<https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Website_security>

2 factor authentication

Use captcha for identifying robots,

Restrict brute force attacks by restricting number of incorrect password

Passing secure data over network

Vulnerability in Javascript

Most important thing is to now it is not always our code which is vulnerable it could the packages and libraries we used

Recently we migrated angularjs 1.6.9 to angularjs1.7.9 as it was vulnerable. Now how will I know that package or library is vulnerable we always trust that this library is built by google so it will be perfect but remember no one is perfect even angular from google or react from facebook could be vulnerable

Finally, there are publically available vulnerability scanner tools that can help you find out if you've made any obvious mistakes.(**WAS scan**)

**Different types of vulnerability and how to prevent from them**

Cross-Site Scripting (XSS)

 Allow an attacker to inject client-side scripts through the website into the browsers of other users

The best defense against XSS vulnerabilities is to remove or disable any markup that can potentially contain instructions to run the code. For HTML this includes elements, such as <script>, <object>, <embed>, and <link>.

Many web frameworks automatically sanitize user input from HTML forms by default.

Content-Security-Policy: script-src 'self' https://apis.google.com

**Content-Security-Policy – Is used to whitelist**

Instead of blindly trusting everything that a server delivers, CSP defines the Content-Security-Policy HTTP header that allows you to create a whitelist of sources of trusted content, and instructs the browser to only execute or render resources from those sources.

**script-src** is a directive that controls a set of script-related privileges for a specific page. We’ve specified 'self' as one valid source of script, and https://apis.google.com as another. The browser will dutifully download and execute JavaScript from apis.google.com over HTTPS, as well as from the current page’s origin.

With this policy defined, the browser will simply throw an error instead of loading script from any other source.

SQL injection

To avoid this sort of attack, you must ensure that any user data that is passed to an SQL query cannot change the nature of the query.

We need to correct pattern validation for any user input

### **Cross-Site Request Forgery (CSRF)**

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user’s knowledge or consent.

This type of attack is best explained by example. John is a malicious user who knows that a particular site allows logged-in users to send money to a specified account using an HTTP POST request that includes the account name and an amount of money. John constructs a form that includes his bank details and an amount of money as hidden fields, and emails it to other site users (with the Submit button disguised as a link to a "get rich quick" site).

If a user clicks the submit button, an HTTP POST request will be sent to the server containing the transaction details and any client-side cookies that the browser associated with the site (adding associated site cookies to requests is normal browser behavior). The server will check the cookies, and use them to determine whether or not the user is logged in and has permission to make the transaction.

**Clickjacking**

Your page could be put in a <frame> or <iframe> without your consent. This can result in clickjacking attacks, among other things.

 In that context a hacker can put a hidden layer over your page. Hidden buttons can be used to run bad scripts.

**A number of other concrete steps you can take are:**

**Use more effective password management**. Encourage strong passwords that are changed regularly. Consider two-factor authentication for your site, so that in addition to a password the user must enter another authentication code

**Use captcha to identify robotic behavior**

**We can add mechanism to disable user for multiple wrong attempt.**

**Configure your web server to use**[**HTTPS**](https://developer.mozilla.org/en-US/docs/Glossary/https)**a**nd [HTTP Strict Transport Security](https://developer.mozilla.org/en-US/docs/Web/Security/HTTP_strict_transport_security) (HSTS). HTTPS encrypts data sent between your client and server. This ensures that login credentials, cookies, POST requests data and header information are not easily available to attackers.

**Keep track of the most popular threats** (the [current OWASP list is here](https://owasp.org/www-project-top-ten/)) and address the most common vulnerabilities first.

Use [vulnerability scanning tools](https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools) to perform automated security testing on your site.

Only store and display data that you need. For example, if your users must store sensitive information like credit card details, only display enough of the card number that it can be identified by the user, and not enough that it can be copied by an attacker and used on another site. The most common pattern at this time is to only display the last 4 digits of a credit card number.

**Important to do WAS scan of any third party library** used most of the vulnerability comes from there

[HTTPS](https://www.cloudflare.com/learning/ssl/what-is-https/) is [HTTP](https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/) with encryption. The only difference between the two protocols is that HTTPS uses [TLS](https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/) ([SSL](https://www.cloudflare.com/learning/ssl/what-is-ssl/)) to encrypt normal HTTP requests and responses.

<http://www.jsfuck.com/>

<https://mikewest.org/2013/09/frontend-security-frontendconf-2013/>

<https://www.ssllabs.com/> - For testing ssl in website

Why htpps

What we think is client -------------------server but it is not true

Client ----router – internetservice providers----attackers-------------------server –

Router and internet service provider can access all the data going from client to server and coming back, what they can do ?

They will not harm you as they are repoted orginisation but they can send your details to many people like govt police, they can sell your email ids, phone number and you start getting email spam emails from and calls for credit card