

Reflected CSRF Attacks and Mitigation

with a Case Study on CVE-2021-32677

Anders Lie

Abstract

Cross-Site Request Forgery (CSRF) attacks are made possible by a combination of authentication cookie misconfiguration and poorly configured or implemented REST servers. Attackers can use CSRF to make authenticated requests to a target site upon a victim's visiting of the attacker's site. This paper explores the differences between reflected and stored CSRF attacks, the implementation and/or configuration flaws necessary in order for CSRF attacks to be possible, and finally how the vulnerability described in CVE-2021-32677 removes one of the necessary conditions for a reflected CSRF attack to be possible.

1 Introduction

Cross-Site Request Forgery (CSRF) attacks are a general class of HTTP attacks which use browser-stored authentication tokens to send unwanted commands to a target site.

A typically flow for a CSRF attack is as follows: A victim user is logged into a target site via an authentication cookie stored in the browser. The victim visits an attacker's site which returns JavaScript with a malicious API call to the target site. The victim's browser executes the JavaScript code with the malicious call and passes the stored authentication cookie to the target site endpoint. Thus the attacker impersonates the authenticated user and makes an authenticated call of their choosing to the target site. There are some necessary conditions for such an attack to be possible, as well as some limitations which will both be discussed later, but this is the basic process of a CSRF attack.

CSRF attacks can be categorized as either a reflected or a stored CSRF attack. In a reflected CSRF attack, the malicious request is made either automatically when the victim visits the attacker's site, or when they click a specific link on the attacker's site. For a stored CSRF attack, the attacker makes some kind of post on the target site (e.g. a forum post) which includes a malicious link leading to the execution of the malicious request, or the attacker utilizes a stored XSS attack so that the malicious request is made automatically via injected code stored on the target site [Lin+09]. The previous paragraph described a reflected CSRF attack, which will be the focus of this paper, as preventing stored CSRF attacks often comes down to preventing XSS, which is not the technique this paper aims to study.

2 Related Work

There is an abundance of research on various aspects of CSRF attacks, including threat models [Lin+09], prevention [YP17], and various techniques for detecting CSRF vulnerabilities [SZ10] [Cal+19] [Liu+20].

The IEEE paper on threat modeling analyzes CSRF vulnerabilities and creates threat models to aid CSRF researchers in designing defences for both reflected and stored CSRF attacks [Lin+09]. The paper defines reflected and stored CSRF attacks and the differences in their execution but does not compare them in depth.

Another IEEE paper on the challenges of CSRF security and CSRF prevention techniques covers some examples of CSRF execution, as well as some possible defenses. It describes both automated defense techniques and defense techniques requiring user interaction [YP17].

Finally, significant research has been conducted on detecting CSRF vulnerabilities. One IEEE paper describes several client-side techniques for detecting both reflected and stored CSRF attacks [SZ10]. Other research dives deep into data-driven approaches for detecting CSRF attacks: for example one with machine learning [Cal+19] and another utilizing state-transition graphs [Liu+20].

This paper differentiates itself from previous research on the topic in several ways. Firstly, this paper focuses on the common misconfigurations and implementation flaws which enable CSRF attacks as opposed to trying to encompass all possible CSRF vulnerabilities, as research in detecting these vulnerabilities has shown that they can exist in many complex forms which are difficult to identify [Cal+19] [Liu+20]. Additionally, while previous research has defined the differences in reflected and stored CSRF attacks [Lin+09] as well as their detection [SZ10], this paper explores the differences in what is necessary to enable either technique and includes an experimental example of a reflected CSRF attack. Finally, this paper describes how a reflected CSRF attack can take advantage of the vulnerability described in CVE-2021-32677.

3 CSRF

3.1 CSRF vs XSS

Before discussing the details of CSRF attacks, it may be useful to clear up the differences between these attacks and XSS (Cross-Site Scripting) attacks. These attacks might be related in some contexts but are not the same type of attack. In short, the difference is, “Unlike XSS attacks, that exploits the trust of the client in the website, [a CSRF] attack exploits the trust of the website in the client,” [ACA16] - the client meaning the browser. So XSS involves somehow injecting a script into a website that the browser then loads, whereas CSRF involves utilizing authentication cookies stored in the browser to access a website’s functions (e.g., authorized REST endpoints).

3.2 Reflected and Stored CSRF Attacks

CSRF attacks always involve a malicious request to some target site endpoint. Consider a piece of malicious JavaScript code constructed by an attacker to make such a request, taking advantage of the site’s trust of a victim’s browser. As briefly mentioned in the introduction, reflected CSRF attacks generally involve delivering the malicious script via some third-party site set up by the attacker, whereas a stored CSRF attack involves somehow storing the malicious script on the target site, for example by using XSS techniques to execute the malicious code when the victim visits the target site.

3.3 Vulnerabilities Enabling CSRF Attacks

In order for a CSRF attack to be possible, there are two key requirements. The first is that the browser must be trusted in some capacity by the backend of the site, for example via a stored authentication cookie. The second is that it must be possible to make requests to the backend of the site from the victim’s browser, either via JS hosted on the attacker’s site or via some injected script on the target site.

As long as these two requirements are satisfied, a stored CSRF attack is possible. Assuming the attacker has a method of injecting a script into the target site (XSS), then that script will be able to use the stored authentication cookie to make malicious, authenticated requests to the backend. Thus the security of this kind of attack comes down to preventing XSS techniques, which is not the focus of this paper. Instead, the specific requirements needed to enable a reflected CSRF attack will be investigated, as this kind of attack does not require XSS.

The first requirement, stored authentication cookies, becomes vulnerable to reflected CSRF attacks if these cookies are misconfigured. In this regard, the most important attribute of the cookie is the SameSite attribute. This attribute can be set to Strict, Lax, or None. Strict and Lax both have measures

to prevent cross-site requests from using the cookie - in other words, only requests made from the site the cookies are stored for can use those cookies. However, if this attribute is set to None, then any call made by the browser, from any origin, can automatically pass that cookie to its respective site in a request. Reflected CSRF attacks are most viable when this attribute is set to None.

The second requirement becomes a vulnerability for reflected CSRF attacks if CORS is misconfigured on the target site backend. CORS, or Cross-Origin Resource Sharing, is an HTTP mechanism for indicating which origins should be able to make requests to a particular server. In order for a CSRF attack to be possible, the malicious request must somehow pass through CORS, for example due to misconfiguration or misimplementation of the target server.

3.4 Classification and Impact

Both reflected and stored CSRF attacks are a type of authentication-based attack, since they take advantage of authentication stored on the browser to access server endpoints. Specifically, this exploits user-to-host authentication. However, the specific CVE vulnerability discussed later is also a header-based attack as it takes advantage of the server implementation's mistreatment of HTTP headers.

The impact of a successful CSRF attack can vary massively, depending on the privileges of the authentication cookie being exploited as well as the influence of the unauthorized server endpoints. It is worth noting that if an authentication cookie only provides access to read-only endpoints, then the CSRF attack can do no damage. Even if a CSRF attack bypasses CORS on the backend, by the time the response reaches the frontend, the browser's CORS will recognize that the origin was incorrect (assuming the CORS headers are correct), and will not read the response. Thus only write operations like POST and DELETE requests are at risk of being exploited by CSRF attacks.

3.5 Mitigation

If the SameSite attribute for all authentication cookies are set to Lax or Strict, and CORS is set up properly on the web server backend, then generally speaking a reflected CSRF attack will not be possible. However, if both of these vulnerabilities exist, or if the CSRF attack is conducted as a stored attack (XSS), then the site can be attacked with CSRF.

4 CVE-2021-32677

4.1 Overview



The Common Vulnerabilities and Exposures (CVE) Program, aims to identify and catalog cybersecurity vulnerabilities. CVE-2021-32677 identifies a vulnerability in FastAPI, a web framework for Python. In versions of FastAPI lower than 0.65.2, when requests are made to the server, the payload is always interpreted as JSON, regardless of the provided content-type header [21]. This allows JSON requests to bypass the CORS preflight by disguising their content-type.

The CORS preflight involves a request from the browser to the server with information about the actual request, to check that the server will handle the actual request. Without this preflight check, any request can be made to the server from any origin. Certain requests are considered "simple requests", which satisfy some conditions, including a "simple" content-type, one of which is text/plain. Simple requests do not trigger a CORS preflight request, thus any simple request can be made from any origin to a given server with no server-side CORS errors.

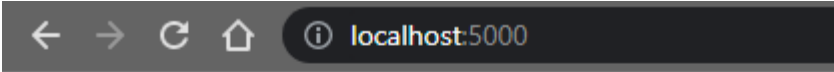
4.2 Exploitation

By passing a normal JSON payload in a request but with the content-type attribute set to text/plain, a request made to a vulnerable version of FastAPI can bypass the server's CORS check. This means that even if the server is configured to only take requests from the origin target.com, the modified request from attacker.xyz will be processed by the server just as well, with no CORS errors (at least until the response returns to the client).

The setup for this demonstration involves a mock bank website which will be targeted by the CSRF attack. A user begins by logging in:

	
<h2>Target Site</h2>	<h2>Target Site</h2>
Username:	Username:
<input type="text"/>	<input type="text" value="victim"/>
<input type="button" value="Login"/>	<input type="button" value="Login"/>
Account Funds: \$[Not Logged In]	Account Funds: \$1000

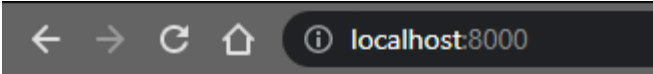
Then, we suppose the user visits the attacker site:



Attacker Site

Hello! Check your account, because I just stole your money!

Now if the user checks their account funds, we can see that funds have been transferred out of their account:



Target Site

Username:

Account Funds: \$500

Even though CORS is properly set up on the server, a CSRF attack via a POST request made to the `/api/transfer` endpoint is successful by setting the Content-Type header to `text/plain`, even though the request contains a JSON payload. Here are the server logs for JSON request with the correct `application/json` Content-Type header, which is blocked by CORS as it should be:

```
INFO: 127.0.0.1:52880 - "OPTIONS /api/transfer HTTP/1.1" 400 Bad Request
```

However, by setting the Content-Type to `text/plain`, the CORS preflight is skipped and the request is made successfully:

```
Session Cookie: 2efdf9b859008ae68781b6aeb73848ca
Updated funds: {'attacker': 1500, 'victim': 500}
INFO: 127.0.0.1:52763 - "POST /api/transfer HTTP/1.1" 200 OK
```

For more details about this demonstration, the code is available at <https://github.com/anerli/cpre-530-paper-demo>.

4.3 Mitigation

Since this vulnerability only exists in FastAPI versions lower than 0.65.2, this vulnerability can be mitigated most effectively by simply updating any REST servers using FastAPI to version 0.65.2 or higher.

Otherwise, as mentioned before, one should be careful to properly configure the `SameSite` attribute of cookies (to either `Lax` or `Strict`), and to properly set up CORS policies to only allow origins which need access to resources on that server. Finally, a common technique used to help prevent CSRF attacks is with CSRF tokens which are NOT stored as cookies. These can be generated per-session or per-request and ensure that that trust is not fully established with the target server with browser cookies alone.

5 Conclusions

CSRF attacks are not necessarily easy to fully prevent, but it helps to understand the basic components of such an attack and what exactly makes them possible. In general, these attacks are enabled when a web server trusts a browser via some stored cookie. Then, all it takes is some misconfiguration of the cookie or CORS, or a bug in the server as seen in CVE-2021-32677, in order for a reflected CSRF attack to be successful. For users of FastAPI, the bug described which enabled these attacks has been patched. However, this example serves as a great reminder of how small implementation errors can enable exploits of potentially very high impact.

References

- [21] *CVE-2021-32677*. Available from MITRE, CVE-ID CVE-2021-32677. May 2021. URL: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-32677>.
- [ACA16] Danny E. Alvarez, Daniel B. Correa, and Fernando I. Arango. “An analysis of XSS, CSRF and SQL injection in colombian software and web site development”. In: *2016 8th Euro American Conference on Telematics and Information Systems (EATIS)*. 2016, pp. 1–5. DOI: 10.1109/EATIS.2016.7520140.
- [Cal+19] Stefano Calzavara et al. “Mitch: A Machine Learning Approach to the Black-Box Detection of CSRF Vulnerabilities”. In: *2019 IEEE European Symposium on Security and Privacy (EuroSP)*. 2019, pp. 528–543. DOI: 10.1109/EuroSP.2019.00045.

- [Lin+09] Xiaoli Lin et al. “Threat Modeling for CSRF Attacks”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 3. 2009, pp. 486–491. DOI: 10.1109/CSE.2009.372.
- [Liu+20] Chang Liu et al. “CSRF Detection Based on Graph Data Mining”. In: *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*. 2020, pp. 475–480. DOI: 10.1109/ICISCAE51034.2020.9236806.
- [SZ10] Hossain Shahriar and Mohammad Zulkernine. “Client-Side Detection of Cross-Site Request Forgery Attacks”. In: *2010 IEEE 21st International Symposium on Software Reliability Engineering*. 2010, pp. 358–367. DOI: 10.1109/ISSRE.2010.12.
- [YP17] Pratibha Yadav and Chandresh D. Parekh. “A report on CSRF security challenges amp; prevention techniques”. In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. 2017, pp. 1–4. DOI: 10.1109/ICIIECS.2017.8275852.