Introduction to CSRF

Charithra Kariyawasam Follow
Sep 28, 2017 · 6 min read

Introduction

CSRF is a type of attack which tricks the victim to do the malicious task on a victim authenticated webapplication on behalf of attackers interests. The level of the attack is based upon the level of privileges that the victim possessed. Because attacker will use the authentication that has gained in the current session to do the malicious task. This is the reason why this attack termed as Session Riding too. CSRF attack will exploit the concept that if the user is authenticated all the requests that come from that user must be originated by the user. The attacker will exploit this concept by identifying the session cookie of the session and use that to send his own payload to run on the application.

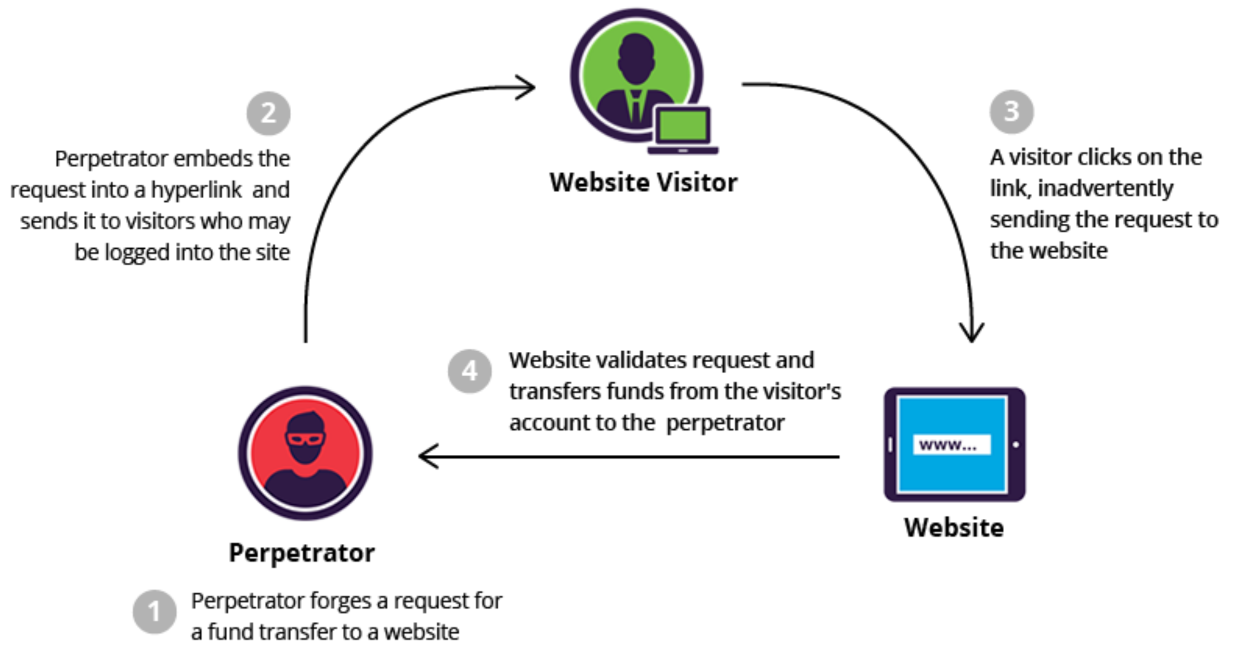
How CSRF works?

CSRF will only work if the potential victim is authenticated.Using a CSRF attack an attacker can bypass the authentication process to enter a web application. When a victim with additional privileges performs actions that are not accessible to everyone, which is when CSRF attacks are utilized. Such as online banking scenarios.

There are two main parts to execute a Cross-Site Request Forgery (CSRF) attack

1) The first part is to trick the victim into clicking a link or loading up a page. This is normally done through social engineering. By using social engineering methods attacker will lure the user to click the link.

2)The second part is to send a “forged” or made up request to the victim’s browser. This link will send a legitimate-looking request to the web application. The request will be sent with the values that the attacker wants. Apart from them, this request will include any cookies that the victim has associated with that website.


2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
Website Visitor
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

CSRF scenario

When a request is made to a web application, the browser will check if it has any Cookies. If the relevant cookies are available, those will need to be sent with the request. This is done to remember users interactions with the web site as HTTP is stateless. So they would not be required to re-authenticate for every page that they visit. If the website approves of the Cookie being sent and considers the session as still being valid, an attacker may use CSRF to send requests as if the victim is sending them.

As cookies are sent, the web application knows that this victim can perform certain actions on the website based upon the authorization level of the victim. Web application will consider these requests as original. But in reality, the victim would be sending the request on the attacker's command.A CSRF attack simply takes advantage of the fact that the browser sends the Cookie to the web application automatically with each and every request.

As we have a brief understanding about CSRF, let's analyze the following scenarios

Example of a CSRF attack using a GET request

When the malicious link is clicked, the attacker may direct the victim to their own malicious web application that will execute a script that will in turn trigger the victim's browser to send an illegal request. This request is defined as illegal since the victim is not aware that it is being sent. But it appears to the web server as if the user sent it. Because it includes the necessary Cookies that the web server needs to verify that a victim is who they say they are.

The below example shows a legitimate URL, which will request that the web application transfers a 100,000 units of the appropriate currency to User's account.

```
http://example.com/transfer?amount=1000000&account=User
```

The request will include with it the Cookie for the authenticated user, so there would be no need to define which account the money will be transferred from. This means that if a normal user would access this URL, they would be required to authenticate, so the web application will know from which account the funds will be withdrawn. Now that we know how this request can be used for legitimate reasons, we can figure out a way how to trick a victim into sending the request that the attacker wants, while authenticated as the victim .

If the web application being exploited is expecting a GET request, then the attacker can include an tag on their own website, that instead of linking to an image, it will send a request to the bank's web application:

```
<img src="http://example.com/transfer?amount=1000000&
account=Fred" />
```

The browser, under normal circumstances, will automatically send the Cookies that are related to that web application, therefore allowing the victim to perform a state change on behalf of the attacker, where the state change is a transfer of funds.

CSRF attack using a POST request

```
<h1>You Are a Winner!</h1>
```

```
  <form action=”http://example.com/api/account" method=”post”>
  <input type=”hidden” name=”Transaction” value=”withdraw” />
  <input type=”hidden” name=”Amount” value=”1000000" />
  <input type=”submit” value=”Click Me”/>
 </form>
```

1. A user logs into www.example.com, using forms authentication.
2. The server authenticates the user. The response from the server includes an authentication cookie.
3. Without logging out, the user visits a malicious web site. This malicious site contains the following HTML form. (The form action posts to the vulnerable site, not to the malicious site. This is the “cross-site” part of CSRF.)
4. The user clicks the submit button. The browser includes the authentication cookie with the request.
5. The request runs on the server with the user’s authentication context, and can do anything that an authenticated user is allowed to do.

Although this example requires the user to click the form button, the malicious page could just as easily run a script that submits the form automatically. Moreover, using SSL does not prevent a CSRF attack, because the malicious site can send an “https://” request.Typically, CSRF attacks are possible against web sites that use cookies for authentication, because browsers send all relevant cookies to the destination web site.

Preventing CSRF vulnerabilities

Although there have been a variety of proposed CSRF prevention mechanisms, not all of them are effective in all scenarios. The following implementations prove to be effective for a variety of web applications.

Anti-CSRF Tokens

The most popular implementation to prevent Cross-site Request Forgery (CSRF), is to make use of a token that is associated with a particular user and can be found as a hidden value in every state changing form which is present on the web application. This token, called a *CSRF Token* or a *Synchronizer Token*, works as follows:

1. The client requests an HTML page that contains a form.
2. The server includes two tokens in the response. One token is sent as a cookie. The other is placed in a hidden form field. The tokens are generated randomly so that an adversary cannot guess the values.
3. When the client submits the form, it must send both tokens back to the server. The client sends the cookie token as a cookie, and it sends the form token inside the form data. (A browser client automatically does this when the user submits the form.)
4. If a request does not include both tokens, the server disallows the request.

This protects the form against CSRF attacks, because an attacker forging a request will also need to guess the anti-CSRF token. Unless they won't successfully trick a victim into sending a valid request. This token should be invalidated after some time and after the user logs out. Anti-forgery tokens work because the malicious page cannot read the user's tokens, due to same-origin policy.

Same Site Cookies

CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular origin, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, tuning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent. The Cookie and the page from where the request is being made, are considered to have the same origin if the protocol, port (if applicable) and host is the same for both.

A current limitation of same-site Cookies is that not all modern browsers support them, while older browsers do not support web applications that make use of same-site Cookies.

Summary

Cookies are vulnerable as they are automatically sent with each request, allowing attackers to easily craft malicious requests leading to CSRF. Although the attacker cannot obtain the response body or the Cookie itself, the attacker can perform actions with the victim's elevated rights. The impact of a CSRF vulnerability is also related to the privilege of the victim, whose Cookie is being sent with the attacker's request. While data retrieval is not the main scope of a CSRF attack, state changes will surely have an adverse effect on the web application being exploited.