**CSRF**

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker’s choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

For most sites, browser requests automatically include any credentials associated with the site, such as the user’s session cookie, IP address, Windows domain credentials, and so forth. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish between the forged request sent by the victim and a legitimate request sent by the victim.

e.g.- If the application was designed to primarily use **GET** requests to transfer parameters and execute actions, the money transfer operation might be reduced to a request like:

GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1

Maria takes the original command URL and replaces the beneficiary name with herself, raising the transfer amount significantly at the same time:

http://bank.com/transfer.do?acct=MARIA&amount=100000

The [social engineering](https://en.wikipedia.org/wiki/Social_engineering_(security)) aspect of the attack tricks Alice into loading this URL when Alice is logged into the bank application. This is usually done with one of the following techniques:

* sending an unsolicited email with HTML content
* planting an exploit URL or script on pages that are likely to be visited by the victim while they are also doing online banking

The exploit URL can be disguised as an ordinary link, encouraging the victim to click it:

<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>

Or as a 0x0 fake image:

<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000" width="0" height="0" border="0">

**POST**

Let’s assume the bank now uses POST and the vulnerable request looks like this:

POST http://bank.com/transfer.do HTTP/1.1

acct=BOB&amount=100

Such a request cannot be delivered using standard A or IMG tags, but can be delivered using a FORM tags:

<form action="http://bank.com/transfer.do" method="POST">

<input type="hidden" name="acct" value="MARIA"/>

<input type="hidden" name="amount" value="100000"/>

<input type="submit" value="View my pictures"/>

</form>

This form will require the user to click on the submit button, but this can be also executed automatically using JavaScript:

<body onload="document.forms[0].submit()">