SECURITY IN ANGULARJS

- AngularJS security features is used to build our application securely.
- It is one of the best way to design angular application in such a way that the users cannot change client-side templates.
    - Server and client side templates should be separate. The mixing can cause security threats.
    - Prevent dynamic template generation by using user input.
    - Do not allow to run user input through $scope.$eval.
    - Always use CSP (content security policy), but also add other mechanisms.

Angular templates and Expressions:

If an attacker has access to control Angular templates or expressions, they can use an angular application through XSS attack, indifferent of the version.

Templates and Expressions can be controlled in the following way:

Generating Angular templates on the server containing user-provided content.

This is common mistake, we will generate HTML template through some server-side engine such as PHP, Java and ASP.NET.

.

Passing an expression generated from user-provided content to the following methods on a scope.

- $watch(userContent, ……)
- $watchGroup(userContent, ……)
- $watchCollection(userContent, ….)
- $eval(userContent)
- $evalAsync(usercontent)
- $apply(userContent)
- $applyAsync(userContent)

Passing an expression generated from user-provided content in requests to services that parse expressions in following way:

- $compile(userContent)
- $parse(userContent)
- $interpolate(userContent)

Passing an expression generated from user provided content as built in orderBy filter

{{ value | orderBy : userContent }}

- We can use suitably secure server-side templating to dynamically generate CSS, URLs, etc., but not for generating templates that are compiled/ bootstrapped by Angular.

- To allow user-provided content in an Angular template then the safest option is to ensure that template insert through the ngNonBindable directive.

HTTP Requests:

- The $http service is a core Angular service that facilitates communication with the remote HTTP servers via the browser's XMLHttpRequest object or via JSONP.
- Designing web applications, consider security threats from:
    - **XSRF**
    - **JSON vulnerability**
- Both server and the client must cooperate in order to eliminate threats.
- Angular derives the preconfigured with strategies that address these issues, but this to work backend server cooperation is required.

Cross Site Request Forgery(XSRF/CSRF):

- Cross site request forgery is provided by using the double-submit cookie defense pattern.
- The attacker can trick an authenticated user into unknowingly executing action on our website.
- This means that when you set the XSRF token cookie, AngularJS will send two tokens through each HTTP request.
    - **The cookie, XSRF-TOKEN** - When performing XHR requests, the $http service reads a token from a cookie.

- **The header, X-XSRF-TOKEN-** It sets it as an HTTP header.

- Javascript only runs on our domain and it can read the cookie, and the server can be assured that the XHR came from JavaScript running on our domain.
- The header will not be set for cross domain requests.
- Token must be unique for each user and must be verified by the server.
- The name of the headers can be specified by using the xsrfHeaderName and xsrfCookieName properties of either $httpProvider.defaults at config-time, $http.defaults at run-time, or the per-request config object.

JSON Hijacking Protection:

- The JSON vulnerability allows third party website turn our JSON resource URL into JSONP request under some conditions:
- If the server prefixes all JSON requests with the following string **")]}', \n"**
- Angular will automatically strip the prefix before processing it as JSON.

If the server needs to return the following way:

```
['one' , 'two' ]
```

Vulnerable attack your server it can return the following way:

```
)]} ' ,
['one' , 'two' ]
```


Strict Contextual Escaping:

Strict contextual Escaping(SCE) is a mode in which AngularJS requires bindings in certain contexts to result in a value that is marked as safe to use for that context.

Example: context is binding arbitrary html controlled by the user through ng-bind-html directive.

ngBindHtml directive will not extract content that is not marked as safe by using **$sce** provider.

ngSanitize module can be used to clean such user provided content and mark the content as safe.

Using Local Caches:

Browser store the data in several places and the browsers itself offers localStorage and sessionStorage objects for caching data. But Angular there are objects created by the $cacheFactory. These objects($templateCache) are used to store and retrieve data, mainly used by $http and the script directive to cache templates and other data.

