International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869 (O) 2454-4698 (P), Volume-5, Issue-2, June 2016

Server Side Protection Against Cross Site Request Forgery Using CSRF Gateway

Jaya Gupta, Suneeta Gola

***Abstract*— The E-Commerce and Social Media has become the new identity for millions of users across the globe. Ease of services for Shopping, Travel, Internet Banking, Social Media, chat and collaboration Apps etc. have become part of one's life where these identities have name, media content, confidential notes, business projects and credit cards. Convenience and connections brings the ease of connectivity and services so does come the concerns related to unauthorized usage and fraudulent transactions that could be lead to loss of money, time, emotions and even life. Web defacement, fake accounts, account hijacking, account lock and unavailability of services has become a common online news and distress for many. There are different Web Attacks and exploits that have sprung up with time and usage for different type of illegal actions performed everyday online. Cross Site Request Forgery Attack is one of the Web top 10 exploited attacks for the past 5 years (Source OSWAP) which can maliciously exploit online services, where unauthorized actions are performed by the fraudulent user on behalf of a trusted and authenticated account for website. It forces the victim user to perform some unauthorized activity on behalf of attacker request. This research work focuses on a new Hybrid strategy that will enhance the server side protection against CSRF attacks. CSRF Gateway, is the proposed solution which provides the Server Side protection against Cross Site Request Forgery (CSRF) Attack.**

***Index Terms*— Cross Site Request Forgery, web application vulnerabilities, CSRF Gateway**

I. INTRODUCTION

Cross Site Request Forgery (CSRF) is also known as "Session Riding" or "One Click Attack". This attack is a Malicious Exploit type of attack against web application users. This attack has been listed as 7th most exploitable among 10 top Web Attacks. CSRF is an attack which allows an attacker to perform unauthorized POST/GET arbitrary HTTP requests on behalf of victim that is currently authenticated to the website.

The fraudulent user performs unauthorized activity on behalf of an authorized and authenticated victim user. If the victim is authenticated, a successful CSRF attack effectively

by-passes the underlying authentication mechanism. Depending on the web applications, the attacker could send post messages or send emails/message on behalf of the victim or manipulate with the login name or password. Account lock, account hijack, data loss and fake online messages are common fraudulent activities using the CSRF methodologies. Furthermore the results of the attack can be more severe depending the usage scenario. But in contrast other well-known web security attacks such as Cross Site Scripting (XSS) or SQL Injection and Cross Site Request Forgery (CSRF) are appears to be a problem known to the web developers.

CSRF attacks are broadly categorized into 2 types. First one is launched from malicious site to a trustful website. In this type, attacker can only send HTTP request to an authentic website but no secret information can be obtained from the true website. The other type of CSRF attack is based on JavaScript and AJAX. It is called the "Multi Stage CSRF attack", which involves a malicious script that generates multiple HTTP requests and secretly sends the generated HTTP requests asynchronously in the background.

Detection and prevention of CSRF attacks is challenging from browser's side, the usage of same origin policy (SOP) is not enough to prevent CSRF attack. Same Origin Policy (SOP) is defined as the same scheme, host and the URL of the host.

There have been many server and client side protection implementations, few protection plans are still relevant and existing for protection, but unfortunately all these protection plans are not able to protect web application completely for new CSRF exploits that have come up with time. The Hybrid strategy for server side CSRF Gateway implementation is an attempt to enhance the protection against CSRF exploits with session and token approach.

In this paper, proposed solution called as CSRF Gateway, which provides the Server Side protection to the most Open Source Web Applications. This solution is intended to demonstrate the working of CSRF Attack using different Attack Vectors on the real world examples. This gateway methodology demonstration will provide the clear picture about the subject, so that it will create a better picture to understand the defensive mechanisms.

Here are some Real World examples of CSRF Attack

1. ING Direct (ingdirect.com)
2. YouTube (youtube.com)
3. MetaFilter (metafilter.com)
4. The New York Times (nytimes.com)
5. Gmail (gmail.com)
6. Netflix

II. RELATED WORK

In previous years, there is lot of research work has been done in this field. In the previous researches, researchers had proposed techniques and solutions to prevent and defense against the Cross Site Request Forgery.

A. Proxy Based Solution

In this approach, solution to the problem was to decouple the necessary security mechanism from the application and to provide a separate module that can be plugged into existing systems with minimal effort. More precisely, they proposed a proxy that was placed on the server side between the web server and the target application. This proxy was very well sufficient to check and change the requests sent by client and the replies to itself extend applications by using the shared secret technique. In particular, the proxy had to ensure that replies to an authenticated user had to modified in such a way that future requests originating from (through hyperlinks and forms) should contain a valid token, and take countermeasures against requests from authenticated users that did not contain valid token. By decoupling the proxy from the actual application, the XSRF protection could be offered transparently for all applications.

B. Referrer Privacy Guard and Defense Technique

In this approach, Defense mechanism included 2 techniques for the solution.

1. Referrer Privacy Guard

The Referrer Privacy Guard revealed how a constant flow of random HTTP requests could mess up the browsing history at the server side, thus preventing infiltrators from getting access to user browsing trends.

2. Detection and Discouragement

In this section, the focus was on how to detect CSRF signatures in web pages and stop it before commencement. The defense attribute first verified the Client side code before each and every page load and found the CSRF attack involved.

C. Attack Detection Using Windows Form

CSRF Attack detection approach that was divided in multiple sections.

Section A: Attack detection framework

In the section they had assumed that a browser could have multiple windows. A trusted website could be viewed by a user in window after performing an authentication process and the session information was saved in the browser. In this section the following processes were followed.

1. Request Checker
2. Window and form checker
3. Request Differentiator
4. Attack Detection policy
5. Attack handler module

Section B: Visibility checking

The proposed notion of visibility relies on examining windows containing web pages and forms presented in a browser. If a request was GET type, they checked whether it contained any query string or not. If no query string was present, no need to examine it further. However, if a query string was present, then tokenize the string to identify the set of parameters and values. And related the extracted parameters and values with a webpage containing forms with similar fields and values. Note that the form action or target field value should match with the resource file of the suspected request. While examining a webpage, two possible scenarios might arise. These were discussed below:

1. Window and no form
2. Window and form

Section C: Content Checking

Content checking relied on the matching of the response of a suspected request with the expected response. A suspected request often resided as part of an HTML tag attribute value or within. A response page might contain various types of elements (static HTML, JavaScript, and Style Sheets). As a result, they relied on the content type of a webpage to differentiate an attack request from a user initiated request based on the identified tag that contained the request. The content type was often specified in the META tag of a page and was accessible from the response header. After that they discussed the comparison of the expected and the actual content type and how to launch a suspected request in next 2 subsections.

1. Comparison between an expected and an actual response content type.
2. Suspected request modification

Section D: Attack detection coverage and attribute checks

The proposed approach was able to detect a variety of CSRF attacks. Some non-exhaustive CSRF examples were highlighted and related with the checks to detect them.

1. Visible form with field and no value
2. Invisible form with field and value
3. Static/Dynamic HTML tag and URL at tribute
4. Program state retrieval or modification
5. Pre- or post-rendering

D. CSRF Guard

CSRF Guard was verifying the integrity of HTTP requests by inserting a special security token to every active HTTP session established among the authenticated client and the web server. Essentially, the CSRF Guard was doing the filtration of the requests coming in. It was executing following functionalities.

1. Inserted a token to the defined preserved resource.
2. This method did the verification of the token when the preserved resource gets requested. The token origination and certification was used to give the protection against the CSRF attack.

E. Protection Approach

Suggested approach was to protect against CSRF attacks by using some or all of these:

1. Use of random tokens

To use random tokens each time with a form submission could make very difficult for the attacker to guess the next random pattern to fill in the URL.

2. Need to Use post method in form instead of Get

Get and Post are the 2 methods of form submission. Post Method was secure for form submission. In Get method anyone could see the variables and values in URL as a query strings.

3. Limiting the lifetime of authentication cookies

Limit the lifetime to a short period of time. If user was going on other website then the cookies were expired after a short period of time. If the attacker was trying to send any HTTP request to user which he was able to know and he would not fill the password again.

4. Damage limitation

Damage limitation involved those steps which reduced the damage from CSRF. For example if an attacker did manage to perform CSRF on a website then any action done by him was required an authentication every time to limit the damage.

5. Force user to use your form

It was forcing user every time to use the form of website. Use of hidden fields was helpful for this purpose. But this way of protection was easy to bypass.

F. Labeling Mechanism

To prevent the CSRF attack, labeling mechanism called Content Box; was suggested. The Content Box consisted of a labeling function and UCC quarantine policies. The labeling function was used to isolate the UCCs, while the UCC quarantine policy enforces propagation rules for the labeled UCCs. The CSRF attack could be prevented using the Content Box when an untrusted UCC try to access a service that contains sensitive/private information.

The main idea was to divide the content into 2 different types. One was called the "trusted contents"; these contents were created by the web server administrator or the content viewer/user. Since these contents were created by the rightful owner, it was that the scripts within the contents were free from the CSRF attack. The other type was called the "untrusted contents" which were created by other users. Since these contents were provided by users other than the rightful owner, the scripts within these contents might cause the CSRF attack. It was important to differentiate the contents of the webpage since the client browser always trusted the contents of a web page provided by the web server even if the authors of the contents were not trusted by the client. In Content Box, they intended to distinguish the untrusted contents and prohibiting the untrusted contents from accessing web services that contain sensitive data.

Initially in web, UCC was the source of the CSRF attack problem. However, most UCCs were harmless providing that if it was created by the current client. This kind of UCC should be classified into trusted contents since the CSRF attack rarely happened when both of the attacker and the victim were identical. Labeling was used to differentiate the contents and ensured that every HTTP request was labeled, provided that the label cannot be disrupted by the client browser. In addition to labeling the contents of a web page, an access control mechanism was required to patrol the accesses of web services.

1. Trusted label had the freedom to access the contents with trusted or untrusted label.
2. Untrusted label could only access the contents with untrusted label.

Once the contents with trusted label were contaminated by untrusted label, its label becomes untrusted.

III. OVERVIEW OF CSRF ATTACKS

A Cross Site Request Forgery is a type of attack that compels an end user to perform unauthorized actions on the web applications on which they are currently authenticated logged in. Specifically, the CSRF is only tangled with state changing requests but not in the theft of data.

A. Anatomy Of Cross Site Request Forgery Attack

A Cross Site Request Forgery is a type of attack that compels an end user to perform unauthorized actions on the

web applications on which they are currently authenticated logged in. Specifically, the CSRF is only tangled with state changing requests but not in the theft of data. This attack typically requires attacker to have prior access to and knowledge of the vulnerable application. To show the Anatomy of Cross Site Request Forgery (CSRF), there is taken an example of the Bank transaction.

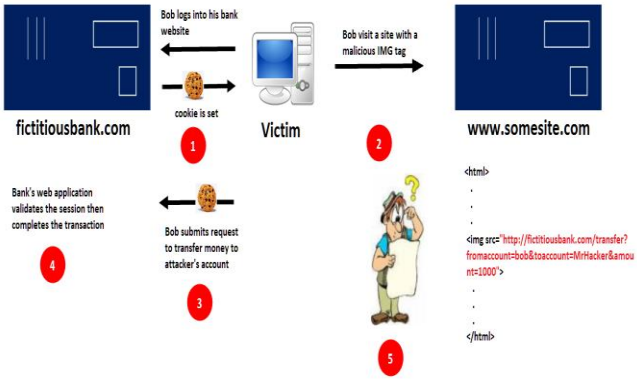

Figure: 1. Diagram of Anatomy of Cross Site Request Forgery (CSRF) Attack

The User Bob logs into the bank website called "fictitiousbank.com". When the session gets established then cookie sets. Then in between Bob visits a site having malicious IMG tag that followed the site called "somesite.com". Behind the scene this link follows the URL which sends amount to another account using Bob's account authentication. The malicious link can be sent either by GET method or POST method. In the GET action all parameters send in the query string and in the POST parameter goes separately to the server. Then in both the cases HTTP request will be,

If action was a POST

```
POST/submitpage
Server: server.com
amount=1000&destination=MrHacker
```

If action was a GET

```
GET/submitpage?amount=1000&destination=MrHacker
Server: server.com
```

If attacker can predict all these parameters, then those parameters can be used to get misused. The GET or POST request can be easily forged by using various HTML elements, such as (img), (script) or (iframe), (a) (hyperlink).

If attacker want to misuse GET then the malicious link will be

```
<img src="……">

http://server.com/submitpage?amount=1000&destination=MrHacker
```

If attacker want to misuse POST then the malicious form will be

```
<form name="evil" action=http://server.com/submitpage action="post">
  <input type="hidden" name="amount" value="1000">
  <input type="hidden" name="destination" value="MrHacker">
</form>

<Script>document.evil.submit () </script>
```

Because Bob does not know about this malicious link, he submits the request which process the request to get submitted on the server and get the money transfer done. Victim user gets to know only when he see the bank statement or after the action gets committed. In this way Cross Site Request Forgery attack takes place. But unfortunately till that time this action cannot be undone.

IV. PROPOSED METHODOLOGY

The *Detection and Prevention of CSRF Attack* is *Challenging*. The previous research work is resulted only limited number of techniques for mitigating CSRF vulnerabilities such as proxy solution, filtering the contents of webpage, using cookies. Such techniques involves moreover much work to perform the task and lengthy process. From the browser perspective Same Origin Policy is not enough to protect against the CSRF attack. Because mostly web applications are using Cross Origin Policy. But even if Cross Origin Policy may not be configured acceptably which can cause the defense or prevent ineffective.

To protect web applications against Cross Site Request Forgery (CSRF) attack, this research work have proposed and implemented ***Hybrid Approach*** which we named as ***CSRF Gateway***. Below are the key features of CSRF Gateway.

Here is the basic flow diagram of the proposed solution.

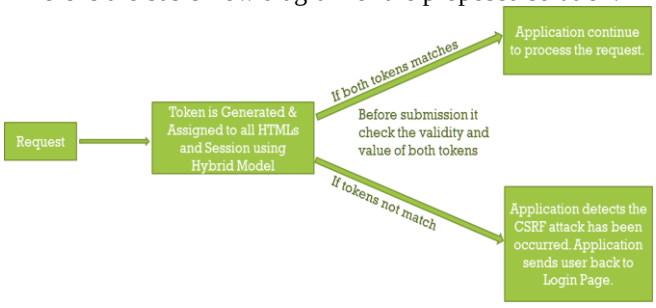

Figure: 3. Basic flow diagram of proposed solution.

A. Server Side Protection

CSRF Gateway provides Server Side Protection from Cross Site Request Forgery Attack. This solution gets installed on the web application. Server side protection is stronger protection approach for control and behavior than client side protection strategies. When the *HTTP Request* by the user then on the server side, web application creates the session and embeds *token* to the *Session* using *Custom Tag Library* which provides the more secured way to insert the token. And in all the forms by newly created *Custom Tag <CSRFToken>*.

B. Angular JS Anonymity

CSRF Gateway also has ***Angular JS Anonymity***, which sends the *CSRF Token* anonymously using *Angular JS and AJAX* in the inner HTML pages like "Add", "Update", "Delete" while *Submitting* the *HTTP Request* to the server without refreshing and reloading the web page with updates. This *Anonymity* creates the secure traversing of the *HTTP Request* to the server. This makes even more difficult for the attacker to speculate the flow of the request and the parameters. There is no other way in which attacker can predict the supported elements of the attack. Even if the attacker is able to predict the knowledge of request then also request cannot be changed or modified maliciously. In addition, *Angular JS Anonymity* enhances the *Performance* of the web solution. When the *HTTP Requests* are traversing so many times to and fro then sometimes server gets overloaded by loading of contents every time, in that case if the Request is very Light Weighted, it gets go over the server very fast to enhance the performance.

C. Token Generation

The key reason for the success of CSRF attack is that attackers can obtain all the parameters of the important operations by analyzing victim request and website and then forge a valid request passing the server-side validation. Thereupon, by adding the hidden token parameter in the operations, (the value which generate random number) so that attackers cannot predict parameters value, and therefore forgery requests cannot pass validation. Using tokens is by far the most effective method to defend against CSRF attacks. CSRF Gateway uses ***Secure, Random and Unique 32-bit Alphanumeric Token***, which makes the token value impossible to predict for forging unauthorized request.

D. Token Insertion

CSRF Gateway uses ***JSP Custom Tag Library*** to insert Token in all the HTML pages. Custom Tag Library provides developers ***more granular control over Token Insertion***. This provides more secure way to embed token. More over this strategy is more useful than normal Java Script or any other Token Insertion method. ***CSRF Gateway has 2 layers of security*** in additional to traditional token insertion alone.

1. First Token embeds with the each HTML page.
2. Second Token embeds with the Session.

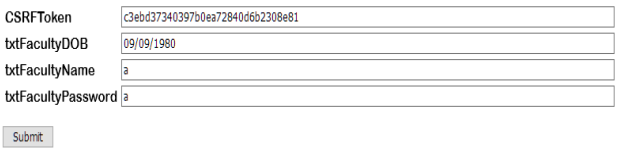

Figure: 4. Diagram of representation of CSRF Token.

V. IMPLEMENTATION

We have implemented a proof of concept of our proposed CSRF attack protection approach as a Server Side installation. An application administrator has to embed the solution to the application.

We have used Java for the development platform. Java Platform is dynamic, security architecture, standards-based and interoperable. This provides a safe and secure platform for developing and running applications. It includes enforcing runtime constraints through the use of JAVA VIRTUAL MACHINE, a security manager that sandbox untrusted code from the rest of the operating system, and a suite of security APIs that JAVA developers can utilize.

We have used Custom Tag Library to insert Random and Unique token to all the HTML pages. Custom tag library is a User defined JSP language element. When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler. The web container then invokes those operations when the JSP page's servlet is executed. In this way, *Custom Tag Library* provides the fine grain level of security. The advantage of the Custom Tag over any Java Script is that functionality is never been shown to the end user or attacker.

In the model, the ***Upper Layer embeds the first CSRF Token to the Session*** which gets assigned per session. Whenever user requests server for any page first time, session gets created. Server embeds token to the session and sends the request back to the user. This token remains same for the whole session. Then ***Middle Layer assigns second CSRF Token to all the HTMLs*** which gets assigned per HTTP request.

```
1. <HTML>
2. <BODY>
3. <FORM action="CRUDController" method="POST">
4. <INPUT type="hidden" name="CSRFToken" value="<csrf: token-value>" >
5. <INPUT type="text" name="name" value="" >
6. <INPUT type="text" name="email" value="" >

7. </FORM>
8. </BODY>
9. </HTML>
```

Figure: 5. Diagram of representation of CSRF Token insertion in HTML page.

In figure 5, it is shown that how this CSRF Token get inserted into the HTMLs. This CSRF token assigned to the *hidden* field, and the value of this token gets set in the *Custom Tag Handler*. Figure 6 shows how the CSRF token value gets displayed in the webpage.

```
<tr>
    <td>
        <br/>
    </td>
</tr>
<tr>
    <td>
        <input type = "hidden" name = "CSRFToken" value="2bb1d24972f13365291723834f744a81
">
    </td>
</tr>
<tr>
    <td>
        <br/>
    </td>
</tr>
<tr>
```

Figure: 6. Diagram of representation of CSRF Token displayed in HTML page.

Whenever the user tries to submit any HTTP request to the server then before submitting the request, server verifies the token associated with each request. If both the tokens matches then request get passed to the server, otherwise it assumes that the CSRF Attack has been occurred and then server Logout the user from the application. ***The Lowest Layer*** where ***Angular JS Anonymity*** plays the most vital

role which ***hides the HTTP Request parameters to get exposed in the request while submitting an important operations to the server by making an anonymous call.***

```
var postApp = angular.module("mainModule", []);

postApp.controller("cRUDController",[ '$scope', '$http', function ($scope, $http)
 {
    $http.defaults.headers.post["Content-Type"] = "application/x-www-form-urlencoded; charset=utf-8";

    $scope.sendPost = function()
    {

        $http({
            url : 'cRUDController',
            method : "POST",
            data :
            {
                'strStudentId' : $scope.strStudentId,
                'strStudentName' : $scope.strStudentName,
                'strSubjectId' : $scope.strSubjectId,
                'strGrade' : $scope.strGrade,
                'ctrlAction' : $scope.ctrlAction

            }

        })
```

Figure: 7. Diagram of representation of Angular JS Anonymity.

Unfortunately, there is no publicly available test suite for Cross Site Forgery Attack Protection evaluation. Thus, we developed the benchmarked test suite called "Student Grading System" to test the proposed solution. In this test suite, we first put the CSRF TOKEN in the hidden field, so that it will not be visible to the victim or attacker. Then we assign the other CSRF TOKEN to the session. Both the tokens are ***32-bit alphanumeric encrypted using SHA1 (32-bit) and MD5 Encryption Technique***. This application performs "Insert Faculty", "Add", "Update" and "Delete" student actions. We have put ***Interceptor Class*** which intercept each and every request and response. Hence at all the time it check for the tokens associated with session and request to be matched.

We have used ***OWASP Zed Attack Proxy (ZAP)*** which is one of the world's most popular free security tools. It can help developers automatically find security vulnerabilities in the web application while developing and testing

the application. In our work, we have used this tool to *Intercept the Request* and test our *Proof of Concept*. ZAP Proxy tool intercept each incoming and outgoing request. In figure 7, it is shown how any request can be forged while before submitting it on the server.

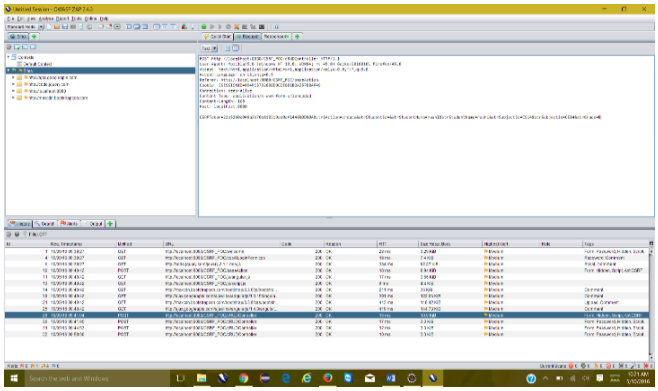

Figure: 8. Diagram of representation of HTTP Request on ZAP Proxy tool.

Here request can be forged and send back to the server. Figure 8 shows the response of the forged request returned by the server using ***CSRF Gateway***.

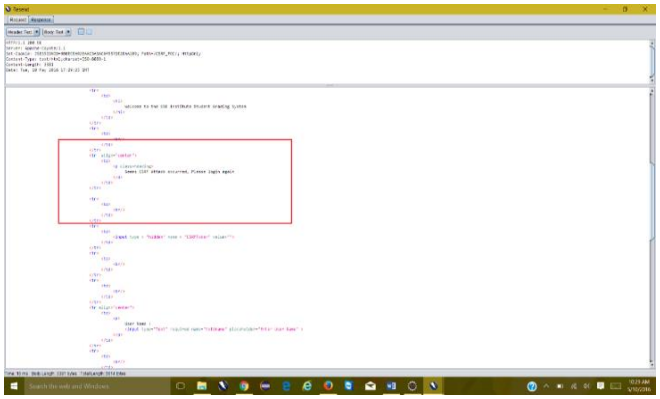

Figure: 9. Diagram of representation of Response returned by Server for forged HTTP Request on ZAP Proxy

VI. RESULTS & CONCLUSIONS

We have tested the applications with the proof of concept and we got results over previous research works. We have done the comparison with previous research work.

The new Hybrid technique is deployed for a test application to provide a detailed proof of concept for CSRF Gateway. The results obtained from the previous research works are compared with this gateway technique that is summarized below.

A. Server Side vs Client Side Protection

CSRF Gateway (our research work) provides Server Side Protection against CSRF against which is stronger and powerful than any Client Side Defensive Solution.

B. Light Weighted Solution

This is Light Weighted Solution. CSRF Gateway is very easy to install at Server Side. Server Side needs to embed the CSRF token to their application.

C. Synchronizer Token Pattern vs Other Defensive Technique

Synchronizer Token Pattern Defensive Technique is most Compatible, Reliable and Official Technique of Protection against Cross Site Request Forgery. (Source OSWAP Site)

D. Encryption Technique

MD5 combined Encryption Techniques used to generate Unique and Random 32-bit Alphanumeric Token. JSP Custom Tag Library is used to embed the Token into the HTML forms, which gives the granular Control over Token Injection. It is more useful strategy over normal Java Script or any other strategy used for Token Injection. AJAX and Angular JS plays an important role to give the great performance because it anonymously all the request parameters to the server. This is very useful for new web services.

This project has implemented keeping an overall efficiency and performance as key factors to cover hidden tags as more secure form of post authorization in a CSRF attack scenario. CSRF Gateway has been designed and implemented to provide robust protection solution against Cross Site Request Forgery using the latest technology for web development that can greatly change the way the traditional proxy based CSRF Gateway was implemented that could itself be a performance throttle for application itself. The solution will become more secure with secure HTTPS transactions to avoid any eavesdropping to ensure the passive data collection are also prevented for user profiling.

	Application without Protection	**Application Protected using CSRF Gateway**
GET Request without Malicious Link	Protected	Protected
POST Request without Malicious Link	Protected	Protected
GET Request with Malicious Link	Not Protected	Protected
POST Request without Malicious Link	Not Protected	Protected
Response Content Check	Not Protected	Protected
Reflected CSRF	Not Protected	Protected
Stored CSRF	Not Protected	Protected

Table: 1. Requests With & Without Protection in Applications

VII. FUTURE WORK

Cross Site Request Forgery (CSRF) attack is a website exploit type of attack. Even though it is very less known to the web developers. As far, we have seen that CSRF Gateway is able to protect against CSRF attack only. In future, we will elaborate this solution to extend to defend the web applications against other threats like Cross Site Scripting

(XSS), SQL Injection, and Session Hijacking, Broken Authentication or other less known web attacks.

This Gateway Strategy can be extended to more features and functions for specific web security against server side malicious code detection and protection.

ACKNOWLEDGEMENTS

Presented thesis work was supported and guided by the faculty members of College of Science & Engineering. We thank you the members of the Silicon valley team in the Bay area who have conducted many security related discussions every month and participating and discussing these methodologies with them gave a new focus and understanding of this project implementation. Also I thank a lot to my project guide and coordinator whose feedback and constant discussion have helped me improve the presentation to provide more detailed feedback.

REFERENCES

[1] Jovanovic, N.; Kirda, E.; Kruegel, C., "Preventing Cross Site Request Forgery Attacks, Securecomm and Workshops, 2006, vol., no., pp.1, 10, Aug. 28 2006-Sept. 2006 doi:10.1109/ SECOMW. 2006.35953

[2] Alexenko, T.; Jenne, M.; Roy, S.D.; Wenjun Zeng, "Cross-Site Request Forgery: Attack and Defense," Consumer Communications and Net working Conference (CCNC), 2010 7th IEEE, vol., no., pp.1, 2, 9-12 Jan. 2010 doi:10.1109/ CCNC.2010.542178

[3] Shahriar, H.; Zulkernine, M., "Client-Side Detec tion of Cross-Site Request Forgery At tacks," Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, vol., no., pp.358, 367, 1-4 Nov. 2010 doi:10.1109/ISSRE.2010.12

[4] Open Source Vulnerability Database (OSVDB), Accessed from http://osvdb.org.

[5] G. Zuchlinski, "The Anatomy of Cross Site Scripting", November 2003.

[6] Siddiqui, M.S.; Verma, D., "Cross site request forgery: A common web application weakness," Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, vol., no., pp.538, 543, 27-29 May 2011 doi: 10.1109/ICCSN.2011.6014783

[7] Yin-Chang Sung; Cho, M.C.Y.; Chi-Wei Wang; Chia-Wei Hsu; Shieh, S.W., "Light-Weight CSRF Protection by Labeling User-Created Contents," Software Security and Reliability (SERE), 2013 IEEE 7th International Conference on , vol., no., pp.60,69, 18-20 June 2013 doi:10.1109/SERE.2013.22

[8] https://www.owasp.org/index.php/Cross-Site_Request_Forgery (CSRF)

[9] http://www.acunetix.com/websitesecu rity/csrf-attacks/

[10] http://www.veracode.com/security/csrf

[11] https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Gallagher.pdf

[12] http://www.toolswatch.org/2016/02/2015-top-security-tools-as-voted-by-toolswatch-org-readers/

[13] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

[14] https://angularjs.org/

[15] http://docs.oracle.com/javaee/6/tutorial/doc/bnawn.html

[16] Chen, Boyan; Zavarsky, Pavol; Ruhl, Ron; Lindskog, Dale "A Study of the Effectiveness of CSRF Guard", *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, On page(s): 1269 - 1272