Vulnerabilities and Defensive Mechanism of CSRF

Purnima Khurana[#1], Purnima Bindal[#2]

[#] *Assistant Professor ,Department of Computer Science*
P.G.D.A.V. College, University of Delhi, Delhi, India

***Abstract*— In today's era the internet and its applications provide an easy way to individuals which helps them in their day to day life. As the use of technology increases, dependency on web applications also increases. But these web applications have some major threats and one of them is CSRF(Cross Site Request Forgery). CSRF is a common web application weakness. Cross Site Request forgery attack occur when a malicious web site causes a user's web browser to perform an unwanted action on a trusted site. There are various possible vulnerabilities and defensive mechanism of CSRF. CSRF flaws exist in web applications with a predictable action structure and which use cookies, browser authentication or client side certificates to authenticate users. This study will help to create awareness about the CSRF attack.**

***Keywords*— Web Application, Vulnerability, Attacks, Defensive measures, Cross-Site Request forgery Introduction**

I. INTRODUCTION

Use of internet is tremendously increasing with technology; it is now used for each possible function that can be performed online. Web applications are playing important role to provide these functions. Web applications have become part of life of human beings but with all these facilities they have also bring some problems i.e. web application attacks. Web application attacks create insecure environment for web application's users [1]. One of the web application attack is CSRF (Cross-Site Request Forgery). Cross-Site Request Forgery attacks are also known as Cross-Site Reference Forgery/ XSRF/ Session Riding/Confused Deputy attacks.

In a CSRF attack, a malicious site instructs a victim's browser to send a request to an honest site, as if the request were part of the victim's interaction with the honest site, leveraging the victim's network connectivity and the browser's state, such as cookies, to disrupt the integrity of the victim's session with the honest site [2].

II. KEY CONCEPTS OF CROSS-SITE FORGERY

- Malicious requests are sent from a site that a user visits to another site that the attacker believes the victim is validated against.
- The malicious requests are routed to the target site via the victim's browser, which is authenticated against the target site.
- The vulnerability lies in the affected web application, not the victim's browser or the site hosting the CSRF.

III. CSRF IN COMPARISON TO XSS

XSS and CSRF are two types of computer security vulnerabilities. XSS stands for Cross-Site Scripting. CSRF stands for Cross-Site Request Forgery. In XSS i.e. Cross Site Scripting a relatively older attack talks about injecting malicious scripts in web pages which then would served to other users over a period of time. The malicious scripts in turn gains access to page content and start misusing it[7]. In other words , the hacker takes advantage of the trust that a user has for a certain website. On the other hand, in CSRF the hacker takes advantage of a website's trust for a certain user's browser.

Comparison between XSS and CSRF[5]:

TABLE I
COMPARISON BETWEEN CSRF AND XSS

S. N0.	Full Form	**XSS**	**CSRF**
1	Definition	Cross-Site Scripting	Cross-Site Request Forgery
2	Dependency	In XSS, a hacker injects a malicious client side script in a website. This script is added to cause some form of vulnerability to a victim.	It takes advantage of the targeted website's trust in a user. A malicious attack is designed in such a way that a user sends malicious requests to the target website without having knowledge of the attack.
3	Requirement of JavaScript	Injection of arbitrary data by data that is not validated	On the functionality and features of the browser to retrieve and execute the

			attack bundle
4	Condition	Yes	No
5	Vulnerability	Acceptance of the malicious code by the sites	Malicious code is located on third party sites

A. How CSRF Attack works:

1. Victim browser visits a target website T and sends login details username and password to T.
2. Targeted website T returns the login response with session cookie.
3. Suppose victim visits a malicious site M and M supplied malicious content to victim browser which contains JavaScript code or an image tag that causes victim browser to send an HTTP request to T. Because the request is going to T, victim browser appends the session cookie to the request.
4. On seeing the request, T infers from the cookie's presence that the request came from victim, so T performs the requested operation on victim account, but the victim is unaware until later or perhaps never. So, this is a successful CSRF attack.

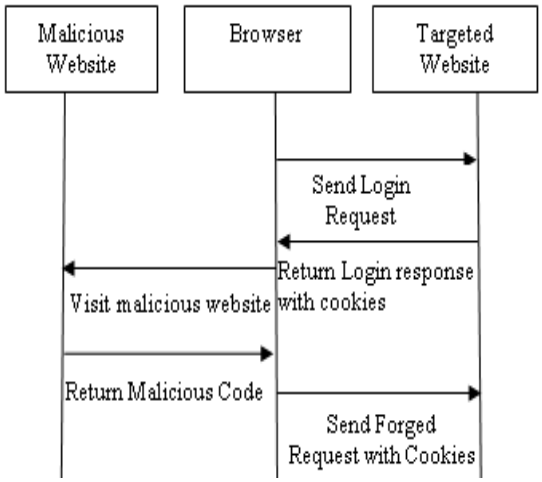

Fig. 1 This figure represents how CSRF attack works

B. CSRF Vulnerabilities

Many flaws are there which helps attackers and make their job easy to satisfy their requirement. In this section we will take review of such vulnerabilities presents in web applications [1].

1) HTTP session handling mechanism

Number of website required user authentication while accessing it, which is most important requirement to carry out user specific tasks as well as to provide privacy to user's data and information. To simplify this requirement HTTP protocol provides facility of session and cookie, which allow web server to differentiate the request coming from different users. Once user gets authenticated, this session cookie information gets passed in every request from server to client and vice versa.

2) HTML Tags

CSRF attackers embed the request they want to execute in HTML tags due to which attack become invisible and while loading particular page (with page, it loads the all elements present on page), request gets executed. Also sometime it is embedded into the tags where it will get execute only if user click on that tag's user interface like 'href tag'. In this case attacker forces the user to click on such tags by showing text which attracts user e.g. " upto 40% discount on footwear " etc. There are so many tags present in HTML which can send request to server, but each and every tag is made for particular type of request like for image file, JavaScript file etc.. HTML does not check the tag source property contains the valid URL or not, and CSRF attackers take advantage of this vulnerability.

3) *GET and POST method of form submission*

Information in the form fields sends to the server by using two methods GET and POST, where GET method generate a request which contain all the information itself in request and it is also visible to the user, so attacker can make use of this easily available information to generate valid request. It was suggested that to use POST instead of GET method to stop this vulnerability**.** But POST method is also not helped to protect web applications from CSRF attack. Once attacker get all form fields he can embed these fields into his web page, which he is going to force the victim to open and can put the JavaScript function which allow form to submit on onload event.

4) Browser's view Source option

There are various different ways by which attacker get knowledge of functionality used by web application, which helps attacker to generate valid request. Attacker can himself log on the website and check the whole functionality, also information about working of forms on the web pages can be easily available by facility provided by web browser using option 'View Source', Which shows all the information of the fields present on forms, validation for each field can be accessed by using JavaScript files and much more information attacker can collect. If web application using extra session variable on each request to protect application from CSRF and if that session information is saved in hidden field, using view source option attacker can get the logic used to generate this session field unless until it is not strongly generated random token.

C. Some Specific Examples of Vulnerabilities on Websites

We found four major vulnerabilities on four different sites. We believe ING Direct, Metafilter and YouTube, and the New York Times have corrected the vulnerabilities[4].

All four sites appear to have fixed the problem.

1) *ING Direct* ***(ingdirect.com)***

ING's website has a vulnerability that allowed additional accounts to be created on behalf of an arbitrary user.Also funds can be transferred out of users' bank accounts.

2) *YouTube* ***(youtube.com)***

CSRF vulnerability was also in nearly every action a user could perform on YouTube. An attacker could have added videos to a user's "Favorites," added himself to a user's "Friend" or "Family" list, sent arbitrary messages on the user's behalf, flagged videos as inappropriate, automatically shared a video with a user's contacts, subscribed a user to a "channel" (a set of videos published by one person or group) and added videos to a user's "QuickList" (a list of videos a user intends to watch at a later point).

3) *MetaFilter* ***(metafilter.com)***

A vulnerability existed on Metafilter that allowed an attacker to take control of a user's account. A forged request could be used to set a user's email address to the attacker's address. A second forged request could then be used to activate the "Forgot Password" action, which would send the user's password to the attacker's email address.

4) *The New York Times* ***(nytimes.com)***

A vulnerability in the New York Time's website allows an attacker to find out the email address of an arbitrary user. This takes advantage of the NYTimes's "Email This" feature, which allows a user to send an email about a story to an arbitrary user. This emails contains the logged-in user's email address. An attacker can forge a request to active the "Email This" feature while setting his email address as the recipient. When a user visit's the attacker's page, an email will be sent to the attacker's email address containing the user's email address. This attack can be used for identification (e.g., finding the email addresses of all users who visit an attacker's site) or for spam. This attack is particularly dangerous because of the large number of users who have NYTimes' accounts and because the NYTimes keeps users logged in for over a year.

D. CSRF DEFENSES

As CSRF become popular various defensive measures against it were suggested, but none of these is able to defense against CSRF completely. But these help to minimize the risk of CSRF up to certain extent [6].

1) Token

The classic solution to CSRF has been a per session token known as the synchronizer token design pattern.

The basic flow using this solution:
Step 1: When the user logs in, a randomized string (token) is then placed in the user session
Step 2: On every form for a non-idempotent request (essentially meaning any request that changes the server-side state – which should be your HTTP POSTs), the token is placed in the form when it is submitted
Step 3: The request handler for the non-idempotent request validates that the submitted token matches the token stored in the session. If either of the tokens is missing, or if they do not match, do not process the transaction.

In the past, this per-session token solution has served pretty well for most CSRF situations, but it can be time-consuming to implement and it also creates opportunities for forgetting validation on some requests.

Another solution that uses the token pattern is the ESAPI project, which has built-in CSRF protection, However, ESAPI's CSRF solution is tied to the authentication scheme.

2) ***CSRFGuard***

A very good option offering solid protection against CSRF is the OWASP CSRFGuard project. This library makes it relatively easy to build CSRF protection into your application by simply mapping a filter and updating a configuration file. This is certainly a resource worth checking out.

3) ***Stateless CSRF Protection***

When you cannot – or do not want to – maintain the user token in the server-side session state, this is a seemingly good solution. The idea is to allow the client side to create a cookie with the CSRF token (which is then submitted on every request), and to then include the token as a request parameter. Because an attacker can not read both the cookie and the request parameter, then all the server side should have to do is validate that the token in the cookie and the request parameter match on another. This solution, to my knowledge, has yet to be widely reviewed or tested, but it is a great example of an elegant

solution to a difficult problem. Only time will tell if this is the go-forward solution for stateless CSRF protection.

IV. CONCLUSIONS

In this survey paper we have discussed CSRF vulnerabilities which help to understand different attack scenarios and various examples of vulnerabilities detected on different websites. We have described how CSRF is different from XSS attack. And also the defense mechanism to protect web application against CSRF attacks. Complete protection against CSRF is not available and our discussed defensive techniques need more improvement so that they can completely protect the application. Web developers need to understand these vulnerabilities so that they can protect web applications from all the side effects..

REFERENCES

[1] Rupali D. Kombade, Dr. B.B. Meshram," CSRF Vulnerabilities and defence technique", I. J. Computer Network and Information Security, February 2012.

[2] Adam Barth, Collin Jackson, John C. Mitchell, " Robust Defenses for Cross-Site Request Forgery" , Oct.2008.

[3] William Zeller and Edward W. Felten, " Cross-Site Request Forgeries: Exploitation and Prevention," The New York Times, 2008.

[4] Bill Zeller (2008) Popular Websites Vulnerable to Croos-Site Request Forgery Attacks webpage on Freedom to Tinker. [Online]. Available: https://freedom-to-tinker.com/blog/wzeller/popular-websites-vulnerable-cross-site-request-forgery-attacks/

[5] Difference Between XSS and CSRF webpage on DifferenceBetween.info. [Online]. Available: http://www.differencebetween.info/difference-between-xss-and-csrf

[6] Niraj Bhatt (2010) Cross Site Scripting (XSS) vs. Cross Site Request Forgery (CSRF) webpage on Wordpress.com. [Online]. Available: http://nirajrules.wordpress.com/2010/01/16/cross-site-scripting-xss-vs-cross-site-request-forgery/

[7] John Melton (2012) CSRF prevention in java webpage on WhiteHat Security. [Online]. Available: https://blog.whitehatsec.com/tag/synchronizer-token /