

# Cross Site Request Forgeries (CSRF)

Overview .....	1
What is CSRF? .....	1
How to protect against CSRF .....	2
Tracking user flow .....	3
Nonce Checking .....	4
CAPTCHA .....	4
IP Address based Rate Limiting .....	5
Referrer Checking .....	5
Conclusion .....	5

## Overview

This technical document is intended to provide a detailed overview of the causes, impact, and mitigation of Cross Site Request Forgery (CSRF) attacks. The information presented here assumes that the reader possesses a solid understanding of web application development frameworks, session management, and web browser functionality.

### ***What is CSRF?***

CSRF is not a new class of attack but it is gaining popularity with the increasing availability of web based interfaces for managing personal information and critical application functionality. Unlike most application vulnerability classes which attempt to leverage weaknesses in application code, CSRF attacks leverage existing web application functionality by hijacking authenticated sessions and providing unauthorized user input.

To further explain how CSRF attacks work, the following example will demonstrate potential abuse against the send Email function of a modern webmail application.

As with typical webmail systems, an authenticated user can specify email recipients, a subject line, some email body text, and optionally, upload attachments.

Once all the information has been entered, the authenticated user clicks the "Send" button which generates the following HTTP GET request using the browser's XMLHttpRequest object and sends the email.

```
GET /sendmail?mailfrom=user@webmail.com&mailto=chris@webmail.com&subj=hi%20chris%21&mailbody=it%27s%20been%20a%20while.%20hope%20things%20are%20well%21%20hope%20you%20make%20it%20to%20the%20beach%20this%20summer.&attach=
```

Cookie:

```
R2xhZCB0byBzZWUgeW91XCdyZSBrZWVulGV5ZSBmb3lgc3BvdHRpbmcyYmFzZTY0IGVuY29kZ  
WQgc3RyaW5ncyBpcyBvbiEgOik%3D
```

In order to abuse this functionality, an attacker would simply change the values of the to:, subject, and body fields then force an authenticated user's browser to request the URL and could then SPAM thousands of victims in seconds.

In order to be successful, CSRF attacks must be able to leverage an existing, authenticated session in a victim's browser. Considering the prevalence of cross site scripting and open-redirect issues out in the wild, and the various social engineering vectors, there is no shortage of techniques to trick browsers and victims to visit unwanted pages.

While the example of the send email function in a webmail application is rather simple and innocuous, consider the other types of web applications you've used and the damage a CSRF vulnerability could cause there:

- Online banking / Online brokerage accounts - Financial loss.
- DMV/Government records - Identity theft.
- Administrative GUIs for hardware - Unauthorized configuration changes.
- Critical business tools - Business confidential information leaks.

## How to protect against CSRF

To protect your application against CSRF the project team must document the application functionality and classify the data processed according to sensitivity. The intent of this exercise is to identify the portions of the application which require additional protection against CSRF attacks due to the sensitive task they perform or the data which they process. The following identify common web application functionality and whether they would typically require additional CSRF protections:

- Profile update functionality - yes, this application functionality should require additional CSRF mitigations due to the task it performs.
- Send to a friend functionality - yes, this type of application functionality is typically abused by SPAMMERS.
- "Favorite" a radio station - no, this functionality isn't critical and the data processed is not confidential.
- Search functionality - no, this functionality isn't critical and the data processed is not confidential.

## ***Tracking user flow***

For application functionality requiring additional protections against CSRF attacks, the project team must document the steps which a legitimate user would follow for the application functionality (functionality business flow). Additionally, the expected data elements for each step in the business flow should be included in the documentation (functionality data flow). The following is an example of the business and data flows for a simple registration system:

- **Step 1** - The user enters their desired username, password and existing email address.  
**Data elements:** username (required, varchar, max length=20), password (required, varchar, complexity rules applied), and email address (required, varchar, valid email address format)  
**Note:** If the desired username is not available, this step will repeat.
- **Step 2** - The user enters their profile information.  
**Data elements:** first name (required, varchar, max length=40), last name (required, varchar, max length=40), URL (optional, varchar, valid URL), City (optional, varchar, max length=40), State (required, valid 2 letter state), Date of birth (optional, valid date).  
**Note:** if required information is not provided this step will repeat.
- **Step 3** - The user provides their privacy preferences.  
**Data elements:** Yes/No answers for the following questions:
  - Show my name?
  - Show my date of birth?
  - Show my location?
  - Allow other users to contact me?
  - Show my email address?
- **Step 4** - User receives an email containing a link with an activation code.  
**Data elements:** activation code (random varchar, minimum length=16)
- **Step 5** - User clicks the activation link.

Once the business and data flow models have been created for the critical application functionality, planning for CSRF mitigations can begin. For multi-step functionality, like the registration flow in the previous example, the most efficient mitigation is to perform application level enforcement of the business and data flows and nonce checking.

Application level enforcement of the business flow requires the development team to build checks into the application which enforce the business rules for the given functionality. Using the previous registration example, the business rules would be:

- Each step must be performed sequentially.  
To ensure that each step is performed sequentially, the anti-CSRF mechanism can use persistent or session level storage keyed off the unique session identifier to write when a step has been completed

successfully. The mechanism can then read the steps completed before honoring requests for subsequent steps.

- Each required answer must be provided before the next step can be accessed.  
The anti-CSRF mechanism must check that all required data has been provided in the correct format before determining that a step has been completed.
- Data provided in previous steps cannot be changed once the step has been submitted.  
The anti-CSRF mechanism must also be able to validate that information from a previous step cannot be changed once the step has been deemed complete.
- The minimum time allowed to complete the steps 1,2 and 3 is 10 seconds per step.  
The anti-CSRF mechanism must be able to determine how much time the user spent providing the required information. This time frame is determined by the difference between the time the page was served to the browser and the time when the required data is returned to the server. The approach will work with the other mitigations to keep scripts from performing mass automated data input.

## ***Nonce Checking***

A nonce is defined as an alpha-numerical variable which is "hidden" in an HTML form and submitted as part of the data elements of the web application function. Because this variable is known by the application prior to serving it to the browser, it can then be compared when the data elements are received and differences can be discarded. Please note that this solution cannot be trusted to prevent CSRF attacks unless it is coupled with other mitigations described in this document.

## ***CAPTCHA***

The CAPTCHA technology attempts to leverage a human's ability to recognize alpha-numeric characters when heavily distorted. There are certain key features which will increase the effectiveness of a CAPTCHA solution. These features should be configurable and include:

- Number of characters: this setting will determine how many random alpha-numeric characters to display for each CAPTCHA. It is recommended that the image include 8 or more characters.
- Number of different fonts: this setting will determine how many different fonts will be used when generating the text for the CAPTCHA.
- Distortion level: this setting will determine the level at which the text is twisted or warped. The more twisted and warped, the more difficult for Optical Character Recognition (OCR) technology to decipher.

- Noise level: this setting will determine how much additional noise (random dots and lines) will be added to the CAPTCHA image.
- Background image: this setting will determine whether to overlay the CAPTCHA on top of an image to decrease the ability for OCR to determine the text.

By being able to configure these options by level, you give your CAPTCHA the ability to be turned up (more complex CAPTCHAs) in the event of signs of abuse by automated scripts. Please note that this solution cannot be trusted to prevent CSRF attacks unless it is coupled with other mitigations described in this document.

### ***IP Address based Rate Limiting***

IP address based rate limiting is a fairly simple concept with many variations. This mitigation entails limiting the access to the web application based on the source IP address of the transaction. Before implementing this mitigation, special consideration should be given to the following topics:

- What will be the threshold after which the IP address will be blocked?
- Should the application block or throttle IP addresses which have passed the defined rates?
- How long should an IP address be considered rate limited?

Once rate limiting rules have been established, a determination must be made whether to implement the rate limiting in the application code, or some other network device. Please note that this solution cannot be trusted to prevent CSRF attacks unless it is coupled with other mitigations described in this document.

### ***Referrer Checking***

A standard HTTP request from a browser will include the URL of the page from which the user came as the Referrer HTTP header. This URL can be used to validate that the user is following the business flow of the application functionality by checking that the Referrer submitted with the request is that of the previous step. Please note that referrer checking alone cannot be considered as a mitigation for CSRF as this header can be modified by a crafty attacker.

## **Conclusion**

CSRF attacks can also be coupled with other web based vulnerabilities such as Cross Site Scripting (XSS). An attacker can leverage an XSS vulnerability to perform the CSRF using Javascript where the user will never see evidence of an attack. Because CSRF attacks abuse intended web application functionality, it's important for a project design team to identify areas of functionality which might be attractive to attackers.