International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Mitigation of CSRF Attack

Nikunj Tandel[1], Kalpesh Patel[2]

[1]Scholar, Computer Department, LD College of Engineering, Gujarat Technological University

[2]Assistant Professor, L.D. College of Engineering.

Abstract: ***Web Application is used in our day to day life and there are several vulnerabilities in web application as per OWASP (Open Web Application Security Project). Most of the developers are unaware about the CSRF attack therefore many web application are still vulnerable from CSRF attack. Cross Site request forgery attack occur when the malicious web site forces a user's browser to send unauthorized request and perform unwanted action on a trusted web site without user's awareness. In this paper we will study about the CSRF attack and existing mechanisms for mitigating CSRF attack and also compare our approach with existing approaches.***

Keywords: OWASP**,** Cross Site Request Forgery (CSRF), web application vulnerability, Mitigation techniques

1. 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 is a branch of Information Security that 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 forces victim's web browser to perform an unwanted action on a trusted web site 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 mitigation techniques. Section V discusses about our proposed scheme section VI discusses results and at the end draws some conclusion.

2. Cross Site request forgery

Cross-Site Request Forgery (CSRF) attacks occur when a malicious web site causes a user's browser to send an authorized request and perform unwanted action on a trusted web site without the user's awareness.
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.

There is also new variant in CSRF attack known as Login CSRF which we discussed later.

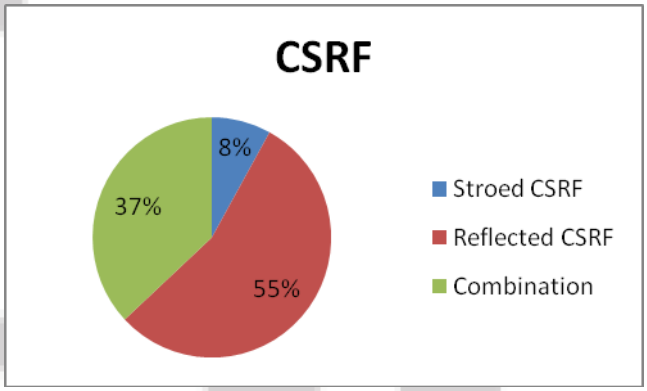

Figure 1: CSRF Attacks Categories until 2013 on NVD [3]

This is the chart for Stored CSRF, Reflected CSRF and Combination attack done on web sites and web application until 2013 on National Vulnerability Database.

3. Working procedure of CSRF

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

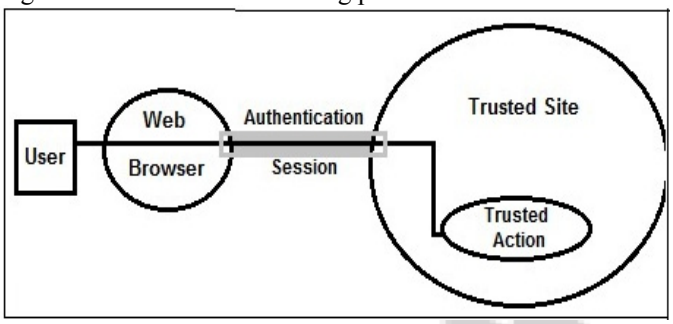

Figure 2: A Valid request [4]

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.

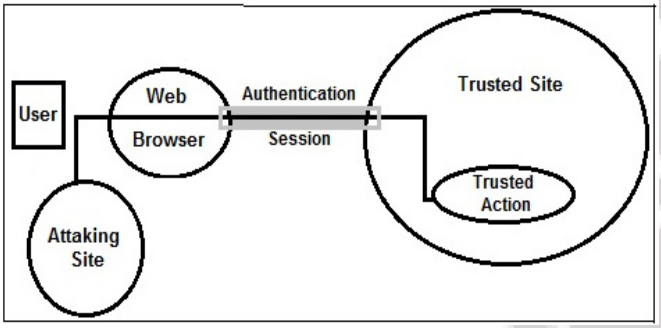

Figure 3: A CSRF attack [4]

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.

3.1 Login CSRF

Login CSRF is a new variation on CSRF attacks, in which an attacker uses the victim's browser to forge a cross-site request to the honest site's login URL, supplying the attacker's username and password. If the forgery succeeds, the honest server responds with a Set-Cookie header that instructs the browser to mutate its state by storing a session cookie, logging the user into the honest site as the attacker. This session cookie is used to bind subsequent requests to the user's session and hence to the attacker's authentication credentials. Many web sites, including Yahoo, PayPal, and Google, were vulnerable to login CSRF.

Search History: Many search engines, including Yahoo! and Google, allow their users to opt-in to saving their search history and provide an interface for a user to review his or her personal search history. Search queries contain sensitive details about the user's interests and activities and could be used by an attacker to embarrass the user, to steal the user's identity, or to spy on the user. An attacker can spy on a user's search history by logging the user into the search engine as the attacker.

4. CSRF Vs. XSS

Cross-Site Request Forgery (CSRF) exploits the trust that a 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 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.

5. Existing mitigation techniques

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

One approach to defend against CSRF attacks is to send an additional information in each HTTP request that can be used to determine whether the request came from an authorize source. This '**validation token**' should be hard to guess for an attacker who does not already have access to the user's account. If a request is missing a validation token or the token does not match the expected value, the server will reject the request.

Random form tokens: To prevent CSRF attacks, a web application has to make sure that the incoming data is originated from a valid HTML form. "Valid" in this context denotes the fact that the submitted HTML form was generated by the actual web application. It also has to be ensured that the HTML form was generated especially for the submitting client. To enforce these requirements, hidden form elements with random and unique values can be employed. These values are used as one time tokens: The triplet consisting of the form's action URL, the ID of the client (e.g. the session ID) and the random form token are stored by the web application. Whenever data is submitted, the web application checks if this data contains a known form token which was stored for the submitting client. If no such token can be found, the form data has been generated by a foreign form and consequently the request will be denied.

Using explicit authentication: There are methods to communicate authentication tokens explicitly: Authentication tokens can be included into the web application's URLs or transported via hidden fields in HTML forms. These techniques are resistant to CSRF attacks.

Drawback: The drawback of this approach is the considerable amount of manual work that it involves. Many current web applications have evolved into large and complex systems, and retrofitting them with the mechanisms necessary for token management would require detailed application-specific knowledge and considerable modifications to the application source code. Even more important, there is no guarantee that the modified code is indeed free of CSRF vulnerabilities, as developers tend to make errors and omissions.

B. The Referer Header

An HTTP request's Referer indicates the URL of the webpage that contained the HTML link or form that was responsible for the request's creation. The Referer is communicated via an HTTP header field. The Referer header, if present, distinguishes a same-site request from a cross-site request because the header contains the URL of the site making the request. A site can defend itself against cross-site request forgery attacks by checking whether the request in question was issued by the site itself. If this is not the case, the request is usually rejected.

Unfortunately, the Referer contains sensitive information that impinges on the privacy of web users [5]. For example, the Referer header reveals the contents of the search query that lead the user to visit a particular site. Although this information is useful to web site owner, who can use the information to optimize their search engine rankings, this information disclosure leads some users to feel their privacy has been violated. Additionally, many organizations are concerned that confidential information about their corporate intranets might leak to external web sites via the Referer header.

Bugs: Historically, browsers and have contained vulnerabilities that let malicious web sites spoof value of the Referer header, especially in conjunction with proxy servers. 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 only to spoof Referers directly back to the attacker's own site.

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

- *in lenient Referer validation*, the site blocks requests whose Referer header has an incorrect value. If a request lacks the header, the site accepts the request. Although widely implemented, lenient Referer validation is easily circumvented because a web attacker can cause the browser to suppress the Referer header. For example, requests issued from ftp and data URLs do not carry Referer headers.
- In *strict Referer validation*, the site also blocks requests that lack a Referer header. Blocking requests that lack a Referer header protects against malicious Referer suppression but incurs a compatibility penalty as some browsers and network configurations suppress the Referer header for legitimate requests.

C. Custom HTTP Header

Custom HTTP headers can be used to prevent CSRF because the browser prevents sites from sending custom HTTP headers to another site but allows sites to send custom HTTP headers to themselves using XMLHttpRequest. For example, the *prototype.js* JavaScript library uses this approach and attaches the *X-Requested By* header with the value XMLHttpRequest. 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 prevent CSRF attacks: the mere presence of the header is sufficient.

To use custom headers as a CSRF defence, a site must issue all state-modifying requests using XMLHttpRequest, attach the custom header (e.g., X-Requested-By), and reject all state-modifying requests that are not accompanied by 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 prevent CSRF attacks, browsers send Origin header with POST requests that identifies the origin that initiated the request. If the browser cannot determine the origin, the browser sends the value null.

Privacy: The Origin header improves on 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 path or query portions of the URL included in the Referer header that invade 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 (e.g., from a list of search results or from a corporate intranet) does not send the Origin header, preventing the majority of accidental leakage of sensitive information.

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

Server Behaviour: To use the Origin header as a CSRF defence, sites should behave as follows:

- All state-modifying requests, including login requests, must be sent using the POST method. In particular, state-modifying GET requests must be blocked in order to address the forum poster threat model.

If the Origin header is present, the server must reject any requests whose Origin header contains an undesired value (including null). For example, a site could reject all requests whose Origin indicated the request was initiated from another site.

6. Proposed Approach

The existing mitigation techniques like secret token validation, the HTTP referer header validation, custom HTTP header validation and origin header are available to prevent

CSRF attacks and none of these mechanisms are providing complete protection from CSRF attack. The OWASP designed CSRF Guard a server side mechanism to prevent against CSRF attack this approach is good but it not gives the protection against Login CSRF attack. So I have design the mechanism that mitigates Stored CSRF, Reflected CSRF as well as Login CSRF.

An activity diagram of our proposed approach.

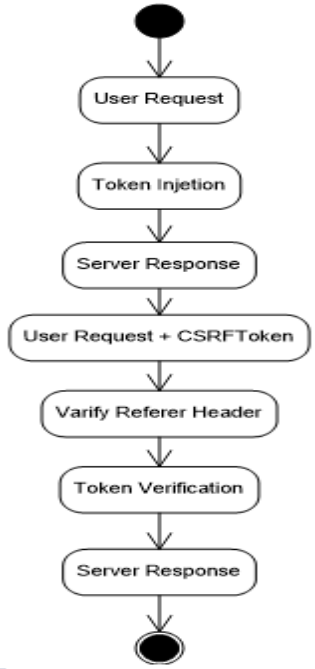

Figure 4: Comparisons with other approach

We shown erlier that most of the approaches uses Secret Token Validation for mitigating CSRF attack but those approach are not worked properly, so we develop our approach that mitigate complete CSRF attack.

For that I have make a method for generating a Token, wich generate a token with UUID (Universally Unique Identifier) random number that appende into the Request, then we the user send request for another time it checks for referer header and also checks for Secret Token.

And if the referer header is not there in request or cross site request than our approach not permit user for futher pross, same in token validation token if user's token is not there our not valid than the approach not permit for further prosess.

7. Experimental Result

In this section existing token based solution such as NoForge and CSRFGuard are compare with our approach (secretToken + Referer Header) to determine accuracy and robust security against Stored CSRF, Reflected CSRF and Login CSRF.

Approaches	Stored CSRF	Reflected CSRF	Login CSRF	Accurate	Robust security
NoForge [10]	✓	✓	X	Less	Less
CSRFGaurd [9]	✓	✓	X	Average	Average
Our Approach	✓	✓	✓	More	More

8. Conclusion and Future work

We have already studied some existing mitigation techniques to mitigate CSRF attack, Secret Validation Token is definitely the best approach for mitigating Stored CSRF and Reflected CSRF but these do not provide complete protection against Login CSRF attack, or require some modification to the existing web application that should be protected. We propose our approach that combines both Secret Validation Token and Referer Header Validation for mitigating all of three Stored CSRF, Reflected CSRF and Login CSRF. Thus, we conclude that our approach is more accurate at mitigation and provide robust security 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] Eric Sheridan. OWASP CSRFGuard Project, 2008. http://www.owasp.org/index.php/CSRF_Guard. Mar 2009.
[10] NenadJovanovic, EnginKirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), 2006.
[11] Feil, Renaud, and Louis Nyffenegger. "Evolution of cross site request forgery attacks." *Journal in Computer Virology* 4.1 (2008): 61-71.
[12] 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.

Books

[1] William, Stallings, and William Stallings. *Cryptography and Network Security, 4/E*. Pearson Education India, 2006

Author Profile

Nikunj Tandel received the B.E. degree in Information Technology from A.D. Patel Institute of Technology in 2012. Presently pursuing his M.E. from L.D. Collage of Engineering since 2012. Currently he is doing his dissertation in Web Security and main area is Network Security.

Prof. Kalpesh Patel received the B.E degree in Information Security and M.E. degrees in Computer Engineering. Presently he is working as Assistant Professor at L.D. College of Engineering.