Introduction to Software Security

Chapter 3.9.3: Cross Site Request Forgery Attacks (CSRF)

Elisa Heymann
elisa@cs.wisc.edu

Barton P. Miller
bart@cs.wisc.edu

DRAFT — Revision 0.1, June 2023.

Objectives

- Understand the causes of cross site request forgery attacks.
- Learn about the risks associated with cross site forgery.
- Learn about the damage that can result from cross site request forgery.
- Motivate the need for preventing cross site request forgery with the use of proper session management (Chapter 3.9.4).

Background

How often does your browser make web requests with a URL that you did not intend or understand? The answer is: quite frequently. This situation can occur in many ways, both legitimate and malicious, including:

- Mistyping a URL.
- Clicking on a link from an untrusted web page or email message (social engineering attack).
- Reading a web page that contains a URL that makes a request to another web page, such as a URL in the `SRC` field of an `<IMG>` tag.

When you combine such an inadvertent URL request with a currently active web session, say with your bank, there is a possibility that an attacker might cause you to initiate a web transaction that you did not intend. Your browser cannot distinguish between a URL request that you intended to make and one that was inadvertent. In both cases, all the cookies that are associated with the website will be sent to the server. These cookies might include information about your identity and, more worrisome, the ID of your active (already authenticated) session with that server.

The inadvertent requests could cause you to execute a bank transfer, make an online purchase, or post to social media. The result of such a request could be loss of money or damage to your reputation (or that of your organization).

Cross site request forgery attacks have had significant impact, affecting many organizations, including TikTok[1], McAfee[2], and Google Chrome[3]. As of the most recent OWASP survey of top CWE's, CSRF is still at position #9 on the list[4], and this position is in spite of wide use of modern web frameworks that are designed to prevent such attacks.

If the web application (server) that generates the HTML does not carefully sanitize the text by neutralizing or preventing the presence of special characters such as "<" and ">", then you are at risk of receiving potentially dangerous HTML. Note that we are most dependent on the server protecting us from cross site scripting, so there is little that a user can do to prevent such attacks.

An Example CSRF Attack

Suppose you were logged in and had an active session with your bank's website, `bank.com`. And also suppose that you were tricked into using a malicious URL in one of the ways mentioned above. This malicious URL might have been contained within an `<IMG>` tag in a web page and look something like:

```
<img src="https://bank.com/xfer?amt=1000&toAcct=evil37">
```

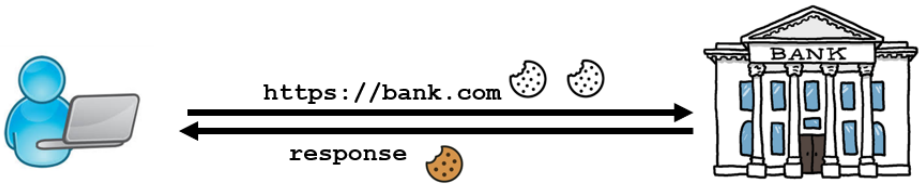
When the browser tried to render the image, it would send the URL to `bank.com` requesting a transfer of $1000 from your account to evil37's account. Any active session information contained in cookies for `bank.com` would automatically be sent with the web request.

Such attacks are effective because the attacker is in no hurry. Many thousands of people might access the web page with this malicious link and be unaffected because they had no account with `bank.com` or were not currently logged into their account. In this case, the attack would quietly fail. However, with time and repetition, such an attack is likely to ensnare unsuspecting people.

Note that the attack is using a secure (encrypted and server authenticated to the user) connection via HTTPS. While using such a connection is an important best practice, by itself it is insufficient to prevent a CSRF attack.

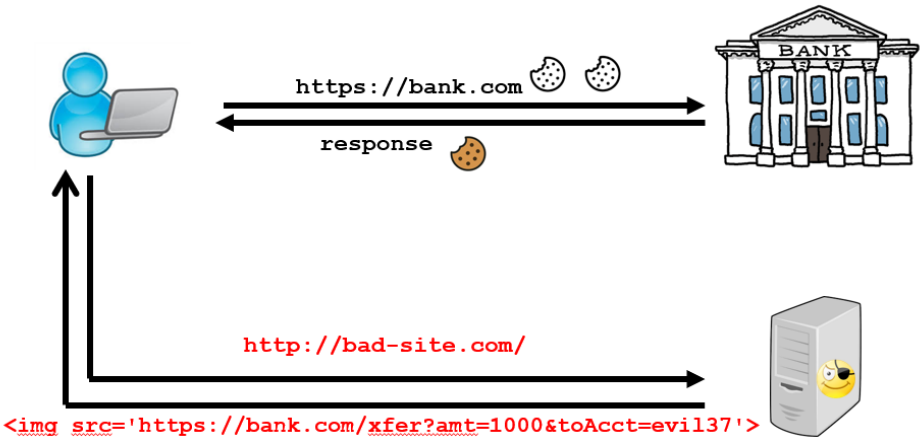
The Attack in More Detail

Now we will walk through the steps of such an attack in detail. We start with the assumption that you have logged into your online bank account and received a session ID from the server in a cookie:



[1] https://www.bleepingcomputer.com/news/security/tiktok-fixes-bugs-allowing-account-takeover-with-one-click/
[2] https://www.cvedetails.com/vulnerability-list/vendor_id-345/opcsrf-1/Mcafee.html
[3] http://www.securityspace.com/smysecure/catid.html?id=1.3.6.1.4.1.25623.1.0.804318
[4] https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

Next, you carelessly accessed a page from a questionable website or clicked on an unverified link in a phishing email message. That web page contained an `<IMG>` tag with the malicious URL that we described previously:



In response, your browser issues the URL causing the unwelcomed transfer of $1000 from your account to that of the attacker:

A More Realistic Version of the Attack

While the example in the previous section is conceptually correct, the simple URL that we used in the example is not quite correct. That URL will generate an HTTP GET method request, which should not be used for web requests that cause side effects, i.e., requests that modify data on the server. For this case, the POST method is appropriate. We used the simple URL in the example since it was more compact. However, we can easily construct an equivalent request that uses the POST method:

```
<script>
  var req = new XMLHttpRequest();
  req.open('POST','https://bank.com/xfer','True');
  req.setRequestHeader('Content-Type',
      'application/x-www-form-urlencoded');
  req.send("amt=1000&toAcct=evil37");
</script>
```

We use Javascript to create a new HTTP request object. (For some reason, the designers of this class put "XML" in the front of the name. Nonetheless, this creates an HTTP request object.) We then define this request to use the POST method and provide the first part of the URL. The last line of code adds the

parameters the web request and submits it. We can obtain the same result using HTML forms and a bit of Javascript. The web page would contain a form that caused no visible change in the appearance of the page. The action associated with the form contains the first part of the URL. The form has hidden fields that define the parameters to be added to the web request, along with values for those parameters. The bit of Javascript following the form automatically submits the web request based on the form.

```
<form action="https://bank.com/xfer"
 method="POST" target="invis">
   <input type="hidden" name="amt" value="1000">
   <input type="hidden" name="toAcct" value="evil37">
</form>
<script>
   document.querySelector("form").submit();
</script>
```

The effect of either of these two attack versions will be a request for the unauthorized transfer of funds.

Summary

In this chapter we covered the basic concepts and mechanisms related to cross site request forgery attacks. We discussed the risks associated with a CSRF attack and steps involved in such an attack. We then showed examples of how such an attack could be conducted.

Exercises

1. In this chapter, we mentioned some major websites that have been vulnerable to CSRF attacks. Choose one of these attacks, or another that you have found information about, and research the details. Try to find enough information that you can understand how the attacker conducted the attack and what they were trying to accomplish.
2. Learn how to list the cookies that are stored in your browser. For websites that you commonly visit and that require logon, see if you can discover which cookies are involved in establishing a session.
3. Try out the techniques presented in this chapter using the exercise based on our virtual machine image. Note that you will need the Virtual Box virtual machine system and an x86-based processor.

   https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/Exercises/3.9.3_Web_Attacks_CSRF_Exercise.html