1. **Cross-Site Request Forgery**

1.1 Definition

- Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated. CSRF attacks exploit the trust a Web application has in an authenticated user.
- A CSRF attack exploits a vulnerability in a Web application if it cannot differentiate between a request generated by an individual user and a request generated by a user without their consent.

1.2 Problem Statement

- Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated. CSRF attacks exploit the trust a Web application has in an authenticated user.
- **Problems:**

  - Submitting or deleting a record.

  - Submitting a transaction.

  - Purchasing a product.

  - Changing a password.

  - Sending a message.

1.3 Available Solutions

- To defeat a CSRF attack, applications need a way to determine if the HTTP request is legitimately generated via the application's user interface. The best way to achieve this is through a CSRF token. A CSRF token is a secure random token (e.g., synchronizer token or challenge token) that is used to prevent CSRF attacks. The token needs to be unique per user session and should be of large random value to make it difficult to guess.
- A CSRF secure application assigns a unique CSRF token for every user session. These tokens are inserted within hidden parameters of HTML forms related to critical server-side operations. They are then sent to client browsers.
- It is the application team's responsibility to identify which server-side operations are sensitive in nature. The CSRF tokens must be a part of the HTML form—not stored in session cookies. The easiest way to add a non-predictable parameter is to use a secure hash function (e.g., SHA-2) to hash the user's session ID. To ensure randomness, the tokens must be generated by a cryptographically secure random number generator.
- Whenever a user invokes these critical operations, a request generated by the browser must include the associated CSRF token. This will be used by the application server to verify the legitimacy of the end-user request. The application server rejects the request if the CSRF token fails to match the test.

1.4 Alternative Solutions

1.4.1 Defense Techniques

- SameSite Cookie Attribute
- Verifying Origin With Standard Headers
- Use of Custom Request Headers
- User Interaction Based CSRF Defense

1.4.2 Defense Techniques Steps

Step 1: Train and maintain awareness

Step 2: Assess the risk

Step 3: Use anti-CSRF tokens

Step 4: Use SameSite cookies

Step 5: Scan regularly

1.5 Pros & Cons of CSRF

1.5.1 Profits of CSRF

- If you implement CSRF tokens as random values stored persistently
- If you implement CSRF tokens as a signature (typically: HMAC) over the session assertion then you're validating that token by checking the signature with your application's secret key, so you don't need to store the token at the server side at all.
- The usual advantage of the signature method is that you don't need storage, so it can potentially be part of a session-less access control scheme.
- Generating a token per session increases useability, as doing it per request whilst ensuring a higher level of security

1.5.2 Consequences of CSRF

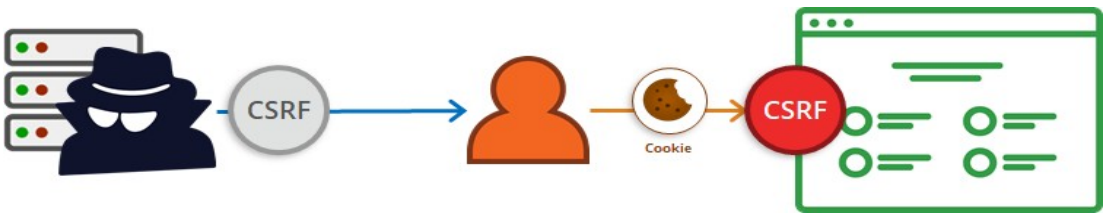
- All forms must output the hidden field in HTML.
- Any AJAX POSTs must also include the value.
- The page must know in advance that it requires the CSRF token so it can include it in the page content so all pages must contain the token value somewhere, which could make it time consuming to implement for a large site.

1.6 Best/Optimal Solution

“The most popular method to prevent Cross-site Request Forgery is to use a challenge CSRF token that is associated with a particular user and that is sent as a hidden value in every state-changing form in the web app.”

1.7 Small Demo

- There are two main parts to executing a Cross-site Request Forgery attack. The first one is tricking the victim into clicking a link or loading a page. This is normally done through social engineering and malicious links.
- The second part is sending a crafted, legitimate-looking request from the victim's browser to the website. The request is sent with values chosen by the attacker including any cookies that the victim has associated with that website.
- This way, the website knows that this victim can perform certain actions on the website. Any request sent with these HTTP credentials or cookies will be considered legitimate, even though the victim would be sending the request on the attacker's command.
- When a request is made to a website, the victim's browser checks if it has any cookies that are associated with the origin of that website and that need to be sent with the HTTP request. If so, these cookies are included in all requests sent to this website.
- The cookie value typically contains authentication data and such cookies represent the user's session. This is done to provide the user with a seamless experience, so they are not required to authenticate again for every page that they visit. If the website approves of the session cookie and considers the user session still valid, an attacker may use CSRF to send requests as if the victim was sending them.
- The website is unable to distinguish between requests being sent by the attacker and those sent by the victim since requests are always being sent from the victim's browser with their own cookie. A CSRF attack simply takes advantage of the fact that the browser sends the cookie to the website automatically with each request.
- Cross-site Request Forgery will only be effective if a victim is authenticated. This means that the victim must be logged in for the attack to succeed. Since CSRF attacks are used to bypass the authentication process, there may be some elements that are not affected by these attacks even though they are not protected against them, such as publicly accessible content.
- For example, a public contact form on a website is safe from CSRF. Such HTML forms do not require the victim to have any privileges for form submission. CSRF only applies to situations where a victim is able to perform actions that are not accessible to everyone.



2. Cross-Origin Resource Sharing

2.1 Definition

- Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources.
- CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request.
- In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

2.2 Problem Statement

- Many modern websites use CORS to allow access from subdomains and trusted third parties. Their implementation of CORS may contain mistakes or be overly lenient to ensure that everything works, and this can result in exploitable vulnerabilities.
    - Security risk.
    - Attacker can use this misconfiguration to get any secret data.
    - Website can obtain null origin using a sandboxed iframe.
    - attacker can register this domain and can steal your data.

2.3 Available Solutions

- **Use the proxy setting in Create App**

  This is a really simple solution which might not work with more complicated situations where multiple API's are involved, or certain types of API authentication is needed.

- **Disable CORS in the browse**

  You can directly disable CORS in the browser. *If you do this, please be aware that you are disabling security restrictions which are there for a reason. I wouldn't recommend browsing the web with CORS disabled; Just disable it whilst developing your website/app.*

- **Use a proxy to avoid CORS errors**

  inally you could use a proxy like cors-anywhere.If you want to easily demo cors-anywhere, Rob — W has setup a public demo which is great for consuming public API's as it requires no registration or config https://cors-anywhere.herokuapp.com. *As it's just a demo requests are limited*.

2.4 Alternative Solutions

- **Enable Chrome to always allow CORS requests**

  A chrome plugin (its name should be *Access-Control-Allow-Origin: **) enables the browser to set `Access-Control-Allow-Origin: *` for every `OPTIONS` request, which means that CORS is enabled everywhere. This is equivalent to the behavior of running Chrome with flags.

- **Add Access-Control-Allow-Origin header**

  Some requests may do a CORS preflight (`OPTIONS`) to get the "approval from the server" (in fact, the true blocking is on the browser). Only when the server allows the current origin to make a request to the backend, the browser will let the request going through.

- **Reverse Proxying the backend APIs to the same domain**

  As we discuss before, the CORS problem only exists when the target origin is different from the current origin, which means that the problem will not exist if we put the frontend and backend on the same origin.

2.5 Pros and Cons of CORS

Pros of CORS

- The CORS can always upgrade and expansion. The new station can be added into the CORS system, that will increase the coverage area, and the software of CORS can always upgrade. Therefore, the CORS is Low prices.
- CORS System is flexibility, security, reliability, stability.
- The reliability of CORS is improved.
- The Network RTK compared with the traditional RTK. The accuracy of Network RTK is improved. In the coverage areas of CORS, the precision of coordinates always 1-2cm (centimeter), it has been limited by the distance of the stations not any more
- The requirement for operation of CORS is reduce.

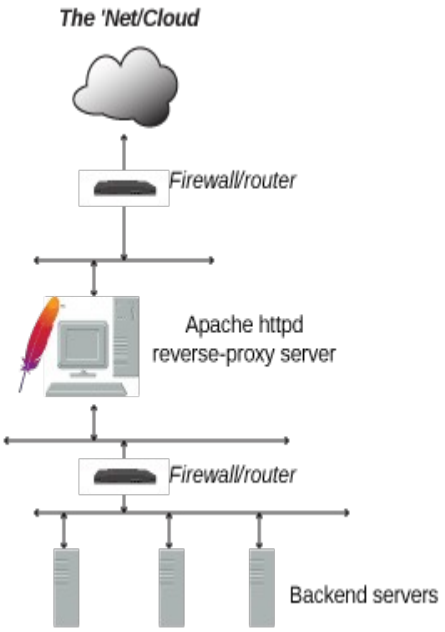
Cons of Cors

- If implemented badly, CORS can lead to major security risk like leaking of API keys, other users data or even much more.
- if a attacker intentionally changes the origin header and gets back ACAO header in response with his website and credentials set to true then the attacker can use this misconfiguration to get any secret data by hosting a html page with simple script on his website.
- If a website allow null origin then also it is exploitable as any website can obtain null origin using a sandboxed iframe.
- If a non existing domain like *some*victim.com allowed access then the attacker can register this domain and can steal your data.

2.6 Best/Optimal Solution

Reverse Proxying the backend APIs to the same domain

- As we discuss before, the CORS problem only exists when the target origin is different from the current origin, which means that the problem will not exist if we put the frontend and backend on the same origin.
- Due to the deployment and administration requirements, we are not able to truely deploy services to the same origin, but we can configure reverse proxy to let them be exposed under same origin.



2.7 Small Demo

- Suppose a user visits http://www.example.com and the page attempts a cross-origin request to fetch the user's data from http://service.example.com. A CORS-compatible browser will attempt to make a cross-origin request to service.example.com as follows.
- The browser sends the GET request with an extra `Origin` HTTP header to service.example.com containing the domain that served the parent page:

  Origin: http://www.example.com

- The server at service.example.com may respond with:

  Access-Control-Allow-Origin: http://www.example.com

- The requested data along with an `Access-Control-Allow-Origin` (ACAO) header with a wildcard indicating that the requests from all domains are allowed:

  Access-Control-Allow-Origin: *

- An error page if the server does not allow a cross-origin request

References Links

Links used in CSRF

1.https://www.synopsys.com/glossary/what-is-csrf.html

2.https://www.youtube.com/watch?v=AHV9ThkRT9w

3.https://www.acunetix.com/websitesecurity/csrf-attacks/

4.https://devnet.kentico.com/articles/protection-against-cross-site-request-forgery-(csrf-xsrf)

5.https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

6.https://stackoverflow.com/questions/20504846/why-is-it-common-to-put-csrf-prevention-tokens-in-cookies

7.https://security.stackexchange.com/questions/51182/advantages-of-multiple-valid-csrf-tokens

Link used in CORS

1.https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

2.https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

3.https://medium.com/swlh/avoiding-cors-errors-on-localhost-in-2020-5a656ed8cefa

4.https://medium.com/@kennch/solutions-to-cross-origin-resource-sharing-cors-limitation-69a928e6297d

5.https://www.section.io/engineering-education/how-to-use-cors-in-nodejs-with-express/

6.https://medium.com/@electra_chong/what-is-cors-what-is-it-used-for-308cafa4df1a#:~:text=%E2%80%9CCORS%E2%80%9D%20stands%20for%20Cross%2D,%2DOrigin%20Policy%20(SOP).

7.http://gmsarnjournal.com/home/wp-content/uploads/2015/08/vol1no2-3.pdf

8.https://portswigger.net/web-security/cors