Cross Site Request Forgery
Web Application Penetration Testing

How Prevalent are Cross-Site Request Forgery Attacks in Web Applications?

THE ABSTRACT

In this research paper, it will discuss the relevancy of a specific attack known as Cross-Site Request Forgery (CSRF) which is known for its severity and critical nature.

It will discuss how and why it is such a pressing vulnerability as well as fixes and solutions put in place to mitigate the instance of this exploitation attempt, alongside, showing examples and performing a demonstration.

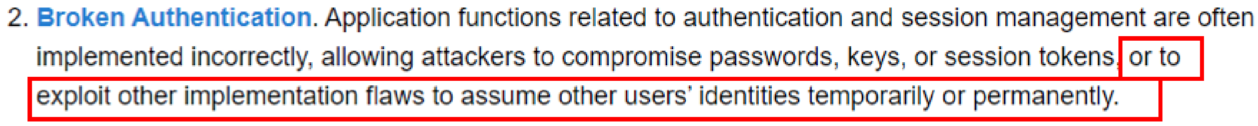
INTRO

Cross Site Request Forgery, more commonly known as 'CSRF' is a security audit attack attackers use in order to force an end user to execute unwanted actions on a web application or altering a request to bypass client-side security.

In other words, it is the manipulation of a HTTP/S request that allows values to be added, deleted, or altered, giving a different response.

An example would be capturing a login request before sending to the server and changing the user from 'John Doe' to 'admin', and removing the password field, allowing a successful login to the admin account instead. This vulnerability isn't always present in every web application.

CSRF vulnerabilities come under the scope of a Web Application Penetration Test (aka. Web App Pen Test or "Website Hacking") and are considered critical or high-risk vulnerabilities. Moreover, CSRF being a big issue made it number two under 'Broken Authentication' on OWASPs top 10 list (a list of the most commonly known vulnerabilities for Web App Pen Tests).



2. **Broken Authentication**. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

Due to the nature of this vulnerability, if found on a bug bounty program the reward will typically be into the thousands.

The reason behind why this is such a prevalent attack is because of its simplicity as well as its underlying complex side which as a result can give attackers access to accounts or change your password without knowing the previous one as well as forging other requests.

MAIN CONTENT

CSRF attacks can be found in various web applications through various user input fields. User input fields is where this vulnerability could lie.

How Would You Test For a CSRF Attack?

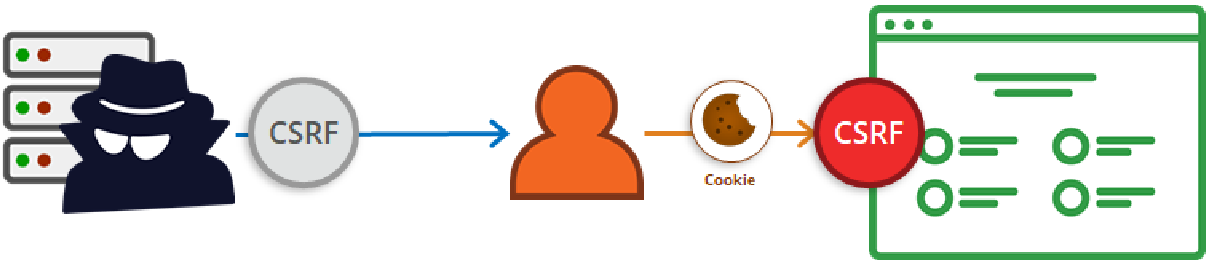
A CSRF attack vulnerability you would have to find and to do so would be as simple as intercepting HTTP/S requests and altering the input fields to see different outcomes as well as studying how requests are made from client-side to server-side. Typically CSRF attacks are common even though they came out a while ago because of the number of input fields that a website could have (username, password, search bar, transaction ID, etc) that they would need to secure.

Most frameworks used for creation and support of websites (web applications) have built-in features to help prevent CSRF exploitations, however, these can often be bypassed or disabled through other types of attacks and some web applications don't enable them entirely when using the frameworks.

The main input fields that could be vulnerable to CSRF attacks are: logins, payments, quantities, buying items, costs, taking out IDs, altering IDs, two step verification, security questions and so on. Any input field could be seen as a potential attack vector for a CSRF attack but some fields are more common than others.

A very good tool used for Web App Pen Testing is Burpsuite, it is a Man in the middle proxy (MITM) that is pre-installed in Kali Linux which is the perfect OS for pen testing due to its functionality and its tools. I have been using for over two years and am now proficient in Burpsuite and the attacks it can entail.

Burp isn't just used for CSRF attacks but allows various attacks on web applications as well as modifying HTTP requests before sending, allowing bypassing client-side validation and authentication (which is what CSRF exploits and bypasses).



CSRF attacks can be executed a number of ways: from the web application alone by forging requests before they are sent to the web server via a HTTP request or by sending forged requests from another domain that the attacker owns (their own server)

For a bank example, a transaction could be made then copied and an automatic payload that once sent to the victim automatically sends money to the attackers bank account.

This means that once the victim clicks on the link automatic JavaScript will be executed, submitting the form and the content of the transaction is then made, if the transaction was to send money to an attackers bank account then this action would then be performed.

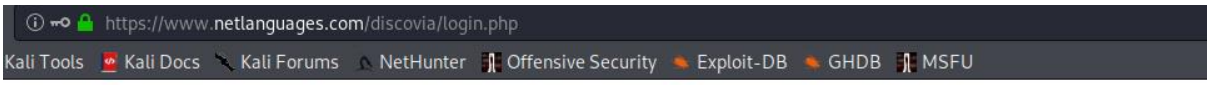
<u>CSRF in practice</u>

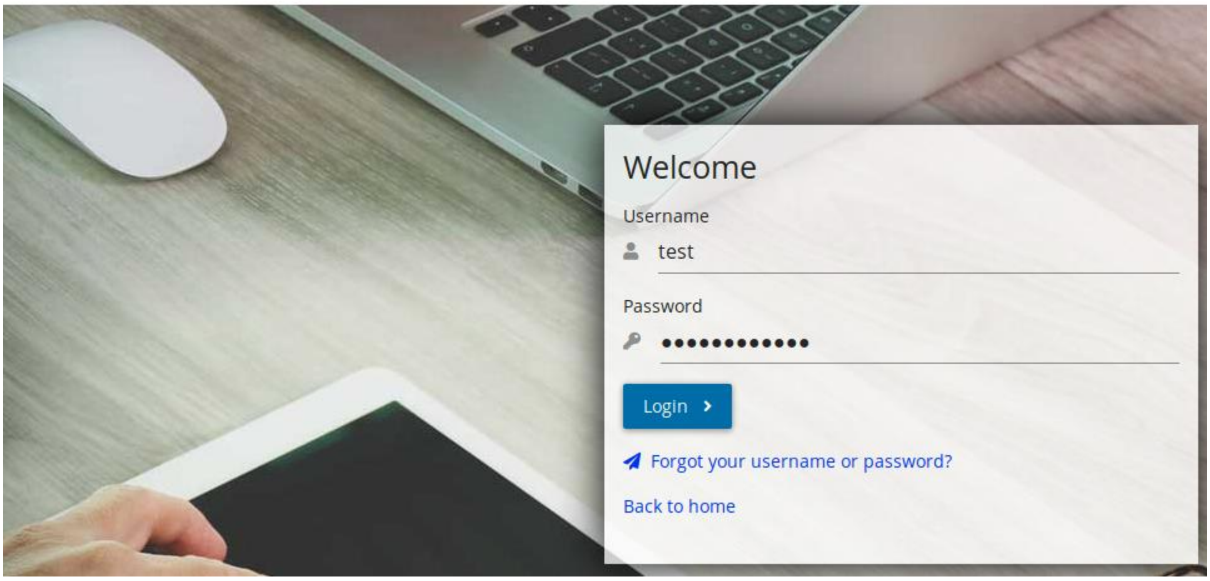
In order to demonstrate the theoretical knowledge I have outlined above; it is imperative that a practical example from myself goes along with it.

Netlanguages.com will be the victim domain for this practice CSRF attack. I found them through the use of Open-Source Intelligence (OSINT) and through the use of Google Dorks, which are specific search terms to return relevant pages.

Through using *inurl="/login.php"* I was returned only websites (domains) that contained a php login page titled "login.php" from the previous directory.

I put some dry data in to see how the login page functions and then captured the request using Burpsuite.


https://www.netlanguages.com/discovia/login.php


Welcome

Username

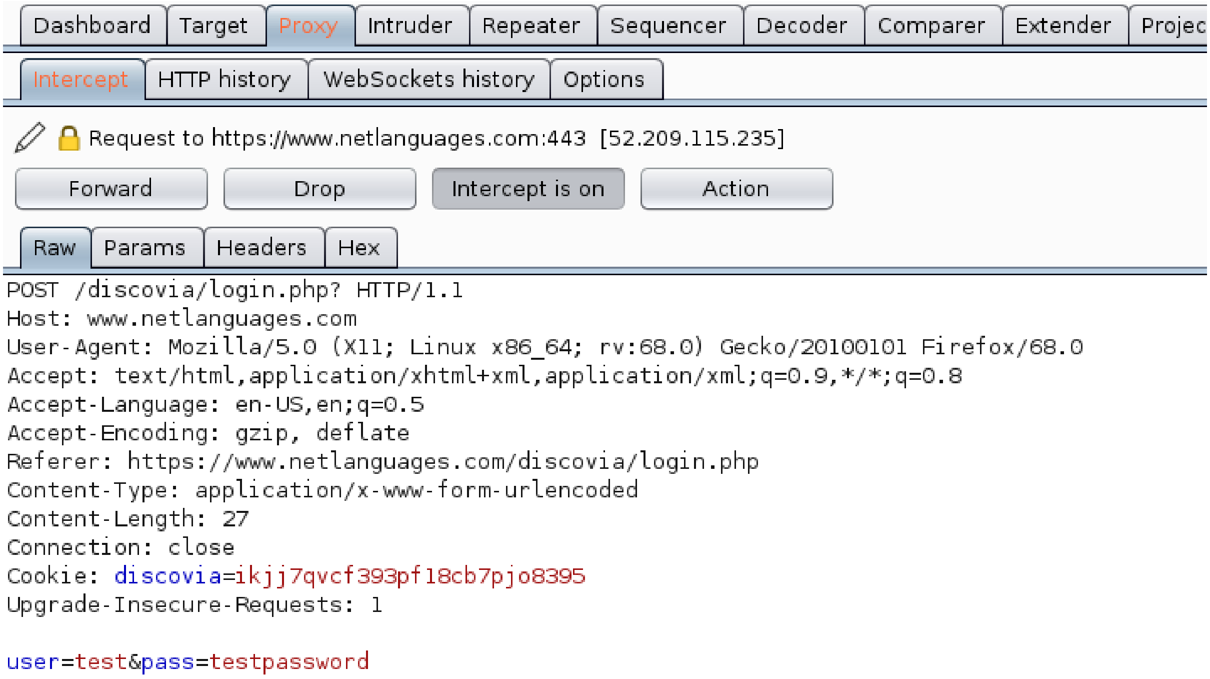
test

Password

•••••••••••••

This is the login page with the username **test** and the password field filled out with **testpassword**, this is simply some test data to see how it operates.

I then intercepted this request with Burpsuite in order to start manipulating the request.



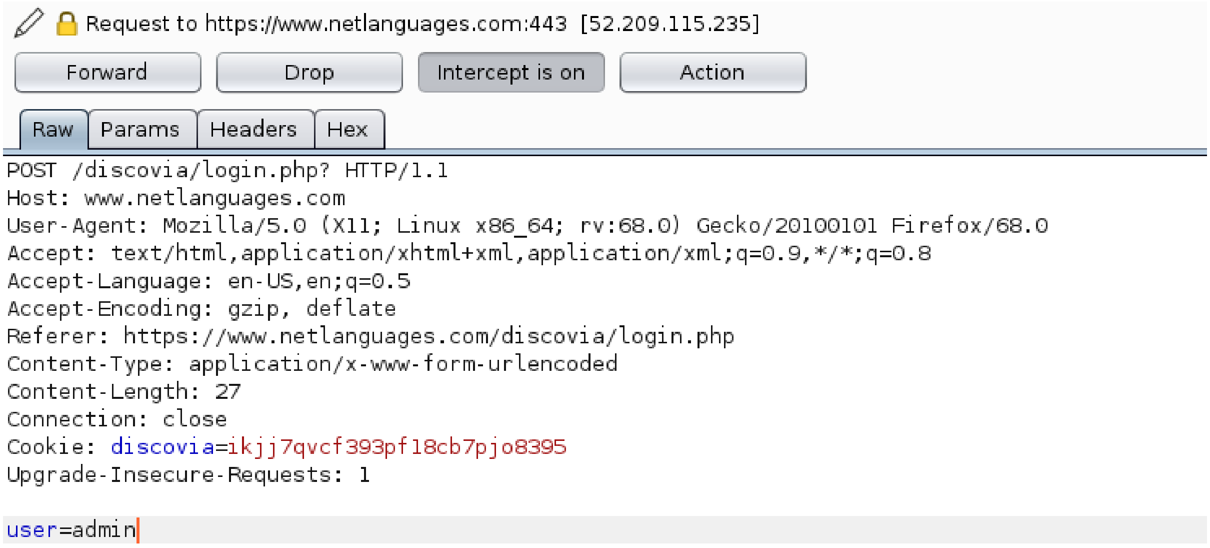
```
POST /discovia/login.php? HTTP/1.1
Host: www.netlanguages.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.netlanguages.com/discovia/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Connection: close
Cookie: discovia=ikjj7qvcf393pf18cb7pjo8395
Upgrade-Insecure-Requests: 1

user=test&pass=testpassword
```

This is Burpsuite Intercept tab which is one of the most used sections and tabs. At the top you can see it is making a POST request to the /login.php over HTTP (thus insecure protocol), the host underneath is the victim domain.

At the bottom you see the input fields called *parameters*. These store the test data we entered into variables. We can manipulate these parameters to suite our motives.

CSRF attacks bypass client-side validation and authentication aka. Browser security. It does not interact or try to bypass server-side security thus if server-side security read each response the server would see that the client-side request does not match the one trying to get to the server thus wouldn't be successful.

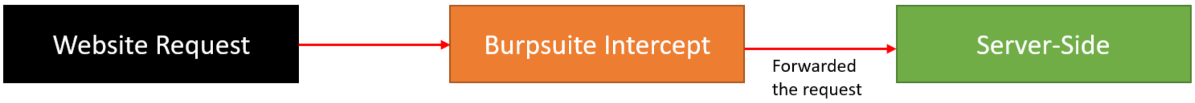
Furthermore, by altering the user parameter and deleting the password field, this can allow attackers in as another user without needing the password in some vulnerable web applications.


Request to https://www.netlanguages.com:443 [52.209.115.235]
Forward
Drop
Intercept is on
Action
Raw
Params
Headers
Hex

```
POST /discovia/login.php? HTTP/1.1
Host: www.netlanguages.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.netlanguages.com/discovia/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 27
Connection: close
Cookie: discovia=ikjj7qvcf393pf18cb7pjo8395
Upgrade-Insecure-Requests: 1

user=admin
```

Once this packet/request was forwarded it allowed access to the administrator account of the site, making a critical issue for web developers. By taking out the password field it has nothing to compare to and as a result ignores the password comparison.

Using the repeater tab in Burpsuite can allow you to see the page responses in the same window, on the left would be the request and the right would show the response of the page, allowing attackers to see what works more efficiently without having to reload the webpage each time for a new request.



CSRF attacks can work through GET and POST methods for HTTP/S requests and not just for one of them, which means securing the entire web application for any GET or POST request.

GET & POST Requests

A GET request is one that receives information back from the server and displays it to the end user in the browser webpage, typically when you request a page it sends a GET request to grab the contents of that page and display it.

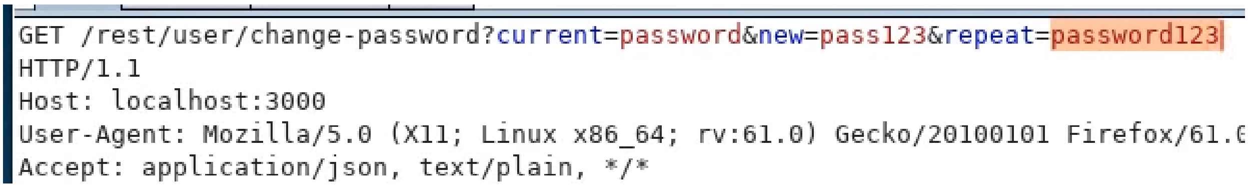
A POST request method is one where information is being updated, altered, or written to the web server or database, an example would be creating a new user or updating a password.

Both are valid methods to use for HTTP and HTTPS websites as you need to query and be able to update information on specific websites. Some developers think making all the requests use the POST method will minimise the risk of such attacks, but in actual fact it doesn't unfortunately due to CSRF and similar attacks being also performed on both methods.

CSRF Attacks on Updating Passwords

Passwords can be reset by an attacker if a CSRF vulnerability is present in the change password field. By going to the "change password" page you have to generally type the current password then the new password twice.

CSRF attacks allow you to intercept the request in Burpsuite then remove the current password field leaving only the two new password entry fields. If a CSRF attack is exploitable here it could simply reset the users password without knowing the current.



```
GET /rest/user/change-password?current=password&new=pass123&repeat=password123
HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json, text/plain, */*
```

End of Main Section Thoughts

There is a variety of attacks for CSRF and each one can me independently unique, some can be uses of redirects and some can be to bypass the domain security to login as another user. Because of this variety, this is what makes it a critical vulnerability due to the nature of how volatile this type of attack can be as it comes under "broken authentication".

Some Cross-Site Request Forgery attacks can be integrated with other attacks to make them more powerful, for example:

An attack of CSRF to change the UID from 233 (normal user) to 1 (which could be the admin UID) would allow the attacker to login to the web application as the administrator giving them access to everything on the web server with endless permissions and privileges, the attacker then changes some settings so commands can be executed, this allows the vulnerability of **Remote Code Execution** which is another critical vulnerability. Therefore, many critical vulnerability attacks can be chained together to perform devastating results and open up to more attack vectors.

FIXES AND SOLUTIONS

CSRF Tokens

These are randomised strings of characters and numbers that are unique to each request, making it almost impossible to forge a request without the same token. This can help stop forgeries cross domain and also in general.

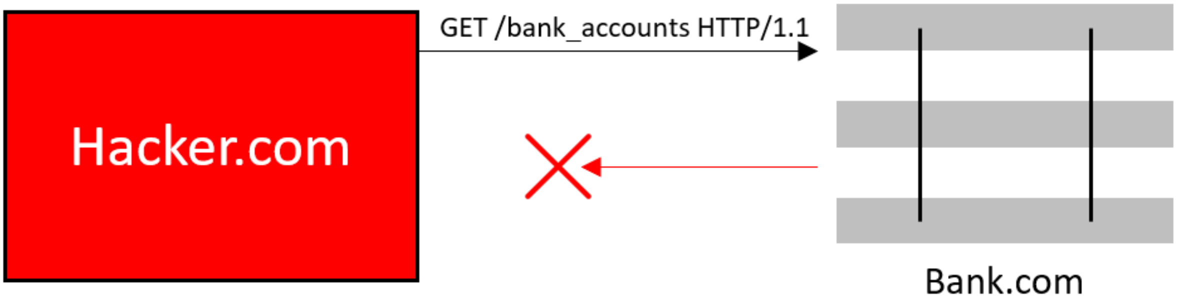
Server-side Verification & authentication

There is two types of verification/validation and authentication. You have **client-side** and **server-side**, CSRF attacks are to bypass client-side security, aka the browser and websites security on local machine, it cannot bypass server-side security as on the server it runs checks as well as compares the client-side request to that of the one trying to interact with the server, if they are not the same the request/packets are dropped and the attack is unsuccessful.

Same origin policy (SOP)

A way to protect a web application from an attacker from a different domain is to enforce the same origin policy, it prevents one domain not listed from accessing information from another domain e.g. Prevents Hacker.com from accessing information from bank.com.

It checks the host (domain) as well as the origin of the request, companies whitelist origins that can access domain information e.g. the subdomain "dev.bank.com" can access "bank.com" but "hacker.com" cannot.

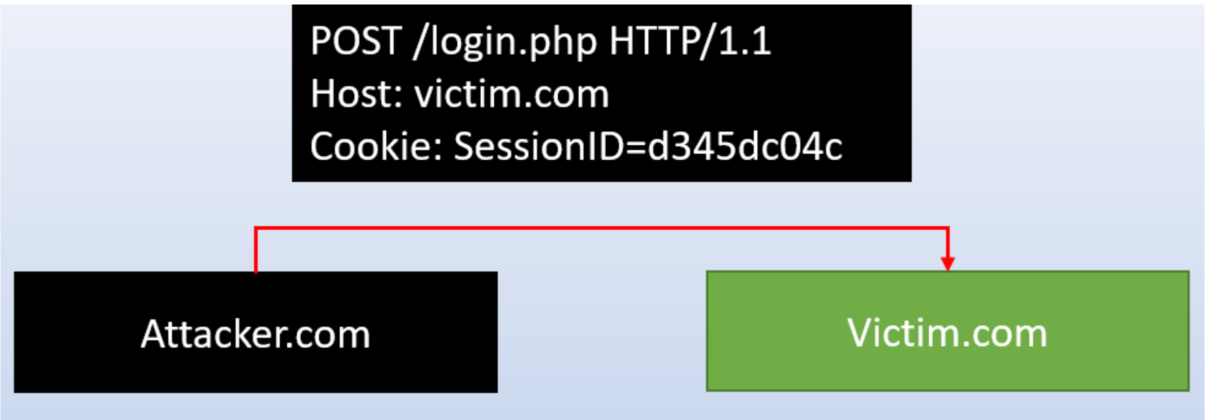


Host: Bank.com
Origin: Hacker.com ✗

Not from the same origin. This attack is called **cross origin.**

Few ways Developers relaxed SOP:

- JSONP (padding) – wrapping a function around JSON data before its returned, prevents API calling and cross-origin reading/writing of information.
- Allow similar subdomains to access the main domain (dev.bank.com -> bank.com) for reading and writing.



```
POST /login.php HTTP/1.1
Host: victim.com
Cookie: SessionID=d345dc04c
```

Another example of SOP is to stop attackers making POST requests to a different website from their web server. As they aren't the same hosts.

Same origin policies aren't fully protective but helps minimise the risk of an attacker forging requests to the victim web application from a different domain.

Other Protections and Security Tools

As there are many programming languages used to make web applications, different ones have different security implementations in order to be compatible.

Java applications can use **OWASP CSRF GUARD** to protect against CSRF attacks on their domains and web applications.

PHP is a very common programming language for the creation of web applications and thus requires security to mitigate these risks too. **CSRFProtector** is a project developed to specifically prevent CSRF attacks being launched in order to secure the web applications from an OWASP top 10 vulnerability and keep its users safe.

SUMMARY/ WHAT I HAVE LEARNT

Before starting this research project/white paper I was aware CSRF attacks were an attack and knew how they worked as well and practiced a few on OWASP juice shop (a purposefully vulnerable web application in order to conduct ethical attacks to get experienced with web app pen tests), but I was able to dive in deeper for this project and learnt a great deal more than before in which allowed me to increase my knowledge and skills around experimenting with these types of attacks ready for the industry.

Because of this research paper I will be able to quickly identify potential attack vectors through user input fields for possible Cross-Site Request Forgery attacks in the future, allowing me to level up faster and save me time, making me more efficient at spotting these possible attacks.

RESOURCES USED

https://owasp.org/www-community/attacks/csrf

https://owasp.org/www-project-top-ten/

https://owasp.org/www-project-web-security-testing-guide/

https://owasp.org/www-project-code-review-guide/reviewing-code-for-csrf-issues

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

https://auth0.com/blog/cross-site-request-forgery-csrf/

https://wiki.owasp.org/download/jmanico/owasp_podcast_69.mp3

https://www.youtube.com/watch?v=eWEgUcHPle0

https://www.youtube.com/watch?v=vRBihr41JTo

https://www.youtube.com/watch?v=Ub5TLow9GL4