Defensive Mechanisms of CSRF Attack

[1]Nikunj .J. Tandel, [2]Prof. Kalpesh .M. Patel

Computer Department, L.D. College of Engineering

[1]nikunj_tandel@ymail.com, [2]kalpeshpatel11@yahoo.com

***Abstract*—Web application is now part of our day to day life, and there are vulnerabilities in web application as per OWASP (Open Web Application Security Project). Most of the Web developers are unaware about CSRF attack and therefore many web applications are vulnerable from CSRF. Cross-Site Request Forgery (CSRF) attacks occur when a unauthentic web site forces a browser on user's computer to send an authorized request and perform unwanted action on a trusted web site without the user's knowledge. In this paper we will study about CSRF attack, and existing CSRF defensive mechanism. This study will help us to build strong and robust CSRF defensive mechanism.**

***Keywords*— OWASP, Web Application Vulnerabilities, Cross-Site Request Forgery, Defensive Mechanisms**

I. INTRODUCTION

Web applications are one of the most prevalent platforms for information and service delivery over the Internet today. As they are increasingly used for critical services, web applications have become a popular and valuable target for security attacks. Its objective is to establish rules and measures to use against attacks over the Internet. Web security deals specifically with security of websites, web applications and web services.

Cross-Site request forgery (CSRF) is an attack against Web application users where an attacker causes victim's web browser to perform an unwanted action on a trusted web site without the awareness of user, via a link or other content. CSRF is listed among the top **ten** web application vulnerabilities of 2013 [1]. Cross-Site request forgery (CSRF) is also known as session riding, one click attack and confused deputy [2]. The nature of attack is that CSRF exploits the trust that a web application for a user. CSRF is a vulnerability which works like a web works, due to the fact that a CSRF attack occurs when loading HTML elements or JavaScript code into a victim's browser that generates a legitimate HTTP request to a target website [2].

This paper organized as follows: Section II briefly discuss about Cross-Site request forgery attack. Section III describes Cross-Site Request Forgery (CSRF) versus Cross-Site Scripting (XSS). Section IV discusses existing defensive Mechanism. Section V draws some conclusions.

II. CROSS-SITE REQUEST FORGERY

There are two types of CSRF attack.

1) Stored CSRF
2) Reflected CSRF

Stored CSRF: A stored CSRF attack is one where the attacker can use the application itself to provide the victim the exploit link, or other content which directs the victim's browser to perform attacker-controlled actions in the application. Stored CSRF vulnerabilities are more likely to succeed, since the user who receives the exploit content is almost certainly currently authenticated to perform actions.

Reflected CSRF: In Reflected CSRF attack, the attacker uses a system outside the application to expose the victim to exploit link or content. This can done using a blog, an email message, an instant message, or even an advertisement posted in a public with an URL that a victim types in.

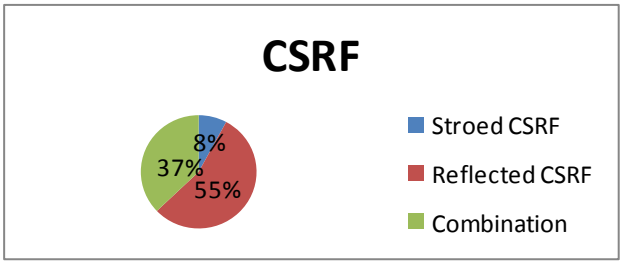

Figure 1: CSRF Attacks Categories until 2009 on NVD [3]

Working procedure of CSRF

Figure 2 and 3 show the working procedure of CSRF attack:

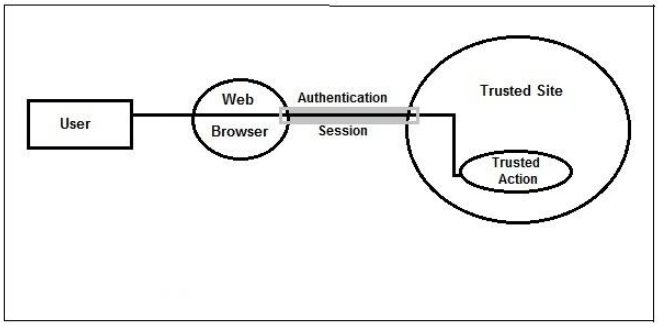

*Figure 2: A **Valid** request* [4]

The web browser attempts to perform a trusted action. The trusted site confirms that the web browser is authenticated and allows the action to be performed.

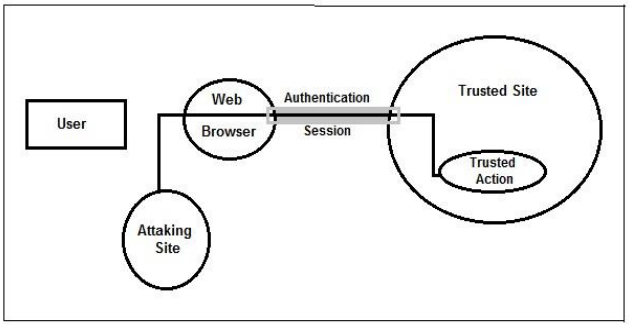

*Figure 3: A **CSRF** attack* [4]

The attacking site causes the browser to send a request to the trusted sites. The trusted site sees a valid, authenticated request form the web browser the trusted action. CSRF attacks are possible because web sites authenticate the web browser, not the user.

Login CSRF

Login CSRF is an another type of CSRF attack, in which an attacker uses the browser of the user to forge a cross-site request to the trusted site's login URL, giving the attacker's credentials. If attack succeeds, the trusted server responds with the Set-Cookie header that instructs the browser to change its current state by storing a session cookie & logging the user into the trusted site as the attacker. This session cookie is used to bind subsequent requests to the current session of the user and hence to the attacker's authentication credentials. Many web sites, including Google, PayPal, and Yahoo, were vulnerable to login CSRF.

Search History. Many search engines, including Google and Yahoo, allow their users to save their search history and provide an facility for a user to review his/her search history. Search queries contain private & sensible details about the user's interests and activities and thus could be used by an attacker to confuse the user, to steal his/her own identity, or to confuse the user. An attacker can spy on a search history of user by logging into the search engine as a user & originally working as the attacker.

III. CSRF Vs XSS

Cross-Site Request Forgery (CSRF) exploits the trust that the site has for the user. The website assumes that if an 'action request' was performed, it trusts that the request is being sent by the user. In contrast, Cross-Site Scripting (XSS) exploits the trust that a client has for the website or application. Users generally trust that the content displayed in their browsers was indented to be displayed by the website being viewed.

Simple example of CSRF could be a hacker can send you mail telling you won a grand prize and to claim it click on a link in his mail. You click and in case you are having persistent (though not necessary) authentication cookie from site that hacker want to manipulate, hacker would latch on it, use your credentials and send a HTTP request to that site. Remember the way browsers work is whenever you send a request for a specific domain also the cookies associated to that domain are also send across. The malicious scripts in turn gains access to page content and start misusing it. A simple example of XSS could be someone entering a malicious JavaScript function in comments section of a webpage. When other users try to fetch that page they would also fetch malicious JavaScript and that can be devastating.

IV. Existing Defensive Mechanisms

There are few mechanisms a site can use to defend itself against CSRF attacks: Validation a secret token, validating the HTTP Referer header, and Origin header. None of these mechanisms completely defend against CSRF attack.

A. Secret Validation Token:

Most popular approach to defend against CSRF attacks is to send an additional information in each HTTP request that can be used to find whether the requests came from an authorize user. This validation token should hard to guess for an attacker who

does not have an access to the user's account. If a request is missing a token or the token does not match the token value, the server will reject the request.

Random form tokens: for protection of CSRF attack, a web application has to ensure about that the data coming from the web application is originated from a valid HTML form. "Valid" in this context indicates the fact that HTML form was submitted by the actual web application. It also has to be ensured that, for submitting client, the HTML form was generated. To causes these requirements, random values are set for hidden form elements. These values are used as one time validation token: The triplet includes three things which are, form's action URL, the Session ID and the random form token are stored by the web application. Whenever form data is submitted, the web application is check for valid token, if it found the invalid token, it just denied the foreign request.

Using explicit authentication: There are some functions to communicate with authentic tokens explicitly, the Authenticated tokens can in the web application's URLs or transferred via hidden form fields in HTML forms. These techniques are prevention to CSRF attacks.

Drawback: The drawback of this approach is the more amount of manual work that it involves, in this approach, have to add token in to every single webpage. Many of current web applications have been developed into large, huge and complex systems, and combining them with the mechanisms necessary for token management would require detailed and wide application-specific knowledge and large amount of modifications to the application source code. After modification, it may possible that the developer's modified code is vulnerable to CSRF Attack.

B. **The Referer Header :**

An HTTP request's Header Referer indicates the URL of the webpage which contained the HTML link or form that was responsible for the request's creation. The Referer is communicated via an HTTP header. If the Referer Header present, it differentiates the original request from the cross site request because it contains the URL of the Request. A site can protects against cross-site request forgery attacks by checking whether the request was issued by the site itself or not. If this is not the matching case, the request is usually rejected.

Unfortunately, the Referer Header contains sensitive information that affects on the privacy of web users [5]. For example, if the contents of the search query are misguiding the user to visit a particular site then Referer Header will inform user. Although this information is useful to the owner of web site, by this information owner can optimize their search engine ranking, this information tell users to feel their privacy may violate. Additionally, many organizations are only concerned with the confidential information about their corporate intranets; those might leak information to the external web sites via the Referer header.

Bugs: When we set the proxy server, in that case browser have contained vulnerabilities that allow malicious web sites to spoof the value of Referer Header. Discussions of Referer spoofing often cite [5] as evidence that, browsers permit the Referer header to be spoofed. Mozilla has patched the Referer spoofing vulnerabilities in Firefox 1.0.7. These vulnerabilities affect only XMLHttpRequest and can be used to spoof Referers directly back to the attacker's own site.

Strictness: If a site selects to use the Referer header to defend against CSRF attacks, the site's developers should decide whether to implement lenient or strict Referer validation.

- *in lenient Referer validation*, if the Referer header has incorrect value the site blocks the requests. If a request header is not there then, the site accepts the request. Although widely implemented, lenient Referer validation is easily make fooled because a web attacker can force the browser to suppress the Referer header. For example, requests issued from data URLs and FTPs do not carry Referer headers.
- In *strict Referer validation*, If Request from browser doesn't contain Referer Header then site blocks the Request. If blocking requests that doesn't have a Referer header that prevents against malicious Referer suppression but that exposes a compatibility penalty as network configurations and some browsers suppress the Referer header for valid requests.

C. **Custom HTTP Header**

Custom HTTP headers can be used to prevent CSRF attack because the browser feature prevents sites from sending custom HTTP headers to another site, but allows another sites to send custom HTTP headers to themselves using AJAX (XMLHttpRequest). For example, the *header.js* JavaScript library uses this approach and attaches the *X-Requested By* header with the XMLHttpRequest value. Google Web Toolkit also recommends that web developers defend against CSRF attacks by attaching a *X-XSRF-Cookie* header to XMLHttpRequest that contains a cookie value. The cookie value is not actually required to protect CSRF attacks: the presence of the header is sufficient.

To use custom headers as a CSRF defence, a site must issue all state-updating requests using XMLHttpRequest, attach the custom header, and reject all state-updating requests that are not attached with the header. For example, to defend against login CSRF, the site must send the user's authentication credentials to the server via XMLHttpRequest.

D. **Origin Header**

To defence against CSRF attacks, browsers send Origin header with POST requests that checks the origin that initiated or started the request. If the browser is unable to determine the origin, the browser sends the null value.

Privacy: The Origin header improves the Referer header by respecting the user's privacy:

- The Origin header includes only the information required to identify the principal that initiated the request (typically the scheme, host, and port of the active document's URL). In particular, the Origin header does not contain the full path or query portions of the URL contained in the Referer header that would occupy privacy without providing additional security.
- The Origin header is sent only for POST requests, whereas the Referer header is sent for all requests. Simply following a hyperlink does not send the Origin header and thus preventing the majority of accidental leakage of sensitive & confidential information.

By responding to privacy concerns, the Origin header will not be suppressed widely.

Server Behavior: To use the Origin header as a CSRF defence, sites should behave as below:

- All state-modifying requests, including login requests, must be sent using the POST method. In particular, state-updating GET requests must be blocked in order to address the threat model.
- If the Origin header contains an undesired value then server should reject the request.. For example, the requests which initiates with another site, that all requests are denied by the original site.

V. CONCLUSION

We have already studied some existing defensive mechanism to mitigate CSRF attack, but these do not provide complete protection against CSRF attacks, or require some modification to the existing web application that should be protected. We may combine two defensive mechanisms in future to defence against CSRF attack.

REFERENCES

[1] OWASP. https://www.owasp.org/index.php/Top_10_2013-Top_10

[2] Lin, Xiaoli, et al. "Threat Modeling for CSRF Attacks." Computational Science and Engineering, 2009. CSE'09. International Conference on. Vol. 3. IEEE, 2009.

[3] http://nvd.nist.gov/ (*CSRF Attacks Categories until 2009 on NVD*)

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

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

[6] Siddiqui, Mohd Shadab, and D. Verma. "Cross site request forgery: A common web application weakness." *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE, 2011.

[7] Jovanovic, Nenad, Engin Kirda, and Christopher Kruegel. "Preventing cross site request forgery attacks." *Securecomm and Workshops, 2006*. IEEE, 2006.

[8] Alexenko, Tatiana, et al. "Cross-site request forgery: attack and defense." *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. IEEE, 2010.

[9] Feil, Renaud, and Louis Nyffenegger. "Evolution of cross site request forgery attacks." *Journal in Computer Virology* 4.1 (2008): 61-71.

[10] Kombade, Rupali D., and B. B. Meshram. "CSRF Vulnerabilities and Defensive Techniques." *International Journal of Computer Network and Information Security (IJCNIS)* 4.1 (2012): 31.