Overview Of Cross Site Request Forgery and Client-Side Protection

Razali Yaakob1, Mohsen Joozdani2, Mohd Taufik Abdullah3 and Azizol Abdullah4

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Malaysia,
Email: 1razaliy@upm.edu.my, 2mohsen.joozdani@gmail.com, 3ntaufik@fsktm.upm.edu.my, 4azizol@fsktm.upm.edu.my

Abstract

As long as internet and web application are a part of our lives to let us to live as easy as we moved like: online market, online bank, online shop and many more, it take attention of malicious to take an advantage of our easy life. Lately there are many types of attacks on web application but so far mostly focused Cross Site Scripting and SQL injection attacks. However there is less attention to prevent Cross Site Request. Cross Site Request Forgery permits malicious to make a request on behalf of user without his/her knowledge. The attack used the authentication between the target website and user through the internet browser. In this paper we would present how Cross Site Request forgery attack works. In additional we present our approach to mitigate Cross Site Request forgery by PCSRF Framework (Prevent Cross Site Request forgery) on Firefox. We propose client side protection. We had experimental test of our framework functionality. From 134 numbers of attacks which contains Post, Get and other methods, we successfully managed to prevent over 79% of attack through three different test sections.

1. Introduction

Cross Site Request Forgery is an attack which uses the victim’s credential to make a request on behalf of him/her. This attack usually made to force the user’s browser to act on behalf of user on trusted site. Thus CSRF attack is working like other web page work, but it contain JavaScript and loading HTML to legitimate HTTP. Most of web developers are used cookies to identify authentication user. The browser will receive cookies to identify by web while there is an authentication session. As long as the user is serving the same webpage, the browser will automatically place the identify login cookie in request and this will be stand until the user close the page or logout. Therefore if attacker could ride this session he/she could easily request whatever needs without user permission. Typically CSRF attack send unauthorized request to the trusted website through user’s browser [1].

CSRF attack has been found early in 1988 by Norm Hardy and it called as Confused Deputy [2]. CSRF has been in the most important web vulnerabilities in last a few years [3]. Among all web vulnerabilities, CSRF attack is the only one could change configuration, generate a main administrator account or execute unwanted action. CSRF attack is quite difficult to detect and to prevent [1]. Typically Web server will respond in all requests comes from the user’s browser as long as there is authentication.

In this paper we will presently describe how Cross Site Request Forgery works by illustration on section 2. Therefore we have illustrated the significant of this attack and present what attacker could do with credential session. More precisely our purpose is on client-side protection to defense from CSRF attack. In additional we mainly focus on client-side protection. However we have short glance of server-side protection as well.

2. Cross Site Request Forgery (CSRF) Attack

CSRF usually happened while there is an authenticated session. Web session cookies and HTTP authentication is the main path of this cross site request forgery attack. Cookies are same as text data which is sent by server to the browser to keep the track of user [4].

Browser keeps the cookies and saved it as long as there is association between browser and server. Therefore while there is authentication with website, for example: user has keyed in the username and password the server return the cookie contains with session ID, which is individually identifies the session to refer cookies authentication. As result most of the time server relies only on the cookies and it assumed whatever request are coming from the browser it will comes from user side. When the browser received a cookies, it will identify an optional characteristic expires among other there characteristic. The expired field takes the value of cookies which how long is valid, after the date is passed the cookies will be deleted. If the cookies is expired then the cookies session should be deleted as well as the user close the browser. Those cookies which have been expired called as persistent cookies. Most financial website and sensitive servers identify the authentication cookie as session cookie. Usually the session cookies are deleted while the browser closed and cannot be used with same computer and browser HTTP authentication [5]. An authentication mechanism defines in the HTTP protocol [6]. In this mechanism, when the user accessing to the webpage which is needed an authentication, the browser will pop up box which asked for user name and password. Once user key in the credential session is made and sent to the web server through the authorization request header and the browser will preserve the credential till the
browser is terminated. After all if the user visiting the same website with same authentication session, the browser automatically includes the credential through the authorization header. The browser keeps the credential until the browser is closed. Once the user visiting the webpage in same authentication session, the browser automatically includes the credential via authorization header.

Cross site request forgery attack usually ride HTTP request such as POST and GET request. GET method knows as safe method which is used to back and recover the object as HTTP/1.1 RFC document has mention [7]. Therefore GET request should not carry permanent consequence (e.g. modification of database). Indeed the operation which has long lasting effect should be made by POST method. Post request usually have request body and they been used to submit forms.

Cross-site request is happening while a user visiting a web page and need to link to another web page. Moreover cross-site request are made while the existing link is request different destination. For example, developer to design the website more creative they are using image, scripts, style file, frame and sub frames from a third website. Thus a user clicks on hyperlink or image it will direct the user to the third party website.

2.1 Example of CSRF

Fig. 1, 2, and 3 are inspired from William Zeller and Edward in CSRF [8], which presenting how normally CSRF works.

Figure 1 shows where a browser creates an authentication session with a trusted website. To have trusted action while the user browser request an action over the authentication session.

![Figure 1: Authentication Session](image1)

Fig. 2 depict which there is a request form user through the browser to the trusted website. Thus trusted website will confirms the authentication and will proceed the action.

![Figure 2: Valid Request](image2)

Fig. 3 shows while there is authentication session between trusted website and user’s browser CSRF will be occur very easy by using the same authentication, because the trusted site doesn’t knows the request is coming from user or browser.

![Figure 3: CSRF Attack](image3)

2.2 Existing Cross Site Request Forgery Prevention

There are two main paths to mitigate CSRF, firstly server side protection tools which can completely protect a potential target site from CSRF attacks [8]. However we would not focus on server side protection on this paper. Secondly, client side protection which can protect the user from certain attacks.

2.2.1 Server Side Protection

There are number of server side protection techniques. In this section we only shortly describe the most 3 popular method. Firstly, the most popular class of server side protection is using secret tokens [4]. This protection is to implant a secret token to each response of server to the web page. Therefore each inbound request, the server will validate which token originated from the server and is correctly bound to the user’s session.

Secondly, HTTP referrer header to determine the origin of request is another class of server side countermeasure. However browser and proxy block this header as privacy reason [9].

Thirdly, Barth et al. [3] had recommended an additional HTTP header to indicate the origin of the request although this header would not be removed by the proxies or browser. Also this method has less privacy-intrusive than the referrer.

2.2.2 Client Side Protection and Requirement

Client side protection should be general as long as they have to deal with every single web application. Thus to be protected from CSRF attack we should provide the following requirements:

- The configuration of the system can be transform by user.
- Automatic decision to strip the authentication session.
- The protection should not be relying on the user input. Nowadays most part of web request in browser session is cross-domain. Thus is not efficient for user to validate each request time by
time. Therefore translucent operation is important.

- Usability of mechanism in web 2.0 contexts without failure and should support the current behavior of application.
- Secure by default. The solution must not have false negatives in default configuration.

We also proposed client side protection because is quite easy to stop the attack through the browser rather than the server side. We recommend a plug in for browser to detect and prevent cross site request forgery attack. Therefore we create an extension for internet browser to be installed on client. Automatic detection is one of the advantages of our proposal which user does not need to define such setting in framework.

3. Related work

Zeller & Felten [8] have been working on client-side protection and they have called it as Request Redeo. They used HTTP respond and classified dynamic URL link which is quite same as our method. As they collecting URL through the browser and is not match with current one, it will consider as attack. Therefore the framework will remove the cookies and authentication and ask the user to re-login again.

Moreover Maes et al. in [10] focus on cross origin request. Method in this research concern about all out going HTTP request which it has such important information like: cookies and parameter. Mao in [11] performs a method which called as Browser Enforced Authenticity Protection (BEAP). They have created a Firefox’s extension base on removing authentication credential. The methodology of this extension is based on GET request which should not be used for other operation.

4. Implementation

On this paper we had illustrate the risk of CSRF which mostly happened through the user’s browser. Thus we have proposed the Client Side protection. We have implemented a Firefox browser add-on to catches all requests are sent by browser. This add-on will remove all the cookies, remove HTTP header regarding the defined scenario. Updating is one of the add-on’s advantages, because attacker is not using same methods. Our add-on will analyzed each HTTP request to finalize the decision. All decision is either Strip, Block or allows through the recorder scenarios.

In general our extension will allow the entire requests which are not post. Allows all the request of the requesting site and target site are in same-origin policy. It’s let the site to make a request to the target website. Therefore, if the framework detect any of attack scenarios, it will notes on status panel for user action. The action can be either strip, block or accept. Furthermore the action will be recorded as new rule for future cases. In Figure 4, the framework will analyse each request in three different sections and forward back to the browser if is accepted. Firstly it will check the HTTP source, then it will analyse the request which is internal or external request. Uncertainty the request is external it will check the sufficient credential to make sure the request is coming from user and does not reuse the credential session. At last if the analyzer received invalid data, it will forward to the decision section to strip the authentication and decide regarding declare scenarios for future action from user.

Figure 4. Framework Analyzer

5. Experimental Design

To evaluate our methodology, we had installed our extension on the latest version of Firefox. Therefore we have used our benchmark’s attack to run main attack on the user to test the functionality of framework. On or after collection of attack we only use 134 attacks which mostly are POST requests. In order to test our framework we have used XAMPP as free and open source cross-platform web server solution to upload our attacks method to run. These attacks are mostly made by PHP language which is lunched through the HTML static. All these attacks are standing to evaluate our framework against of CSRF attack which contained HTML and JavaScript code. On Fig. 5 we have depict the number of attack methods.

Figure 5. Attack Model

To evaluate impolite our framework we have test our framework over three classifications as following steps.
Section 1: We have implement attack scenario in term of no input data, in separate web page. For example once user is login in a page.

Section 2: This section is quite same as section 1, but it contains an arbitrary page which contains a form of attack.

Section 3: We have test our framework through the first and second section but it contain add, delete and edit buttons.

In Fig. 6 we have demonstrate the result of collected 134 attacks which contain 36 GET requests, 95 POST requests. From 134 attacks, 89 requests can retrieve data and 45 attacks can modify the system. Based on Figure 5,

- R represents number of request which has been retrieve data from database.
- M represents a number of requests which modify the data from database.

Figure 6. Result of Mitigation

We had classified our attack test into three sections as we have mentioned on above to criticize test mode. In Section 1 and Section 2, our framework detected and prevents 134 attacks as we have shown on the diagram. However in Section 3, the framework could not detect any of 89 retrieve method but it detected and stops all 45 modification methods. As result our framework successfully detect and prevent all M in each three sections. However the prevention was not good accurate in third section on retrieved data from database.

6. Conclusion

CSRF method is mainly abuse the trust of server through the user credential session, allows attacker to make random HTTP request on behalf of user. On this paper we have demonstrate CSRF attack in basic and depict what will happen if attacker could achieve to authenticate session. We have proposed client side protection since the existing server side protection mechanism fain in widely adopted. In our methodology, available scenario will help the framework’s decision to strip, block, and allow the request form authentication credential. Thus it will analyze all HTTP request and finalized the correct decision before removing cookies and HTTP header according defined scenario. The rule and regulation of this framework are truly evaluated against of cross site request forgery attack.

7. References