):

**Injections (SQL):** SQL injections exist when an application doesn't sanitise user input before passing it into an SQL command. Here are a few recommendations:

- Sanitise user input and remove or encode special characters like ‘ “ - () # etc.
- Use whitelist filters, which means if a parameter is supposed to have integer values, do not allow non-numeric input. If it is an email field, allow alphanumerics, @ and .(dot)
- Use strong web application firewalls to make exploitation difficult
- Never run SQL server software (MySQL, MsSQL, etc.) as high privilege user such as ‘root’
- Use prepared statements for SQL queries instead of inserting user controlled input into SQL queries
- Remove default databases and accounts such as test, guest, admin, etc.

**Cross Site Scripting (XSS):**
This happens when a user controlled input is reflected somewhere else in an HTML page and is not encoded/sanitised properly. This leads to an attacker being able to inject HTML code in the affected page. So, the fix is to make sure that any input which is taken from a user, while being written into an HTTP response must be cleaned first. So, the recommendations can be written as:

- Perform proper output encoding of special characters like < > “ ‘
- For example, before printing untrusted user-supplied data into an HTML response, convert special characters into HTML encoding (< > " etc.) or URL encode them (%3C %3E %22 %27)

**Insecure Direct Object References (IDOR) + Rate Limiting (Brute forcing) flaws:**
This happens, when an application doesn't check if a user who is requesting a resource actually is requesting data that he is supposed to view/edit. So, the recommendations can be:

- Sensitive information must only be accessible to authorised users
- Implement proper authentication and authorisation checks at every function to make sure the user requesting access to a resource whether to view or edit is his own data and no one else's
- Implement proper Rate Limiting checks that disallows large number of requests from/to a single resource. For example, if from a single device, a single module like OTP check, password check, signup, etc. is being called 100 times in a single minute, it should be blocked
- Similarly, if an account’s password is being attempted to reset even from different devices, the account should be locked for a while
- Implement these checks on the basis of IP addresses and sessions

**Arbitrary File Uploads:**
This happens when applications do not implement proper file type checking and allow uploading of files of different file formats. For example, a PHP file instead of a jpeg profile picture.

- Perform proper server-side validations on what kind of a file user is uploading
- Use white lists filters instead of black list filters. Example: in case of a resume upload feature, instead of banning PHP and .exe files, only allow .pdf, .doc and .docx files
- Rename the files using a code, so that the attacker cannot play around with file names
- Use static file hosting servers like CDNs and file clouds to store files instead of storing them on the application server itself

**Cross Site Request Forgery (CSRF):**
This happens, when an application accepts allows critical actions without any additional tokens like passwords and also allows HTTP requests without checking from where exactly they are coming from. To prevent CSRF, all critical actions must be password protected, CSRF token technology should be used to prevent and all requests must first check and make sure that the request is coming from the website itself and not some third party page. So, recommendations can be:

- Ask the user his password (temporary like OTP or permanent like login password) at every critical action like while deleting account, making a transaction, changing the password etc.
- Implement the concept of CSRF tokens which attach a unique hidden password to every user in every <form>. Read the documentation related to the programming language and framework being used by your website