Cross-Site Request Forgery

Technical Overview of Security Vulnerability

A report presented for the degree of
MSc Cyber Security

CSCM28 908306
Swansea University
United Kingdom

March 11, 2020

Contents

Cross-Site Request Forgery

1 Introduction

Cross script request forgery focuses on the idea that an attacker tricks a user into performing an action of the attacker's choosing, through a link or other content, directing the user's actions on the target application [1]. This attack exploits the trust of a website on the user's browser. This type of attack is also known as session riding. CSRF attacks effect applications that use either HTTP GET or HTTP POST to call their actions, although actions invoked with HTTP GET are often easier to exploit [2]. The vulnerability has been known for well over a decade yet frequent occurrences are documented of the exploitation. Searching the common vulnerabilities and exposures list returns 10 recent cases, and 1865 overall [3].

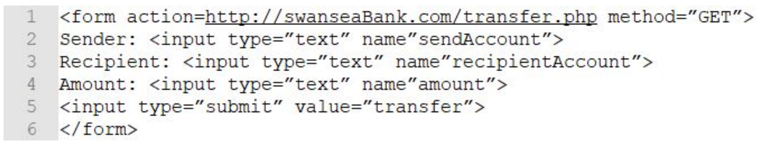
Despite the serious vulnerability concerns this attack presents, a large majority of web developers do not know about the attack and confuse it with cross-site scripting (XSS). Unfortunately, most that do know about CSRF think that protection against XSS will also work for occurrences of CSRF [1]. Therefore, greater awareness should be generated about this type of vulnerability and a framework outlined for dealing with potential attacks in future.

2 Overview of Cross-Site Request Forgery Attacks

Cross script request forgery attacks typically involve 3 main components. These are the user, a trusted website and an illegitimate website from which unintentional actions can be directed from. The victim has two websites open, for example. These websites are trusted site A and malicious site B. Site B injects a HTTP request for the trusted site into the victim user session. After now logging into website A, the user would be working in a valid authenticated session. Without logging out from the session, the user then visits website B. However, in the site B, the attacker has already posted a malicious link which is able to send a HTTP request to site A to request to perform some action that would usually require a valid session [2]. When the victim visits the malicious website and mistakenly clicks on the link, the HTTP request is sent to the trusted website which uses the valid session of the victim to perform that action.

For example, the link **http://www.google.com/search?q=Swansea+University** should redirect anyone that clicks on it to search Google for 'Swansea University'. This may appear harmless by design, however, a link such as **http://www.swanseaUni.com/EditProfileGrade?action=set&key=grade&value=80** could tell an application which authenticates users only by cookie, browser authentication or certificate to update a student's profile and change their coursework grade. Links can be easily masqueraded to conceal word that would expose function and obfuscated so they appear to go elsewhere [4].

The following bank example illustrates a potential use of the exploit. Bob opens his personal banking website to check his balance. This online banking facility also has a page to transfer money to another account. The form to transfer these funds looks something like the following.

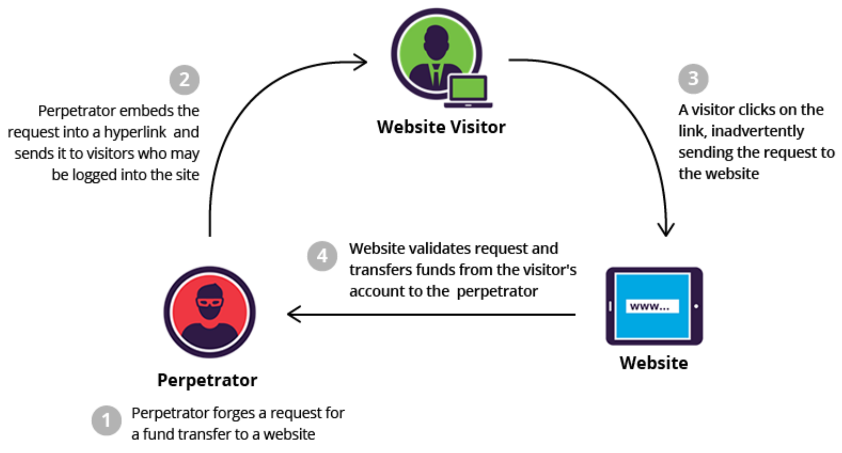


```
<form action=http://swanseaBank.com/transfer.php method="GET">
Sender: <input type="text" name"sendAccount">
Recipient: <input type="text" name"recipientAccount">
Amount: <input type="text" name"amount">
<input type="submit" value="transfer">
</form>
```

Figure 1: Transfer Form for Swansea Bank

If anyone wanted to transfer money, they would have to submit this form through the browser. After submitting, the user should then be directed to the page transfer.php and the URL transformed to look something like the following [1]
http://swanseaBank.com/transfer.php?account=sender&amount=amount&for=receiver.

While the attacker needs to know the structure of the forms, they can create a link in which sends a legitimate HTTP request to the banking website. Therefore, creating a link such as **img src=http://swanseaBank.com/transfer.php?amount=amt&for=receiver** should cause the logged in user to automatically submit the form and the action performed. In this case, the user would then unwillingly transfer funds from their account to another. There are many ways to send these links to users, including third party websites where they are posted as image links (see example). These also include sending links through email and instant messaging services [1]. The use of URL shortener services can also be employed to generate shorter, more generic, URLs appearing less suspect that redirect to the original link [5].



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

Figure 2: Cross Site Request Forgery Illustration

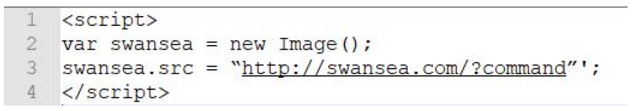
Reference https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/ [6]

3 Ways of Performing Cross-Site Request Forgery Attacks

Cross script request forgeries require prior knowledge of systems. This knowledge may be obtained through caches of the target sites. Once the systems have been analysed, they are reasonably simple to exploit. You require no programming knowledge, no shell codes and no exploit frameworks. If you can construct a JavaScript image object, or HTML image tag, then you can exploit the vulnerability. The most popular way to perform CSRF attacks is through the use of HTML image tags. These are unrestricted and allow the perpetrator to embed any source link [1].

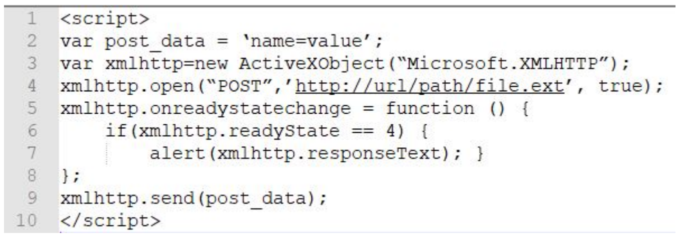
There are common ways in which an attacker may try sending a HTTP request. Tokens and other random values through the use of GET (authentication data in URL) and POST tags. HTML and JavaScript methods and Mozilla scripts may also be employed. Some real-world examples should help illustrate the vulnerability and an exploration of the limitations [2].

HTML methods include the likes of image source **src=http://swansea.com/command**, script source **src=http://swansea.com/command**, iframe source **src=http://swansea.com/command**. JavaScript methods include image and XMLHTTP objects [7].



```
<script>
var swansea = new Image();
swansea.src = “http://swansea.com/?command”';
</script>
```

Figure 3: JavaScript Image Script



```
<script>
var post_data = ‘name=value’;
var xmlhttp=new ActiveXObject(“Microsoft.XMLHTTP”);
xmlhttp.open(“POST”,’http://url/path/file.ext’, true);
xmlhttp.onreadystatechange = function () {
    if(xmlhttp.readyState == 4) {
        alert(xmlhttp.responseText); }
};
xmlhttp.send(post_data);
</script>
```

Figure 4: JavaScript XML-HTTP Object

The following examples describe the vulnerabilities aforementioned. ING Direct (**ingdirect.com**), whilst fixed post 2008, discovered a vulnerability that allowed accounts to be created on behalf of an arbitrary user. They would then be able to transfer funds out of user's bank accounts. This is believed to be the first CSRF vulnerability to allow the transfer of funds from a financial institution [4]. YouTube (**youtube.com**) was also found to have CSRF vulnerabilities in “nearly every action a user could perform on YouTube” [8]. Exploiting the URLs generated by YouTube, an attacker could add videos to a user's ‘favourites’, added themselves to a user's ‘friend’ or ‘family’ list, send messages on the user's behalf, flag videos as inappropriate and subscribe a user to a particular channel.

The Common Vulnerabilities and Exposures website (found at **cve.mitre.org**) contains a list of entries – each containing an identification number, a description and at least one public reference for known cybersecurity vulnerabilities [3]. Searching for cross script request forgery attacks returns 1865 entries; 10 of which from 2020. The second most recent occurrence, provided the CVE-ID CVE-2020-3148, describes a vulnerability in the web-based interface of Cisco Prime Network Registrar (CPNR). It was found that an unauthenticated and remote attacker could perform CSRF attacks on the affected system due to “insufficient CSRF protections in the web-based interface” [9]. The attack could use the exploit to persuade a target user, with an active administrative session, to click a malicious link. A successful implementation of this would allow the attack the change an affected device's configuration, which could include the ability to edit or create user accounts of any administrative privilege level. Cisco has since released free software updates that address the vulnerability described from the list of common vulnerabilities and exposures [10].

4 Preventing Cross-Site Forgery Attacks

Common advice for mitigating cross script request forgery attacks within the web development community is to use POST instead of GET parameters. However, as mentioned previously, simply changing the HTML tag is not adequate for preventing these attacks. Completely removing the GET parameters is also often not possible and would result in applications that are more difficult for users to navigate and developers to implement [11]. Exploring three protection approaches, we assess the level of protection and ease of implementation.

Approach #1 **Change to application only with HTTP Post Operation** Level of protection: Very Low. Only allowing POST operations to perform changes to the state of an application such as creating or updating objects. The approach makes exploitation through image tags ineffective and helps reduce exploitability, however, attackers can adjust their attacks to be form-based, submitting forms automatically and/or misleading users with bogus submit buttons [1].

Approach #2 **Use the HTTP Referrer header to verify action** Level of protection: Medium. When the applications receives a request for action, though before execution, verify that the value of the Referrer header in the HTTP request is from a source that is authorised to call the application. Disallowing access if the header is not present, or if its value points to a page or site that should not be called the action. If the header is not from the allowed set then the action should be aborted. This may not always be the case, however, as the header is optional and disabled in some browsers. It is also not accessible with interactions that occur between HTTPS and HTTP pages. The header can also be spoofed, and the tracking of valid sources may be difficult for some applications [12].

Approach #3 **Use Cryptographic tokens to verify Session-Specific actions** Level of protection: High. As recommended by iSEC, the protection approach involves using query parameters (usually as hidden input tags) to generate a name/value pair with a name such as CSRFExploitToken and the value SEC-sec2 (action name and secret session ID). When the application receives an action request, but before execution, it then verifies the value of CSRFExploitToken by comparing the value of the token to a calculation of SEC-sec2. If the values do not match, then the action was from the correct application. The action should then be aborted and logged as a potential security incident. The advantages include very strong protection and no additional memory requirements per user session. It does, nonetheless, require a small amount of computation when actions are verified [13].

5 Occurrences of Cross-Site Request Forgery Attacks

As we have seen, cross script request forgery attacks are present in our current systems. CSRF vulnerabilities have been known and in some cases exploited, however, since 2001 [1].Due to the nature of the attack, carried out from the user's IP address, some website historical logs may not have evidence of cross script request forgery. These exploits are user dependent and publicly under-reported. We have seen how to initiate these vulnerabilities and cases where they pose actual threats. We have also seen some mitigation techniques, and as of late the introduction of SameSite Cookies [14]. These are new attributes that can be set on cookies to intrust the browser to disable third-party usage. The attribute is set by the server and ensures that requests originate from the same origin. Requests made by third-parties would subsequently not include the SameSite cookie and be rejected. This effectively eliminates CSRF attacks without the use of cryptographic tokens. Yet, the SameSite method is yet to be implemented across all web browsers and should be an important consideration moving forward to eliminate the problem [15].

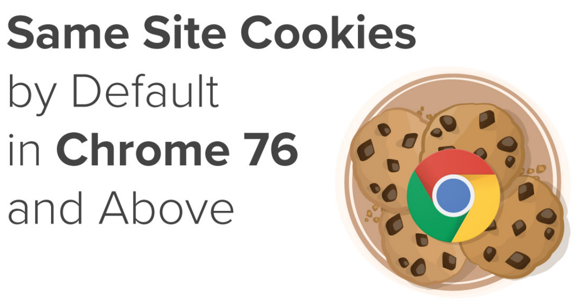


Figure 5: Same Site Cookies Default in Recent Chrome Updates

Reference https://www.netsparker.com/blog/web-security/same-site-cookies-by-default/

With that said, there are still reported instances of CSRF attacks in the current century. According to DrayTek, attacks against web devices were carried out in 2018, attempting to change the DNS settings of certain routers. Some of the manufacturers for these routers hurriedly released updates to promote protection and encouraged users to change these settings to minimise risks. The exact details were not released, citing ‘obvious security seasons’ [16].

6 Conclusion

To conclude on the vulnerability of cross script request forgery, we know that it is a website exploitation unbeknown to website developers so that it still exists in live systems. There are many popular websites that are still vulnerable to these types of attacks and while it can be difficult to protect against, there are numerous methods out there to mitigate the risks associated. Similarly, we are developing systems, such as SameSite Cookies, to futureproof the web environment and force developers to address the problem. The most effective preventative method appears to be that of cryptographic tokens to verify session-specific actions. This is the method employed by many web sites to protect against CSRF.

Bibliography

[1] J. Burns, “Cross site request forgery,” *An introduction to a common web application weakness, Information Security Partners*, 2005.

[2] M. S. Siddiqui and D. Verma, “Cross site request forgery: A common web application weakness,” in *2011 IEEE 3rd International Conference on Communication Software and Networks*, pp. 538–543, IEEE, 2011.

[3] CVE, “Common vulnerabilities and exposures (cve).”

[4] P. Khurana and P. Bindal, “Vulnerabilities and defensive mechanism of csrf,” *International Journal of Computer Trends and Technology (IJ CTT)*, vol. 13, no. 4, pp. 2231–2803, 2014.

[5] ShortURL, “Shorturl,” 2020.

[6] Imperva, “What is csrf: Cross site request forgery example: Imperva.”

[7] S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, and R. Berg, “Saving the world wide web from vulnerable javascript,” in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pp. 177–187, 2011.

[8] F. to Tinker, “Popular websites vulnerable to cross-site request forgery attacks,” Sep 2008.

[9] CVE, “Cve-2020-3148 cisco prime network registrar cross-site request forgery vulnerability.”

[10] “Cisco security threat and vulnerability intelligence,” Nov 2014.

[11] N. Jovanovic, E. Kirda, and C. Kruegel, “Preventing cross site request forgery attacks,” in *2006 Securecomm and Workshops*, pp. 1–10, IEEE, 2006.

[12] R. Garskof, “Apparatus and methods for preventing cross-site request forgery,” apr 28 2015. US Patent 9,021,586.

[13] T. Alexenko, M. Jenne, S. D. Roy, and W. Zeng, “Cross-site request forgery: attack and defense,” in *2010 7th IEEE Consumer Communications and Networking Conference*, pp. 1–2, IEEE, 2010.

[14] I. Muscat, “What is a csrf attack,” Jul 2019.

[15] L. O’Reilly, T. Peterson, L. O’Reilly, K. Barber, M. Willens, T. Peterson, S. Joseph, and L. Southern, “Wtf is chrome’s samesite cookie update?,” Jan 2020.

[16] DrayTek, “Home.”