Building a Robust Client-Side Protection Against Cross Site Request Forgery

Abdalla AlAmeen
College of Art and Science- Prince Sattam bin Abdulaziz University

***Abstract*—In recent years, the web has been an indispensable part of business all over the world and web browsers have become the backbones of today's systems and applications. Unfortunately, the number of web application attacks has increased a great deal, so the matter of concern is securing web applications. One of the most serious cyber-attacks has been by cross site request forgery (CSRF). CSRF has been recognized among the major threats to web applications and among the top ten worst vulnerabilities for web applications. In a CSRF attack, an attacker takes liberty be authorized to take a sensitive action on a target website on behalf of a user without his knowledge. This paper, providing an overview about CSRF attack, describes the various possible attacks, the developed solutions, and the risks in the current preventive techniques. This paper comes up with a highly perfect protection mechanism against reflected CSRF called RCSR. RCSR is a tool gives computer users with full control on the attack. RCSR tool relies on specifying HTTP request source, whether it comes from different tab or from the same one of a valid user, it observes and intercepts every request that is passed through the user's browser and extracts session information, post the extracted information to the Server, then the server create a token for user's session. We checked the working of RCSR extension, our evaluation results show that it is working well and it successfully protects web applications against reflected CSRF.**

***Keywords*—Security; Reflected CSRF; client-side protection; tab ID; token**

I. INTRODUCTION

In recent years, the web has been an indispensable part of business all over the world and web browsers have become the backbones of today's systems and applications. Unfortunately, the number of cyber-attacks has increased a great deal, so the matter of concern is securing web applications. One of the most serious attacks has been called cross site request forgery (CSRF). CSRF is also known as XSRF, Session Riding, One-Click-Attack, and Confused Deputy [3]. In a CSRF attack, an attacker takes liberty be authorized to perform a sensitive action on a target website on behalf of a user without his knowledge.

CSRF attacker takes advantages of implicit authentication mechanisms of HTTP protocol and cached credentials in the browser to inject web applications with malicious script [17]. The malicious script may destroy the privacy of the user's session with a web application. CSRF attack tricks user's browser into performing requests into a target web site that is vulnerable to CSRF [4]. A website is vulnerable to CSRF attack when it has inadequate mechanism to check whether a valid request has been sent intentionally or unintentionally by a logged in user [15]. A CSRF attack involves three actors as shown in Fig 1, a user, a trusted website, and a malicious website. To perform a CSRF attack, the user must hold an active session with the target site [13]. Suppose the victim user is authenticated (a logged in user), the attacker can upload HTML element or JavaScript code on a third-party website, subsequently the victim user visits an attacker controlled third-party website or he/she clicks on a link in the same web browser (without logging out form the trusted website). Thus, the attacker malicious script will be executed without the victim user being aware of it. Attacker uses illegal strategies to deceive the victim to send unintended request [2]. For instance, an attacker may attract browser's user into clicking on a malicious link or image, which is hosted on untrusted third party server or he/she can post a message in a social website, this message may contain malicious image tag as shown in Listing 1.

```
<img src="http://mybank.com/withdraw?
account=Sender&amount= amount-&for= reciever ">
```

Listing 1. Image tag containing a malicious Code snippet

As shown in Listing 1, the attacker may send an image tag a third-party website, that contains a request to perform a sensitive action (withdraw money) on a trusted-website of an authenticated user (mybank.com), probably without their knowledge.

In the early appearance of World Wide Web (WWW) in 1989 [12], it only contains a set of static pages interconnected via hyperlinks. But when images were added to web pages in 1993 [12], a request to a web page could cascade a set of requests to multiple other web pages. Thus, cross-site or cross-origin requests triggered without explicit user interaction. With the coming of interactive web thought Java scripts and Web forms in 1995 [10], cross-site interactions become a real security threat to web applications.

Typically, today's websites implement cookies to identify authenticated users [1]. After the user is successfully authenticated by the Web server, the browser will get an identity login cookie to remember the logged-in status [10]. Later, when the user is visiting the Web pages of the target website, the browser will automatically attach the identity login cookie in the HTTP request [10]. This cookie will not be removed until the browser is closed or the user is logged out. The attacker is able to abuse this duration to make some user's browser perform authenticated requests probably without their knowledge, and that is what is called cross site request forgery CSRF.

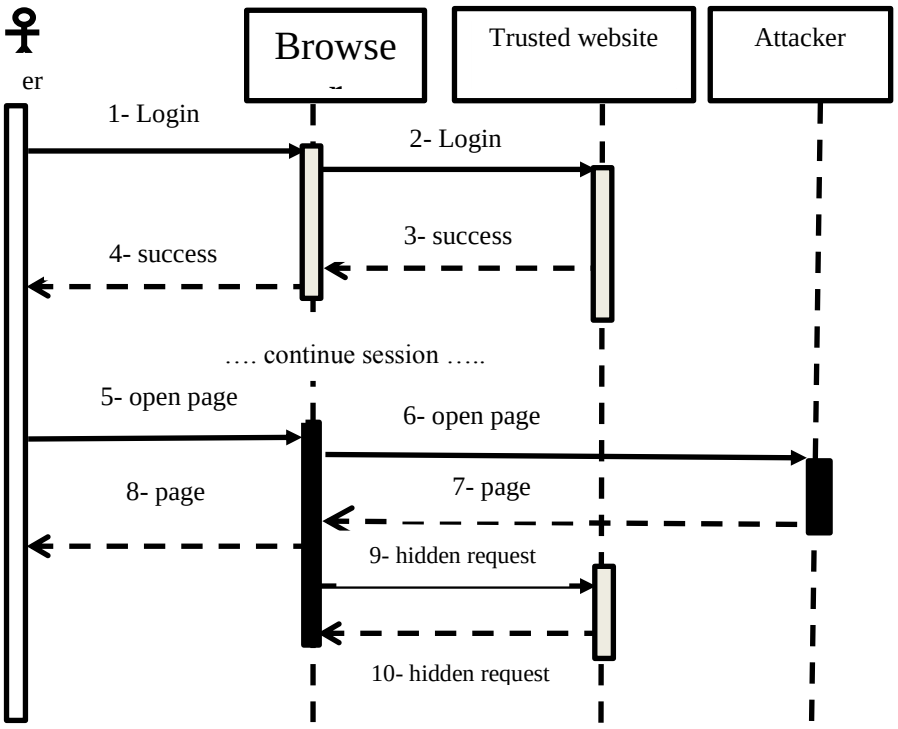

Fig. 1. Simple CSRF attack scenario

A number of serious CSRF vulnerabilities in some websites were documented [7], which allowed an attacker to transfer funds from victim's account to an account chosen by the attacker [7] as shown in listing 1. What makes detection or prevention of CSRF attack so difficult is the fact that web applications look to all requests triggered from an authenticated user's browser just like another. Since the requests are being made directly to the real Web application (no man-in-the middle) therefore the unintended malicious requests are considered legitimate in server perspective: "The only problem is the victim did not intend to make the request, but the Web server does not know that" [12]. The majority of web application are vulnerable with users having very little ability to defend themselves against CSRF" [12].

One of the primary causes of CSRF attacks is the misuse of cached credentials in cross-domain requests [7]. The attacker can easily send some requests to web applications in another trusted web site without the user involvement and knowledge. This makes web browser send cross-site requests, while implicitly using cached credentials in web browser [7].

CSRF attacks are as powerful as a user. Whatever action that the user can do can also be done by an attacker using a CSRF attack. Thus, the more rights a site gives to a user, the more dangerous are the possible CSRF attacks. The seriousness of CSRF attack comes from the fact of malicious request arriving from authenticated user. For instance, if the account of the target has full rights, this can destroy the overall web application. However, if we can understand all the steps in which Web applications are attacked via CSRF attacks, we can design countermeasures to thwart it. Moreover, if we know who the attackers are, and what they want, their goals, motivations and abilities we will have to educate users to protect themselves from CSRF attacks.

The main aim of this paper is to follow preventive techniques in order to make web application more secure than it is at present .This paper, however, provides an overview about CSRF attack, the various possible attacks, the developed solutions, and the risks in the current preventive techniques. This paper comes up with a highly perfect protection mechanism against reflected CSRF. RCSR is a tool that gives computer users full control on the attack. RCSR tool relies on specifying HTTP request source whether coming from different tab or from the same one of a valid user. RCSR observes and intercepts every request that is passes through the user's browser. RCSR extracts the session information such as tab ID, IP address, then post the extracted information to the web server, the server creates a token for user's session to validate the legitimacy of the request before changing any sensitive data in the server database. We have checked the functioning of RCSR extension, our evaluation results shows that it is working well and it successfully protects web applications against reflected CSRF.

The remainder of this paper is structured as follows: Section 2 describes the main concepts of CSRF and the processes involved in the attacks. Section 3 describes the existing protection and prevention techniques against CSRF. Section 4 focuses entirely on the development of tokens concepts as a standard defence mechanism against CSRF. Section 5 summarizes some existing defence's techniques and their attributes. Section 6 presents RCSR, our proposed scheme, section 7 describes the implementation of RCSR. In section 8, we extensively validate the efficiency and the

capability of the RCSR tool against reflected CSRF attack and finally in Section 9 we conclude this paper.

II. Cross Site Request Forgery

The launching of CSRF attack may be carried out in different steps depending on the type of CSRF attack. CSRF can mainly be classified into two types: reflected and stored [14]. First of all, the attacker should know the structure of the website request forms, then check the main functionality of targeted web site. A professional attacker may perform that manually or by searching the web using specific software tools. Toolkits such as seobook, webconfs and web spider are the software available on the web for free .They can be used for displaying the contents of a web-page and its functionality. Secondly, the attacker will specify specific functionality in the web-page that it can be used to perform malicious actions on behalf of a victim user. Then, the attacker will send a parameterized request. Some network protocol analyser such as Wireshark, Cain & Abel and Tcpdump can be used to examine data from a live network and browse the captured data that may contain buttons or links that can perform actions. The following step is to create a malicious link that can send this legitimate HTTP request to the website and will execute some interesting functionality on the server such as transferring money, changing a password, etc. Finally, the attacker needs to convince a logged in user into the target website to click on the malicious link to execute the CSRF attack successfully.

For launching reflected CSRF attacks, the attacker needs to include the malicious link on the attacker's controlled website and trick the user to click on the link, or where an XMLHTTPRequest object may automatically execute the attack when a user visits the website [14].

For stored CSRF attacks, the attacker needs to create some posts that embed the malicious link in the target website, or execute a stored XSS attack on a website where an XMLHTTPRequest object will automatically execute the attack as soon as a user visits the page [14]. This removes the step of convincing a user to click on a link.

III. Existing Countermeasures

To overcome CSRF attacks, a variety of techniques are available to protect server applications and the end-users from CSRF attack [7]. CSRF protection and prevention techniques can be classified into two main categories:

1) *Client side protection techniques*
2) *Server side protection techniques*

Client side protection techniques can be used to protect users from CSRF attacks by monitoring outgoing requests and incoming responses. Client side protection techniques can be implemented as a browser proxy (plug-in or extension) to web browsers [19]. Browser extension is the technique that we have adopted in this paper as shown in section (6).

The basic idea behind the Server side protection techniques is that server can strip authentication credential and session information from suspected requests, or it can refuse such requests. Using validation of secret token and checking HTTP Referrer header are the most applied Server side protection techniques [7]. Unfortunately, not any of the proposed mechanisms is fully capable of carrying out this task, in other words the existing solutions are time-consuming, error-prone, and not immune to avoid CSRF attacks.

IV. CSRF Tokens Concept

In the early appearance of World Wide Web in 1989 [12], it only contained a set of static pages interconnected via hyperlinks. But when images were added to web pages in 1993 [12], a request to a web page could cascade a set of requests for other multiple web pages. Thus, cross-site or cross- origin requests triggered without explicit user interaction. With the coming of interactive web thought Java scripts and Web forms in 1995 [10], cross-site interactions has become a real security threat to web applications.

Typically, today's websites implement cookies to identify authenticated users [1]. After the user is successfully authenticated by the Web server, the browser will get an identity login cookie to remember the logged-in status [a10]. Later, when the user is visiting the web pages of the target website, the browser will automatically put the identity login cookies in the request [a10]. This will not be removed until the browser is closed or the user logged out.

CSRF vulnerabilities arise because the browsers send the cookies back to the Web server automatically with each subsequent request. If Web applications relied solely on cookies as a mechanism to keep track of user sessions, they will be at risk for this type of attack [12].

The attacker is able to abuse this duration to make the user's browser perform authenticated requests probably without their knowledge, and that causes what is called CSRF.

To create a standard defence mechanism against CSRF attack, we must support cookie-based and HTTP authentication with additional means to keep track of sessions. These additional means may be as additional tokens that are transmitted through hidden fields with any request to the server [12].

When the server receives a request, in addition to the process of verifying the validity of session cookies, it also verifies that the received token is valid for the current user, else the request will be rejected. If we assume that the attacker does not have the ability to know the value of this token, so he won't be able to put the right token in its submissions, therefore he does not have the ability to launch a successful CSRF attack. When we use CSRF tokens in this way, they must be subject to adequate protection because they are considered as sensitive data [12]. If the attacker can predict the value of CSRF tokens that have been sent to another user, so he can obtain a valuable data to perform malicious action on behalf of the user.

V. Literature Review

To overcome CSRF attacks, a variety of defense techniques exist, these protections and prevention schemes propose to make forgery requests harder for adversaries, or to confirm the origin of page requests. By assuring the integrity of requests' origins, defense techniques can ignore page requests coming from the cross site domains because web transactions are usually intended for the requests initiated from the same

domain, not the cross sites. Most defense schemes can be classified into two main types, Client side techniques and Server side techniques. Client-side defense protects users even if web application doesn't prompt in fixing their vulnerabilities. Moreover, they have accurate information about the requests sources, whether they result from clicking on a link, or a bookmark on a trusted web page. There are several Client- side protection and prevention schemes (Secret Validation Token, RequestRodeo, CsFire etc.) to prevent CSRF attack.

A. Secret validation token

Secret Validation Token is a well-known client- side protection scheme against CSRF attacks. Not like other verification, token approach does not require user intervention, so the users will not know that something has been used to protect them. This scheme sends additional information with each HTTP request to determine whether the request came from an authorized user. To apply token, web applications must first create a "presession,", and then proceed forward a real session after successful authentication [18]. Validation token should be hard to guess for attacker who does not already have access to the user's account. If a request is missing a token or the token does not match the expected value, the web application should reject the request and prompt the user [18].

One disadvantage of using the Secret Validation Token is that, occasionally, some users disclose the contents of web pages they view to third parties, for instance via email . If the page contains the user's Validation Token, anyone who views the contents of the page can impersonate the user to the web application until the session expires [19].

B. Requestrodeo

Johns and Winter proposed RequestRodeo as a client-side protection proxy against CSRF [11]. RequestRodeo lies, next to cookie-based HTTP authentication. This technique offers protection via detecting cross-domain requests and then removal of cookie values from these requests (stripping of implicit authentication) [11]. A request is authenticated when it satisfies the Same Origin Policy (SOP) and it initiated as a result of an interaction with the currently viewed browser' tab. RequestRodeo is limited to only certain HTTP requests and no HTTPS requests, so it does not scale well to web 2.0 applications. RequestRodeo fails to detect all JavaScript dynamic links in the responses, since this dynamic content has come after passing through the proxy. Also, RequestRodeo does not differentiate between malicious and genuine cross origin requests, so it provides very poor protection against CSRF [16].

C. CSFIRE

CsFire [8] is integrated extension into Mozilla browser to mitigate CSRF attacks, it extends the work of Maes et al. [8], CsFire is the only system that provides formal validation through bounded model checking to defend against CSRF in the formal model of the web developed by Akhawe et al. [8]. CsFire strips cookies and HTTP authorization headers from a cross-origin request. The advantage of stripping cookies and HTTP authorization headers is that there are no side-effects for cross-origin requests that do not require credentials in the first place.

Additionally, CsFire supports users for creating custom policy rules, which use user-supplied whitelist and blacklist to certain traffic patterns. Furthermore, CSFire utilizes a sophisticated heuristic to identify legitimate cross-domain requests which are allowed to carry authentication credentials [8]. The disadvantage of CsFire approach is that without the server supplied or user supplied whitelist, it will not be able to handle complex, genuine cross origin scenarios and the whitelists need to be updated frequently.

According to the defensive techniques discussed above, none of them is able to provide full protection. So it seems necessary to overcome the drawbacks of present defensive measures. We propose to develop a new client side defensive approach, in the form of a Firefox extension, to prevent Reflected CSRF attacks effectively as explained in section 6.

VI. THE PROPOSED SCHEME

The web browser is the right place to apply appropriate protection mechanism for web application because it is the first place to detect CSRF attack symptoms. So the proposed defense mechanism against reflected CSRF attacks should be applied on the client side in order to reduce the overtime efforts of web developers. Client side protection techniques can be implemented as a proxy or as plug-in (extension) to web browsers.

Mozilla is an extensible architecture, open source, and the second most popular browser, also Mozilla browser behaviour can be modified by creating appropriate XPCOM (Cross Platform Component Object Model) objects and implementing a set of APIs. Mozilla supports the global browser object called (gBrowser) to access the active tab windows and examine its ID through GetSelectedTab function. We propose to provide a robust client side defense mechanism against CSRF, hence named as "Robust Client Side Request" (RCSR). Once implemented on the browser, RCSR can be the best solution over other techniques to protect web applications against reflected CSRF. RCSR is a technology independent tool and does not depend on user input, so it solves the drawbacks of current protection techniques.

We designed the plug-in using JavaScript, which can be installed in Mozilla browser to protect users against reflected CSRF attacks. A user needs to enable CSRF from the tools menu of a browser after loading a page that needs to be monitored for attack detection.

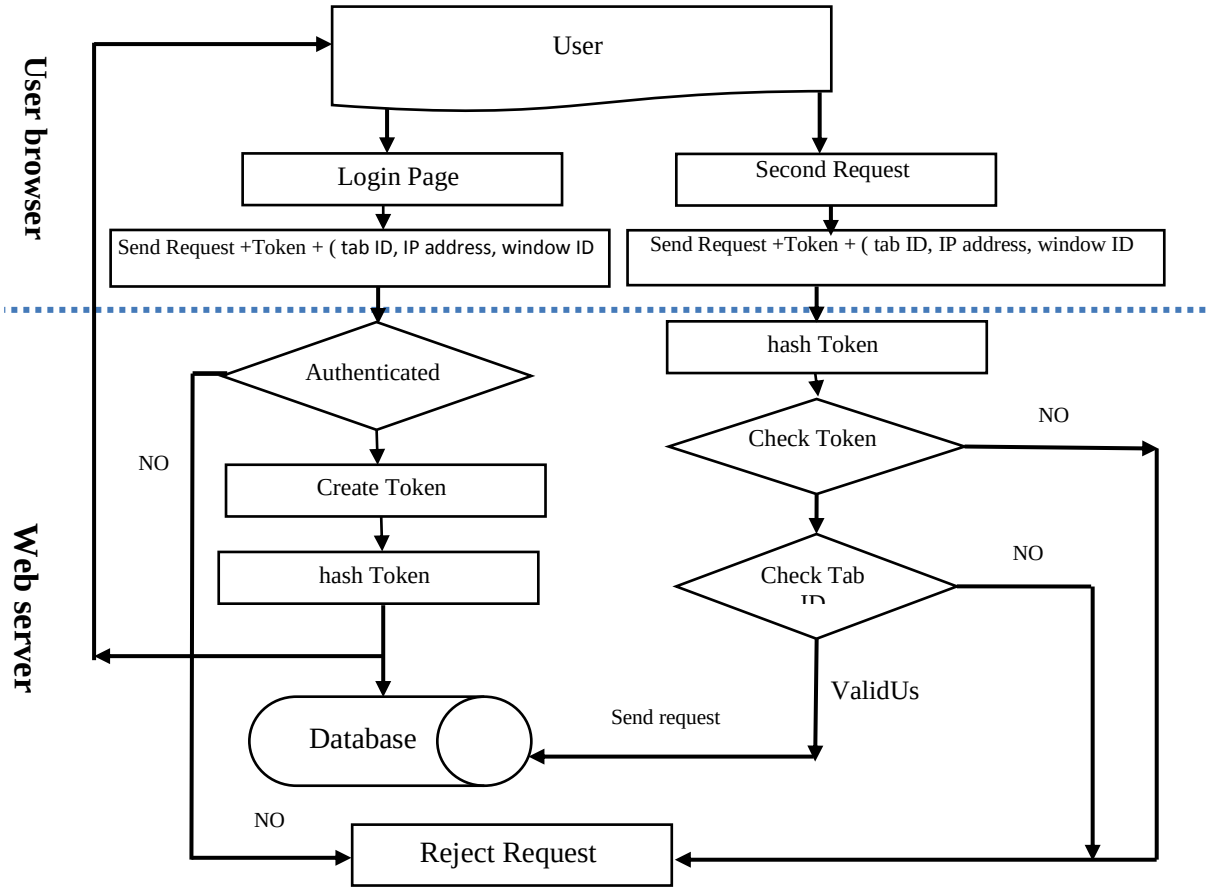

Fig. 2. The proposed scheme diagram

VII. IMPLEMENTATION

Our solution, RCSR tool a simple policy, is implemented in the form of a client-side plug-in to protect web applications against reflected CSRF. In general, RCSR allow the web application developer to plug in new functionalities to web browser.

The general mechanism of RCSR functions as follows:

To specify HTTP request source, whether coming from different tab or from the same one of a valid user, RCSR observes every request that is passes through the user's browser, intercepts HTTP requests and extracts session information. Listing 1 below show snippet code to extract the tab ID from web browser.

```
function RCSRObserver()
                                {
  this.register();
  this.windowIds = new Object();
                                }
RCSRObserver.prototype =
                                {
                observe : function(subject, topic, data)
                                 {
    var tabId = this.getTabIDfromDOM(httpChannel, subject);
                                if (tabId)
                                {
                var windowId = this.windowIds[tabId];
                                if ( ! windowId)
                                {
                this.registerTab(tabId);
                windowId = this.windowIds[tabId];
                                }
    httpChannel.setRequestHeader("Window-Id", windowId,
                                false);
                                }
                                }
    getTabIDfromDOM : function(aChannel, aSubject)
                                {
                registerTab : function(tabId)
                                {
    window.addEventListener("load",
                                function(e)
                                {
    observer = new RCSRObserver();
    var num = gBrowser.browsers.length;
    for (var i = 0; i < num; i ++ )
                                {
    var b = gBrowser.getBrowserAtIndex(i);
    observer.registerTab(b.parentNode.id, i);
                                }
```

Listing 2. Extracting tab ID Code snippet

To create a new token for the current user session, RCSR post the extracted information (tab ID, IP address, window ID) to the web Server.

The system will store the session information on the server database to map token with user's session or identity. The web server will send to the client with a unique token. After hashing the information by a cryptographic hash algorithm based on SHA-1 the web server will define the tab ID value. Then store the hashed value in database table. The web application can repeatedly validate the legitimacy of the attached tokens before changing any sensitive data in the server database.

Sever verifies if the request is generated from the same tab of the browser. This verification is performed by comparing the stored hash information with the hashing information that is sent with each request. The request will be executed if the comparison result is true, otherwise the session will be destroyed. Fig. 2 shows all steps of the RCSR detection tool.

VIII. Evaluation

We conducted some tests to evaluate the efficiency and the capability of the RCSR tool against reflected CSRF attack, to make sure that its results match with what is predetermined, to discover the problem and try to fix before the deployment.

PhpBB 3 is an open source discussion forum software. IT includes all the features in today's top of the line software written in PHP, and MySQL [9]. PhpBB 3 uses cookies to authenticate user's requests which are an important element in CSRF attack. Despite the fact that phpBB3 is popular application and well-maintained, but easily we discovered some CSRF vulnerabilities [9].

By exploiting CSRF attacks, we modified some important information through abusing of an authenticated user privileges. Through malicious link we could access the user cookies and valid session on the victim's browser, so we were able to send and delete some messages from the forum or even change user name and password on behalf of the victim user.

To evaluate the ability of RCSR to protect vulnerable applications, we installed RCSR tool as an extension to Mozilla browser. When we repeated the previous attacks, RCSR tool detected and rejected all CSRF attempt correctly.

While testing, that RCSR doesn't interfere with the normal application behaviour. We observed and compared phpBB3 application behaviour without the RCSR tool protection to the behaviour with enabled CSRF protection. The results were identical and the RCSR tool succeeded in performing its task transparently.

We tested some functionalities of the Mozilla browser after installing RCSR tool. For instance, we observed the correct behaviour of the Mozilla's "Back" and "Forward" button, which is a widely used convenience feature that must not be broken by CSRF protection.

To test RCSR performance, we observed no noticeable delay when interacting with the applications protected by RCSR.

IX. Conclusion

One of the most serious cyber-attacks has been by cross site request forgery (CSRF). CSRF has been recognized among the major threats to web applications and among the top ten worst vulnerabilities for web applications. In a CSRF attack, an attacker takes liberty be authorized to take a sensitive action on a target website on behalf of a user without his knowledge. To conclude this paper, we have discussed CSRF in different domains, the severity of the attack on the current web applications, the various possible CSRF attack and risks in the current preventive techniques. To overcome the drawbacks of present defensive protection, this paper proposed a new client side defensive tool (RCSR). RCSR is a Firefox extension, which can prevent Reflected CSRF attacks effectively. RCSR is a tool gives computer users with full control on the attack. RCSR tool relies on specifying HTTP request source, whether it comes from different tab or from the same one of a valid user, it observes and intercepts every request that is passed through the user's browser and extracts session information, post the extracted information to the Server, then the server create a token for user's session.

In a practical evaluation, the working of this extension was checked against reflected CSRF, the evaluation results show that it is working well. It successfully protects web applications against reflected CSRF. In future work we plan to extend the RCSR functionality against stored CSRF attacks and evaluate its performance to make it more powerful and accurate. Finally, we hope that RCSR tool will prove useful in protecting web applications.

Acknowledgment

This project was supported by the Deanship of Scientific Research at Prince Sattam Bin AbdulAziz University under the research project 2014/01/1660

References

[1] Barth, Adam, Collin Jackson, and John C. Mitchell. "Robust defenses for cross-site request forgery." In Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 75-88.

[2] Batarfi, Omar A., Aisha M. Alshiky, Alaa A. Almarzuki, and Nora A. Farraj. "CSRFDtool: Automated Detection and Prevention of a Reflected Cross-Site Request Forgery." (2014).

[3] Calafato, Andrew, and Kostantinos Markantonakis. "An analysis of the vulnerabilities introduced with the java card 3 connected edition." PhD diss., Msc thesis, Royal Holloway, University of London, 2012.

[4] Chen, Eric Y., Sergey Gorbaty, Astha Singhal, and Collin Jackson. "Self-exfiltration: The dangers of browser-enforced information flow control." InProceedings of the Workshop of Web, vol. 2. 2012.

[5] Chin, Erika, Adrienne Porter Felt, Kate Greenwood, and David Wagner. "Analyzing inter-application communication in Android." In Proceedings of the 9th international conference on Mobile systems, applications, and services, ACM, 2011, pp. 239-252.

[6] Czeskis, Alexei, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. "Strengthening user authentication through opportunistic cryptographic identity assertions." In Proceedings of the 2012 ACM conference on Computer and communications security, ACM, 2012, pp. 404-414.

[7] De Ryck, Philippe, Lieven Desmet, Thomas Heyman, Frank Piessens, and Wouter Joosen. "CsFire: Transparent client-side mitigation of malicious cross-domain requests." In Engineering Secure Software and Systems, Springer Berlin Heidelberg, 2010, pp. 18-34.

[8] De Ryck, Philippe, Lieven Desmet, Wouter Joosen, and Frank Piessens. "Automatic and precise client-side protection against CSRF attacks." InComputer Security–ESORICS 2011, Springer Berlin Heidelberg, 2011, pp. 100-116.

[9] Fong, Matthew, Herman Lee, Chih-Hao Lin, and David Yue. "Security Analysis of phpBB3 Bulletin Board Software." (2010).

[10] Hall, Marty, and Larry Brown. Core Web Programming. Prentice Hall PTR, 2001.

[11] Johns, Martin, and Justus Winter. "RequestRodeo: Client side protection against session riding." In Proceedings of the OWASP Europe 2006 Conference. 2006.

[12] Kappel, Gerti, Birgit Pröll, Siegfried Reich, and Werner Retschitzegger. Web engineering. John Wiley & Sons, 2006.

[13] Poyar, Ryan L. "Cross-site request forgery attacks against Linksys wireless routers." (2010).

[14] Shahriar, Hossain, and Mohammad Zulkernine. "Client-side detection of cross-site request forgery attacks." In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, IEEE, 2010, pp. 358-367.

[15] Singh, Nanhay, Achin Jain, Ram Shringar Raw, and Rahul Raman. "Detection of Web-Based Attacks by Analyzing Web Server Log Files." In Intelligent Computing, Networking, and Informatics, Springer India, 2014, pp. 101-109.

[16] Telikicherla, Krishna Chaitanya, Venkatesh Choppella, and Bruhadeshwar Bezawada. "CORP: A Browser Policy to Mitigate Web Infiltration Attacks." InInformation Systems Security, Springer International Publishing, 2014, pp. 277-297.

[17] Wedman, Shellie, Annette Tetmeyer, and Hossein Saiedian. "An analytical study of web application session management mechanisms and HTTP session hijacking attacks." Information Security Journal: A Global Perspective 22, no. 2 (2013), 55-67.

[18] Xing, Xinyu, Elhadi Shakshuki, Darcy Benoit, and Tarek Sheltami. "Security analysis and authentication improvement for ieee 802.11 i specification." InGlobal Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, 2008, IEEE, pp. 1-5.

[19] Zeller, William, and Edward W. Felten. "Cross-site request forgeries: Exploitation and prevention." The New York Times (2008): 1-13.