International Journal of Information & Network Security (IJINS)
Vol.1, No.3, August 2012, pp. 171~180
ISSN: 2089-3299

Client Side CSRF Defensive Tool

Rupali D. Kombade*, Dr. B.B. Meshram*
Veermata Jijabai Technological Institute Matunga, Mumbai.

Article Info

Article history:

Received June 15th, 2012
Revised July 20th, 2012
Accepted July 22th, 2012

Keyword:

Cross Site Request forgery
Defensive measures, Referer
Stored CSRF
Reflected CSRF
Web Applications

ABSTRACT

Cross Site Request Forgery (CSRF) attack is immerged as serious threat to web applications which based on the vulnerabilities present in the normal request response pattern of HTTP protocol. It is difficult to detect and hence it is present in most of the existing web applications. Various defensive mechanisms have been suggested for CSRF but none of them provides complete protection against it. Few of these are client side tools and other needs both client as well as server side implementation. Maximum of these works for Reflected CSRF and very few has taken a note of stored CSRF. So to handle protect web applications securely, strong and client side protection against CSRF is needed. In this paper we have proposed CSRF defensive tool which provide complete CSRF protection. This is client side tool and not disturbs server side functionality. It can be implemented on browser as a plug-in. This tool works for both stored as well as reflected CSRF attack.

Corresponding Author:

First Author,
Veermata Jijabai Technological Institute Matunga, Mumbai.
Email: rupalikombade@gmail.com

1. INTRODUCTION

Cross Site Request Forgery Attack (CSRF) is less known to developers and it is difficult to detect hence it is present in wide range of web applications. It has been known as “sleeping giant” of web-based vulnerabilities [1] because many sites on the Internet fail to protect against them and they have been largely ignored by the web development and security communities. So CSRF considered as a serious threat to web applications. It is listed at the fourth position in OWASP top ten web application attacks. [2]

CSRF attack exploits HTTP protocol’s functionality of sending session cookies for each and every request in case user is authenticated. It can be carried out by forcing the authenticated end user to perform action he is not intended. This does not require a lot of efforts; this can be done simply by studying the request pattern of action attacker want the end user should perform. Then adding this request URL into the page which end user is going to use at the same time when he is logged in into the vulnerable site. As user is logged in into the site, browser will send a cookie header in request. When server get request with cookie header, it considered request as a valid request and execute this request. Hence CSRF makes successful execution of a forged request without knowledge of end user. Section II has described the CSRF vulnerabilities and attack methodology with example.

Various defensive mechanisms have been suggested to protect web application from CSRF. But every mechanism has some drawbacks as shown in [3]. Some are technology dependent like CSRF guard, anti-CSRF etc… which can work for specific technology only [2] [4]. These mechanisms use extra session token for sensitive actions in web application that’s why it need server side implementation. There are large numbers of websites which are vulnerable to the CSRF and hence server side implementation is not a good solution, each website owner can’t make changes to his website. That’s why our solution mainly focuses on client side tool, which do not need server side implementation and keep web application owners and developers away from extra efforts. Some defensive measures blocks suspected request based on cross origin policy, which is not good solution as number of web application need cross site requests execution and hence this solution will not work for them.

To solve this issue we are not blocking cross origin request instead we just removing cookie header which make request to execute but without cookie header. And hence attacker cannot perform authorised action on user's account. There are very few defensive mechanisms which work for both Stored and Reflected CSRF; our designed solution provides protection against both types of CSRF.

In this paper we have proposed the new CSRF protection tool and shown its implementation. This proposed tool can be installed on browser and it overcomes the most of the shortcoming of previously suggested defensive mechanisms of CSRF. We designed this tool as a Mozilla plug-in.This paper is divided into following sections. Section 2 describes various CSRF vulnerabilities present in web applications; Section 3 contains available CSRF Mitigation techniques, section 4 explains the proposed CSRF defensive mechanism and section 5, Results and analysis consist of implementation of the proposed solution and its results, section 6 concludes this paper, and then referenced material and author's bibliography are listed.

2. CSRF VULNERABILITIES

There are different vulnerabilities exist in web applications [3] which allow CSRF to execute successfully. Most important vulnerability exists in HTTP protocol. This is very first and important vulnerability which cause CSRF to occur. HTTP protocols way of handling cookie information is vulnerable to the CSRF. Let's consider here login process on web application. Whenever user logged in into site, to maintain the state of user, server sets cookie or session. Server sends this cookie/session information in the response header to the client. So once client got this set cookie header, for each and every further request from the client, for specific server, will contain cookie header having same cookie value until cookie expires. This helps server to identify the logged in user. CSRF attacker uses this HTTP protocols functionality to perform any action which needs user authentication. So whenever user logged in into any website and if browser gets a request for same website, browser will send cookie header with that request and hence request will get executed. So attacker can generate this request for the action they want and manage to execute this request when user is logged in into the website. To stop this we need to go change this HTTP protocols functionality. So many other defensive measures are suggested, we will discuss them in next section.

Different HTML tags can be used by attacker to set request to execute. These tags automatically execute request and hides this fact from user. Such tags with their exploited form are listed in [5]. Input validation error vulnerability in web application helps attacker to include the malicious request in web application itself for stored CSRF. Form submission mechanisms used by web application provide information to the attacker to generate request to perform specific action. For example GET method of form submission shows request in address bar and request generated by POST method can be obtained by building the same form fields and submitting the form by using JavaScript [6]. Browser itself provides sufficient information to the user to generate valid request externally. For example view source option in menus of browser shows all structure of forms, hidden fields in forms [3]. Many real world CSRF vulnerable sites with their details are listed in [6].

2.1. CSRF Example

Above discussed vulnerabilities are present in maximum web applications, which helps attacker to carry out CSRF. Attacker makes use of these vulnerabilities using different ways and carry out attack. Here we will see one example of CSRF.

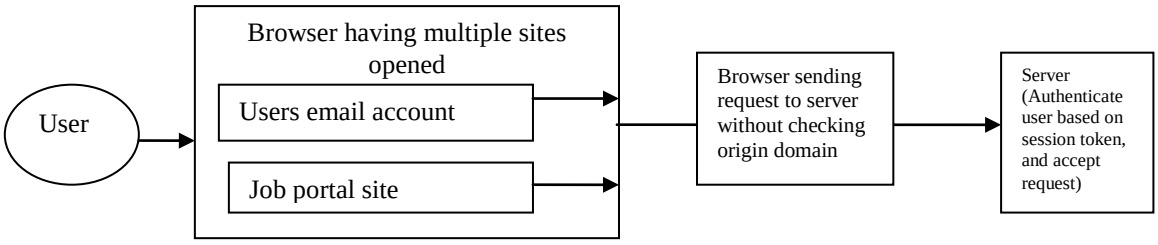

Figure. 1. CSRF Attack scenario

Consider, the user logged in to his email account and found email saying that 'check jobs matching to your profile', If user open this email and follow the link given there, in next tab jobs website page will get open. If attacker also has email account in same site as user and he know how 'change password' functionality works. Based on this information he can generate a link for this change password action. Consider attacker has embedded this request in any html tag as shown below and put this tag in jobs website page. <img src=www.examplemail.com/changepass. php?newpass=somevalue>

3. CSRF DEFENSIVE MECHANISM

Referer privacy guard blocks the suspected request based on referer header. Referer header consists of origin URL of specific request. This can protect against Reflected CSRF only. Also in case where referer header is not sent for privacy or security reason it will not work at all. Whitelist tool works on concept of whitelisting the suspected URL for specific website [5]. It can be implemented as client side tool which alerts user on detecting suspected Form action and will ask user to add this URL as a valid against websites address. This information added by user will work as whitelisting and get used for further actions for that specific website. This tool requires user involvement in defensive strategy, but all internet users are not aware of attacks and its effects. So this cannot provide complete protection against CSRF. Custom headers are also used to protect against CSRF, these prefixed with X-, are sent to the client together with the default HTTP header. One importantproperty of these headers is that they cannot be sent cross domain that's why cross origin requests are not able to execute [7]. All these above listed mechanism works for Reflected CSRF only; they cannot protect web application from Stored CSRF. We need the defensive mechanism which will work against both types of CSRF attacks.

Some defensive mechanism works on concept of extra session token; these are strongly generated random tokens which sent with each request. These tokens are generated by these tools automatically for each request and it also check availability of these tokens while processing request on server. So with authentication details this token is also required to perform specific action on server. This increases attacker's work while generating forged request and make him difficult to perform CSRF. CSRF Guard and Anti CSRF are tools which are based on extra session tokens concept. These tools need to install on web application to protect them from CSRF. CSRF Guard can work with .NET, Java and PHP web applications and Anti CSRF is build specifically for .NET applications[2] [4]. So both of these defensive mechanisms are application specific as well as need server side implementation. There should be CSRF defensive mechanism which is application independent and can be used anywhere.

CSRF detector is a client side tool which work on concept of visibility and content checking of suspected requests, hence it works for both reflected as well as stored CSRF [5]. Requests are considered as a suspected based on the visible forms and their fields values. These suspected request first send to the sever by making it benign, then response of these requests are checked for content type and if original content type of request is matched with response's content type, if it is not match then it is considered as CSRF attack.This tool is just detector and it required large processing time and generate extra traffic to get content type of response of request. DRef is a practical defense mechanism against cross-site and same-site request forgery attacks using privacy-preserving fine-grained access control. For each request to be sent to the website, the browser checks the initiating URLs and the target URL associated with the request against the scopes configured in the policy file. The main idea of DeRef is to employ two-phase checking. First, the website configures (i) the scopes that are permitted to initiate sensitive requests and (ii) the scopes on the website that are protected by DeRef. [11] Hence Dref also need server side implementation and it only protects against Reflected CSRF.

So any of the defensive mechanism discussed above is not providing complete protection, some of these providing good protection but they need to implement on server side. We are trying to solve these issues in our proposed solution which is explained in section IV

4. OUR APPROACH

We discussed different existing defensive mechanism against CSRF and their drawback in previous section and it concludes that web application needs defensive mechanism which should have following properties. (1) It can be implemented as client side to reduce efforts and extra work of application developers, (2) It should provide protection against reflected and stored CSRF attack.

Above two requirements are a basic of our proposed solution. We tried to provide CSRF defense completely through client side and hence named the tool as 'Client side CSRF defense' (CSCD). We need client side defense because many of present web application are vulnerable to CSRF, also most of the web developers are not aware of it and hence they are not implementing any security mechanism against CSRF while developing new website. In such condition we cannot ask each developer or website owner to make changes to their already developed website. This will increase their expenses as well as it is time consuming

solution. Client side tool can be a solution over this problem. Client side tool once implemented on browser will protect all technology and all types of web applications from CSRF. It is technology independent and not need user involvement, which are the drawbacks of already present defensive mechanisms. Also CSCD protects against both stored and reflected CSRF; which is major drawback of previous discussed mechanisms. We implemented CSCD as a browser plug-in. As Mozilla is open source and most popular, we designed plug-in which can be installed on Mozilla browser. We use JavaScript to build this plug-in. Figure 2 explained the flow of 'CSCD'.

CSCD intercepts each request going to server, checks whether it is reflected or stored CSRF attack, and if it detects the request as attack, instead of blocking the request it just removes cookie header from request. So that attacker cannot perform authorized actions. This will block attack without blocking any request and hence false negative detection of CSRF will not affect the web application performance. We can divide the CSCD working in two parts.

4.1. Reflected CSRF prevention

When any request is arrived CSCD will get its target and source URLs. To get source URL we are using two mechanisms. First one is Referer header in http request provide us origin URL of request. But it is found that referer header is widely blocked at network layer due to privacy concern [9], in such cases we are using another way to get source URL. That is we are checking the tab or window from where request was originated. Then get the URL of that tab or window. This will provide us source URL in case where referer is not present. Once we get source and target URL, we checked their enterprise-level domain i.e. www.google.com, www.yahoo.com, here enterprise-level domain is 'Google', and 'Yahoo' etc… we are concentrating on enterprise-level domain part of Domain name because so many website has internal sub domain where they are sending request and performing important actions which may need authentication. For example, on opening yahoo email account, following requests gets send to server.

```
ucs.query.yahoo.com
prod2.rest-notify.msg.yahoo.com
dps.msg.yahoo.com
http://us.bc.yahoo.com
http://mail.yahoo.com/
http://mail.yimg.com/pd/
```

Here we can see different third level domain like query, msg, bc., mail and yimg. Here all URL's have a same enterprise level domain i.e. 'yahoo' except 'http://mail.yimg.com/pd/'. We are checking enterprise-level domain for cross domain request detection, if Source and target Enterprise-level domain are not same that request considered as CSRF attack and the cookie header get removed from it and it sent to the server. This will not affect the false positive results of detection of CSRF. So in above example cookie header will get removed in case of request 'http://mail.yimg.com/pd/' only. Other requests will not get affected in this functionality. We can see here that yimg is a domain where images are getting from and while loading images there is no need of cookie header i.e. Authentication header. We are removing cookies header based on the fact that if any web application contain cross domain functionality, cookies/session or authentication details are not involves in it. For example web application send request to payment gateway, in this case no cookies/session of one site required to other site. Session between payment gateway and web application is maintained by some parameters. Removal of cookie header from request will not allow cross domain request to perform any action that need authentication details. Hence attacker cannot perform CSRF.

4.2 Stored CSRF prevention

In case of reflected CSRF we checked for dissimilar enterprise-level domains, if it is found that enterprise-level domain are similar then it meanse that the request originate from same domain, in this case to detect stored CSRF we checked the various HTML tags attributes, where attacker can add forged request. We have seen how different HTML tags can be used to send forged request [5]. Just consider image tag, we know that 'src' attribute of image tag contain request to load image, and this has not to do anything with authentication credential or session/cookie.

```
<img src="" alt="">
```

Same like image tag there are so many different tags which loads files, the request to load files do not need cookie header or authentication details. For example link tab contain 'href' attribute to load CSS

file, 'background' attribute of different tag is used to load background images. CSS files and images while loading does not need cookie header or authentication details.

We matched the request's target URL with 'src', 'background' and 'href' attribute of different tags present on particular page. 'src' attribute is present with 'iframe' tag, where application loads web pages and this needs cookie header. Same thing will happen with 'href' attribute, it is present with both 'a' tag and 'link' tag, where 'a' tag is used to redirect page to other webpage and this may need session cookie. Hence our second step is to get the tag of the matched attributes, and if these tags are other than 'a' and 'iframe' then cookie header from the request is removed and the request is sent to server. This way we are managing the stored CSRF protection. We are still searching for more tags and their attribute which are used to load request, so to make stored CSRF protection mechanism more strong.

5. RESULT & ANALYSIS

Our implementation of CSCD is described below. First we will go through attack implementation, and then we described how CSCD installation blocks the CSRF attack.

5.1. Attack

Here we have discussed the Reflected CSRF attack. For this attack first we searched for a CSRF vulnerable website. We found one shopping website where we can add shipping address to users account if user is logged in. Then we build a web page to attack on this shopping website.

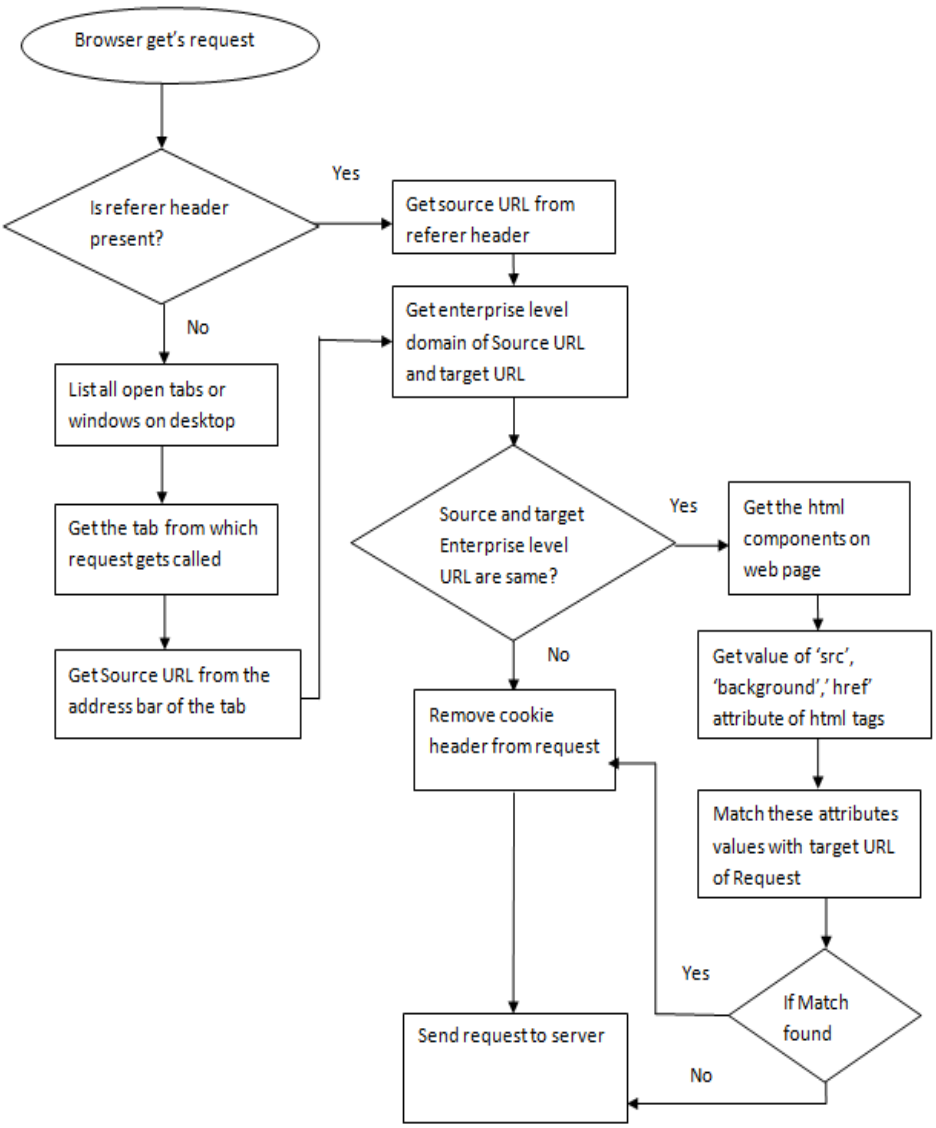

Figure 2. Work flow of CSCD

We create a account on the shopping website, then checked the whole functionality of adding shipping address, checked all the form fields required for it. Then we build a new website to attack on this shopping website. First we build index page same as timesjobs website, in this page we added the iframe tag; 'src' attribute of this tag contains the link of page named attacker.php. This attacker.php has following code, which sending forged request to shopping website. As jobs website's front page is loaded, it will load the iframe tag and open attacker.php in iframe. As soon as attacker.php will load, it will submit the form and request get sent to the shopping website.

Now consider if I am as an attacker sending emails to my friend who also have account on same shopping website. Subject of the website are 'Discount on some products on shopping site' and another email saying 'check the jobs matching to your profile'. My friend saw these emails and logged in into the shopping website and then opened the jobs email and clicked on link given for jobs website in other tab. The jobs website will do its work; new shipping address will get added into the users account. Without his knowledge new address got added to his account and successful CSRF attack is carried out.

```
<body onload="javascript:document.forms[0].submit();" >
<form action="http://www.example.com/userdir/addressdir/" method="post">
<input type="text" name="first_name" value="Test1c" />
<input type="text" name="last_name" value="Testc2" />
<textarea class="input_l tgd" name="address" cols="40" rows="3"
id="id_address">456abc</textarea>
<input type="text" id="id_cityname" maxlength="100" class="input_m tgd"
value="abcc" name="cityname"/>
<input type="text" id="id_pincode" maxlength="6" class="input_m tgd"
value="123456" name="pincode"/>
<select name="statename" class="input_m tgd" id="id_statename">
<option value="Maharashtra" selected="selected">Maharashtra</option>
</select>
<input type="text" id="id_countryname" value="India" class="input_m tgd"
maxlength="100" name="countryname" readonly="readonly"/>
<input type="text" id="id_phone" maxlength="10" class="input_m tgd"
value="9812345678" name="phone"/>
<input type="text" maxlength="75" name="email" class="input_m tgd"
id="id_email"/>
<input type="checkbox" id="id_defaddress" name="defaddress" checked="checked"/>
Make this the default shipping address
</form>
</body>
```

5.2. Defense

We implemented the CSCD functionality as a Mozilla plug-in. Mozilla has provided framework to build a plug-in. We installed this plugin on Mozilla browser. After installing plug-in the link 'CSCD' appeared in tools menu as a new menu item as shown in fig. We provided the functionality in CSCD that on mouse over of the CSCD menu item, it will show two more new menu items i.e. START and STOP as shown in figure 3. On clicking the START CSCD will start working and on clicking STOP option will stop its working. We checked the working of CSCD for two cases i.e. when referer header is on and when referer header is off.

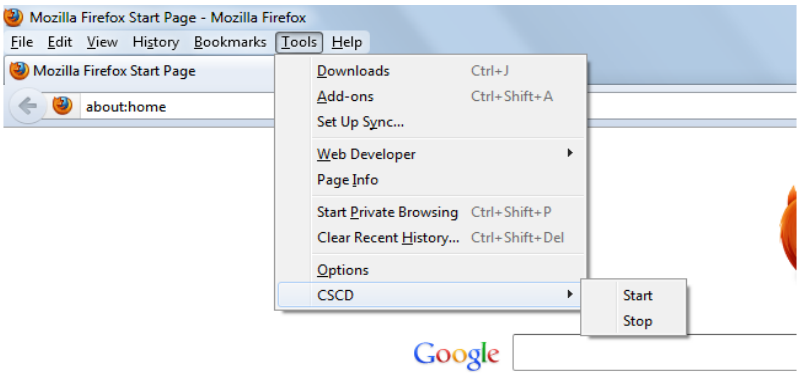

Figure 3. Installed CSCD in mozilla firefox

5.2.1 CASE I CSCD- working when Referer header is on

We Clicked on 'START' menu item of CSCD, and now once again repeated the actions performed in attack section. Format of request going from jobs website to the shopping website in attack case and after starting the CSCD is shown in figure 5 and figure 6 . This format of request is captured by' httpfox' plug-in provided by Mozilla. We installed HTTPfox plug-in on Mozilla browser to capture requests for each action. HTTPfox shows format for each and every request and response running through browser. If CSCD detect that the request is coming from other domain, it removes the cookie header from the request. Figure 5 shows the normal attack case where we can see the jobs website sending request to shopping website and cookies header as shown in figure 5. Figure 6 shows that after installing CSCD, the cookie header is removed from the request as request is coming from jobs website and going to shopping website. When shopping website got the request without cookie header, it does not allow to execute the request and hence address is not get added to users account as authentication details i.e. cookie header was absent.

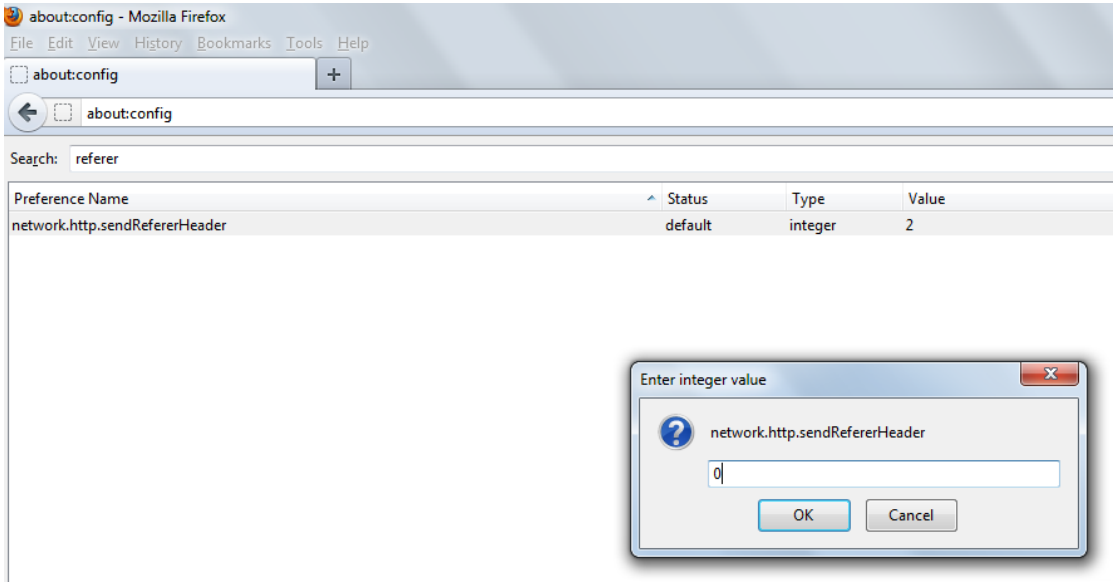

Figure 4. Disabling referer header (for Mozilla browser)

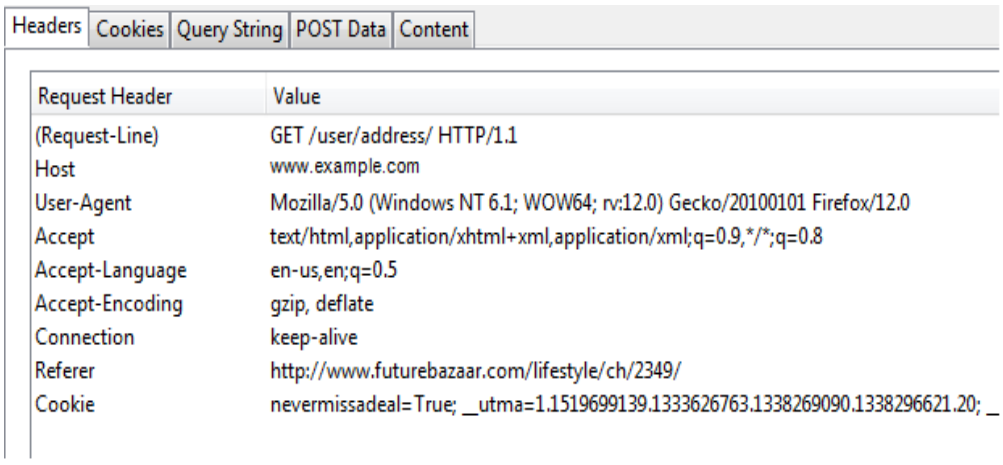

Figure 5. Attack in CASE I

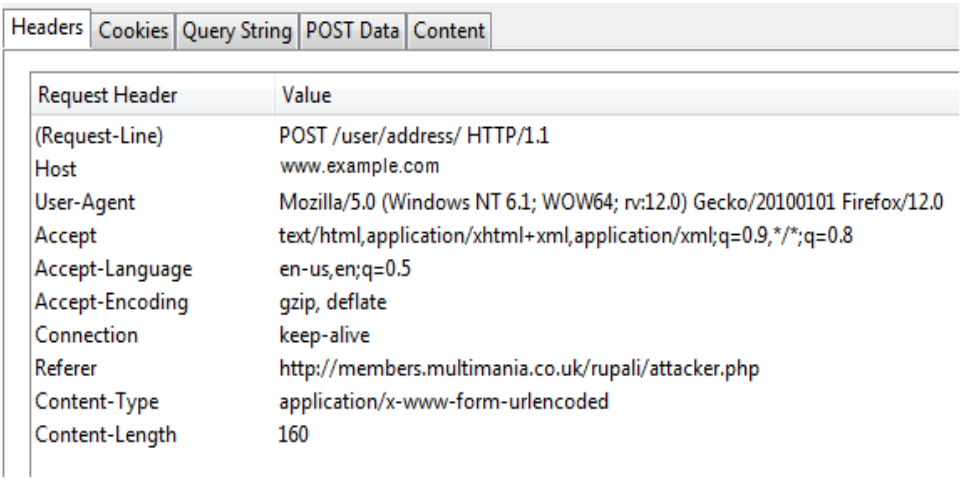

Figure. 6. Defense in CASE I (cookie header is removed)

5.2.2 CASE II CSCD - working when referer header is off

We know that some websites don't send referer header in their requests because of security reason and hence we developed the tool which can work in this situation as well. To test this functionality we disable the referer header property of browser which makes browser to skip referer header while sending any requests. We can make referer header off by changing the browser's properties. we typed the about:config in address bar, it shows us warning and onclicking the yes button it takes us to page where we can see different Mozilla browsers properties and their values. These properties can be changed by rightclicking on it and clicking modify from side bar appeared. This brings a box where we can put new value for this Mozilla property. To disable the referer header in requests we need to search for property 'network.http.sendRefererHeader', by default it has value 2 and to disable it we changed its value to 0. Changing the referer header property of Mozilla is shown in figure 4. Now once again we repeated the same actions we performed in 'CASE I'. It gives us same results as in case I, the request format captured by httpfox in this case is shown in figure 8. Where we can see that request does not contain the referer header and still CSCD detects the cross domain request and removed the cookie header.

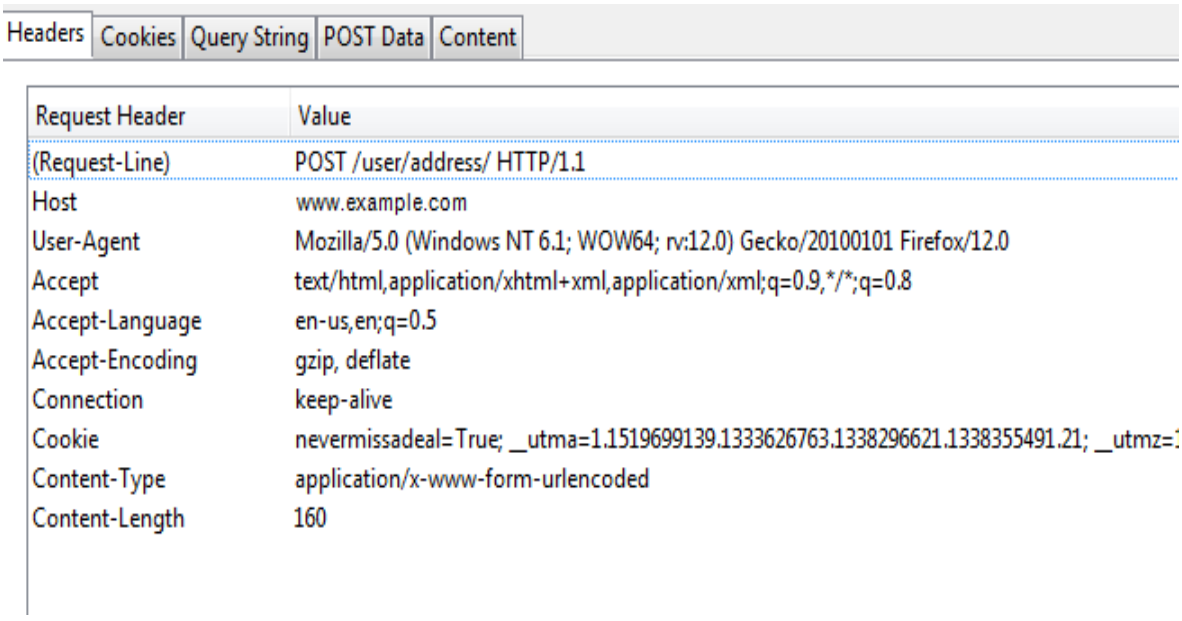

Figure. 7 Attacks in CASE II

We checked the CSCD working in different cases, and found that CSCD works independent of referer header property of request. It works fine in both the cases. We checked other websites functionality after starting CSCD, as they contain so many cross domain operation and these operation should not get disturbed due to CSCD. We found that all functionality on websites running well. We are still working on building a strong protection strategy for stored CSRF.

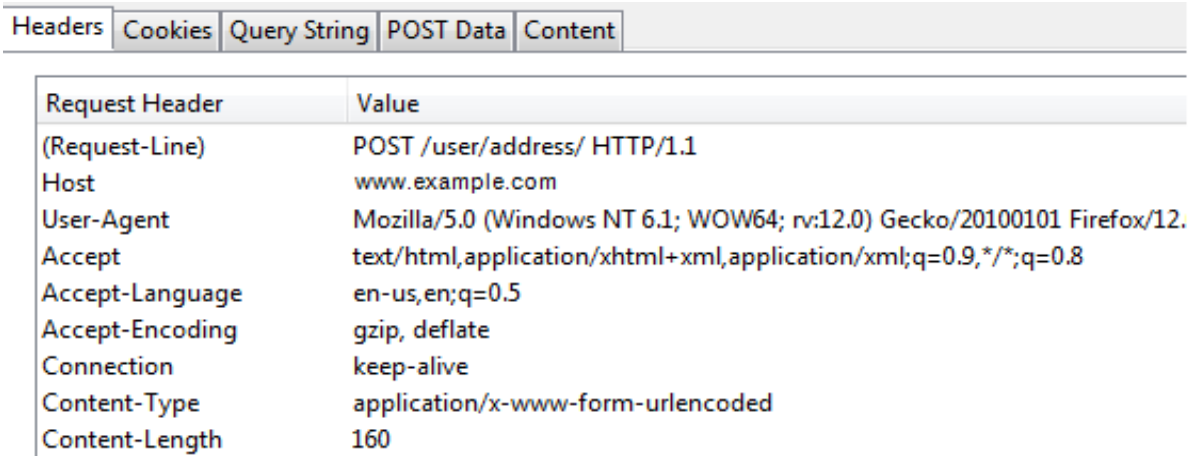

Figure. 8. Defense in CASE II (cookie header is removed)

5. CONCLUSION

We have discussed the CSRF vulnerabilities and defensive measures. To overcome the drawbacks of present defensive measures, we developed new defensive measure which is client side tool and work for both Stored and Reflected CSRF, named as 'Client Side CSRF Defense' (CSCD). This tool will reduce the web application developer's or owner's extra efforts to make changes into their web application while protecting it from CSRF. This tool is not blocking the request after detecting the attack, that's why it allows cross domain requests but just without cookie header, which does not affect web 2.0 functionality. We developed CSCD as Mozilla plug-in because Mozilla is open source and most popular browser. We checked the working of this tool for reflected CSRF, and results shows that it is working well. It successfully protects the system from reflected CSRF. Also it is not disturbing the web applications normal working. In future we will work on CSCD working for stored CSRF to make it more strong and accurate.

REFERENCES

[1]. Grossman, "Cross Site Request Forgery 'The Sleeping Giant of Website Vulnerabilities'", in *RSA Conference, San Francisco*, April 2008.
[2]. OWASP. https://www.owasp.org/index.php/CSRF, Cross-Site Request Forgery, Testing for CSRF (OWASP-SM-005)
[3]. Rupali kombade, B.B. Meshram, "CSRF Vulnerabilities and Defensive Mechanism", IJCNIS 2012
[4]. http://anticsrf.codeplex.com/ AntiCSRF - A Cross Site Request Forgery (CSRF) module for ASP.NET
[5]. Hossain Shahriar and Mohammad Zulkernine "**Client-Side Detection of Cross-Site Request Forgery Attacks",** 21st International Symposium on Software Reliability Engineering, **2010** IEEE
[6]. Mohd. Shadab Siddiqui, Deepanker Verma, "Cross Site Request Forgery: A common web application weakness", 2011 IEEE
[7]. "Seven Deadliest Web Application Attacks", Mike Sharma.
[8]. Xiaoli Lin, Pavol Zavarsky, Ron Ruhl, Dale Lindskog, "Threat Modeling for CSRF Attacks", International Conference on Computational Science and Engineering, 2009
[9]. Adam Barth, Collin Jackson, John C. Mitchell, "Robust Defenses for Cross-Site Request Forgery", CCS'08, October 27–31, 2008, Alexandria, Virginia, USA.
[10]. Dafydd Stuttard, Marcus Pinto "The Web Application Hackers Handbook", discovering and exploiting security flaw"
[11]. Ben S. Y. Fung and Patrick P. C. Lee, "A Privacy-Preserving Defense Mechanism Against Request Forgery Attacks" 2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11

BIOGRAPHY OF AUTHORS

Dr. B. B. Meshram is working as Professor in Computer Technology Dept., VJTI, Matunga, Mumbai, INDIA. He is Ph.D. in Computer Engineering and has published international journal is 25, National journal is 1, international conference is 70 and national conference 39 papers to his credit. He has taught various subjects such as Object Oriented Software Engg., Network Security, Advanced Databases, Advanced Computer Network (TCP/IP), Data warehouse and Data mining, etc at Post Graduate Level. He has guided several projects at graduate and post graduate level. He is the life member of CSI and Institute of Engineers, etc.

Rupali Kombade received her B.E. Degree in Computer Engineering from RTM Nagpur university. She has worked as PHP and .NET Web developer. She is pursuing M.Tech degree in Network Infrastructure Management System from VJTI, Matunga, Mumbai, INDIA. Her research interest includes web development, web security, web application attacks and its defense