# Survey on Cross Site Request Forgery (An Overview of CSRF)

**Conference Paper** · March 2013

CITATIONS
2

READS
1,491

**1 author:**

Sentamilselvan K
Kongu Engineering College
**10** PUBLICATIONS **10** CITATIONS

SEE PROFILE

IEEE - International Conference on Research and Development Prospects on Engineering and Technology (ICRDPET 2013)
March 29, 30 - 2013 Vol.5

# Survey on Cross Site Request Forgery

## *(An Overview of CSRF)*

*Sentamilselvan K*
*Department of computer science*
*Pondicherry Engineering College*
*Pondicherry*
*ksentamilselvan@pec.edu*

*Dr.S.Lakshamana Pandian*
*Assistant Professor*
*Pondicherry Engineering College*
*Pondicherry*
*l.pandian72@pec.edu*

*Dr.K.Sathiyamurthy*
*Assistant Professor*
*Pondicherry Engineering College*
*Pondicherry*
*sathiyamurthyk@pec.edu*

***Abstract*— Cross Site Request Forgery is considered as one of top vulnerability in today's web, where an untrusted website can force the user browser to send the unauthorized valid request to the trusted site. Legitimate user will lose his/her integrity over the website when the Cross site request forgery takes place. So far many solutions have been proposed for the CSRF attacks such as the referrer HTTP Header, Custom HTTP header, Origin Header, client site proxy, Browser plug-in and Random Token Validation. But existing solutions is not so immune as to avoid this attack. All the solutions are partially protected only. This paper focuses on describing the implementation of various possible cross site request forgery methods and describing the pitfalls in the various preventive techniques of cross site request forgery and so we suggested some defense mechanism to prevent this vulnerability.**

***Keywords*— *security threats, security breaches, browser security, forgery prevention, defense mechanisms, open web application security.***

## I. INTRODUCTION

Now-a-days, web users are increasing in manifolds, at the same time attackers also increase in proportionately. So the necessity of security in accessing web is a must for secure organizations, defense personals and financial bank those interact with public. In 2010, Open Web Application Security Project [10] reported the following most critical web application security vulnerabilities that are been exploited

| S.no | Web application Attacks |
|---|---|
| 1 | Injection |
| 2 | Cross-Site Scripting (XSS) |
| 3 | Broken Authentication and Session Management |
| 4 | Insecure Direct Object References |
| 5 | Cross-Site Request Forgery (CSRF) |
| 6 | Security Misconfiguration(NEW) |
| 7 | Insecure Cryptographic Storage |
| 8 | Failure to Restrict URL Access |
| 9 | Insufficient Transport Layer Protection |
| 10 | Unvalidated Redirects and Forwards (NEW) |

Here, Cross site request forgery attack (CSRF) is $5^{th}$ position in this list. This attack is severe vulnerability in web applications. According to rsnake [13], founder of ha.ckers.org, there are too many CSRF vulnerabilities on the Internet. The CSRF attacks are typically as powerful as a user, i.e. any action that the user can perform can also be performed by an attacker using a CSRF attack. Consequently, the more power a site gives a user, the more serious are the possible CSRF attacks. For example, if the victim account has administrator rights, this can compromise the entire web application.

Customers are provided with safe web services and too they are protected from many web threats. The web has become an indispensable part of our life. Unfortunately, as our dependency on the web increases, so does the interest of attackers in exploiting web applications and web-based information. There are more number of attacks which exploits the web application and integrity of the web users. By using the web browser, one can access webmail's, online banking, community websites, search engines, and specific business applications for each sector, etc. from the private network or from the Internet. They may contain sensitive information and it required an authentication.

The general class of cross site request forgery (XSRF) attacks was first introduced by Peter W in a posting to the BugTraq mailing list, and it has since been picked up by web application developers [11]. An untrusted website can force the user browser to send the unauthorized valid request to the trusted site is called as the Cross site request forgery. This is can be done by without the knowledge of users. It is a new class of attack on web applications which exploit the trust of authenticated users. By launching a successful CSRF attack against the authenticated user, an attacker is able to initiate arbitrary HTTP requests using the user credentials to the vulnerable web application. A successful CSRF attack effectively bypasses the underlying authentication mechanism and it can compromise the end user data and operation. If the victim account has administrator rights, this attack can compromise the entire web application.

Image based CSRF attacks are easy to exploit on the web applications. HTML image elements and JavaScript image objects are the two most popular paths to CSRF. There is no solution available to prevent CSRF attacks using IMG elements. The IMG elements with static URLs (Uniform

ISBN: 978-1-4673-4948-2 © 2013 IEEE

Resource Locator) have predefined extensions (i.e. .jpg, .bmp,.png, .gif, etc.) in their SRC attribute value (i.e. Uniform Resource Identifier or Uniform resource locator), to retrieve the image. By validating the image extensions [12] in the web page source code, we can prevent the CSRF attacks. Many web applications forget that HTTP requests they receive from browsers may have been forged by another web page opened in the same browser. Without the user being aware of it, this malicious web page can take over his identity and send a request to other website on his behalf. This is also one kind of attack in Cross-Site Request Forgery (CSRF).which is shown in the following figure.

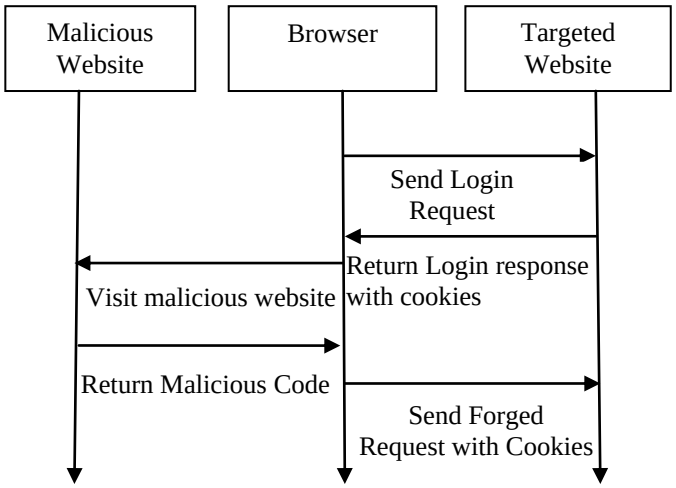


Figure 1. Processing of CSRF

CSRF exploits the HTTP protocol's feature, that a web page can include HTML elements that will cause the browser to make requests to other websites. Like all HTTP transactions, the requests to the other sites will include the user's session information such as cookies or HTTP integrated authentication if they have an established session. Regardless of if the user has a session with the other sites, elements of those sites will be loaded in the victim's browser and can appear in the browser's cache and history. A CSRF can occur on an HTTP request using either the GET or the POST method. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data.

A CSRF attack can be carried out in different ways. The attack could be done using a HTML IMG Tag or a specially crafted URL embedded into the Target application. This works for sure since the victim will be logged into the application. Another way of doing it is to host a site/blog and influence the victim to visit the site. Many sites can be used to host a CSRF including online forums (which often allow a user to link to an image as an attachment), HTML email, photo galleries, wikis, blogs, online auctions, and E-Commerce sites. CSRF attack using these sites might not work always as users may not be currently logged into the target system when the exploit is tried. CSRF flaws exist in the web applications with a predictable action structure and which use the above credentials to authenticate users. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish a malicious request from a legitimate user request.

## II. CSRF CLASSIFICATIONS

As we discussed earlier CSRF is one of the powerful vulnerabilities where an attacker performs the unauthorized activities without the knowledge of the user. This Vulnerability can be classified as

### *A. Stored CSRF*

A stored CSRF vulnerability is one where the attacker can use the application itself to provide the victim the exploit link [12] or other content which directs the victim's browser back into the application, and causes attacker controlled actions to be executed as the victim. Any application that uses HTML is vulnerable to CSRF attack where the attacker stores the malicious code using the IMG, IFRAME tag or XSS. Examples of that tag are shown below. The asperity of the site will be high, if the attacker can able to store the CSRF attack on that site. The likelihood is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. Stored CSRF vulnerabilities are more likely to succeed, since the user who receives the exploit content is almost certainly authenticated to perform actions. Stored CSRF vulnerabilities also have a more obvious trail, which may lead back to the attacker.

- <img> tag
  <imgsrc="http://127.0.0.1/klog.js">
- <iframe> tag
  <iframesrc="http://127.0.0.1/" width=100% height=100%>
  </iframe>
- <form> tag
  <form action=http://127.0.0.1/klog.js method="get">
  </form>
- <link> tag
  <a href="http://127.0.0.1/klog.js">click here</a?
- <script> tag
  <script src="http://127.0.0.1/klog.js"></script>

### *B. Reflected CSRF*

In this attack, the attacker performs the attack outside the system application, by exposing the victim to exploit link [12] or content which is malicious. This can be done using a blog, an email message, an instant message, a message board posting, or even an advertisement posted in a public place with an URL that a victim types in. Reflected CSRF attacks will frequently fail, as users may not be currently logged into the target system when the exploits are tried. The trail from a reflected CSRF attack may be under the control of the attacker, however, and could be deleted once the exploit was completed. Here the page can be submitted automatically, example of that tag is shown below,


ISBN: 978-1-4673-4948-2 © 2013 IEEE

- *Auto posting Forms*

```
<body onload="document.forms[0].submit()">
<form method="POST" action="https://abc.com/fn">
<input type="hidden" name="sp" value="8109"/>
</form>
```

### C. Login CSRF

By using adversary's identities this attack can be implemented. Making a forgery request with the identity of the victim is not gets succeeded here. Personal data's of the victim are getting collected in this attack. The main objective of this type of attack is the attacker and the users are both authorized users. But the attacker will send his identity to the user by means of some link. Thus user will now enter using the attacker's identity. Then all the information related to user will get stored in the user browser, but actually in the attacker identity. So, the attacker will now see this information of the user. Therefore, they input personal data to the web server such as credit card numbers or search keywords. Moreover, by using a Gadget, adversaries can execute malicious gadgets registered to the adversaries' accounts within the victim's local machine. Both CSRF and Login CSRF are subtle exploitations based on the common belief that every web request made by a browser is initiated under users' supervision. However, in the real world, there are many ways for malicious adversaries to initiate page requests from the victim's browser [1].

Cross-site request forgery focuses on requests that mutate server-side state, either by leveraging browser's network connectivity or by leveraging the browser's state. CSRF attacks that mutate browser state have received less attention even though these attacks also disrupt the integrity of the user's session with honest sites. In a login CSRF attack [1], the attacker forges a login request to an honest site using the attacker's user name and password at that site. 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. Login CSRF attacks can have serious consequences, depending on other site behavior. This is an example script for login CSRF.

```
<form action=https://www.google.com/login method=POST
target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyz>
</form>
<script>document.forms[0].submit()
</submit>
```

*Search History*-Many search engines, including Yahoo! and Google [1], 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. The user's search queries are then stored in the attacker's search history, and the attacker can retrieve the queries by logging into his or her own account.

## III. Real World Examples For CSRF

### A. ING Direct (ingdirect.com)

It is one of the financial institutions where CSRF attack was first took place [16]. Attacks implemented by transferring the funds out of the user's bank account by the unauthorized people. This is due to the vulnerability on lNG's website. It makes to add the additional accounts on behalf of an arbitrary user.

### B. YouTube (youtube.com)

YouTube is one of the most viewable sites [16]. Vulnerabilities on the YouTube make the unauthorized people to add the account on behalf of the legitimate user. Using that account attacker 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, tagged 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 "Quick List" (a list of videos a user intends to watch at a later point).

### C. MetaFilter (metafilter.com)

Metafilter is one of the site where vulnerability present here makes the user's account taken over by the attacker [16]. 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 Equations.

### D. The New York Times (nytimes.com)

Vulnerability present in the New York Time's website allows an attacker to find out the email address of an arbitrary user [16]. 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 email 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.

ISBN: 978-1-4673-4948-2 © 2013 IEEE

Also, Times People, a social networking site launched by the New York Times on September 23, 2008, is also vulnerable to CSRF attacks.

### *E. Gmail (www.gmail.com)*

In January 2007 this serious vulnerability was discovered in GMail which allowed an attacker to steal a GMail user's contact list [7].

### *F. Netfix*

It was discovered in Nettlix which allowed an attacker to change the name and address on the account, as well as add movies to the rental queue etc [7].

### *G. EBay's site*

EBay is one of the auction sites, where more and more information get stored. CSRF attack was implemented which leads to loss of many personal information of about 18 million people. This issue was discovered in February 2008 [14].

## IV. DEFENSE MECHANISMS

It is a policy that needs to be adopted by developers and users to avoid this attack to some extent.

*a) Use random tokens:* To defend against the CSRF attack it is one of the greatest mechanisms. Form proposal was done during the use of random tokens at all time. To fill in malicious URL, prediction of next random pattern was difficult for the aggressor [7].

*b) Use of post in form rather than get:* Form submission involves two methods they get and post. Form submission is the secure one for post method. Variables and values in URL are understood by anyone in get method as a query strings [7].

*c) Limiting the lifetime of authentication cookies:* Beside CSRF it is a durable deterrence. In a short period of time lifetime was limited. After a short period of time cookies will be terminated when the user moves to further web site. For any action user involves in re-login. Re-submission of password by the user was not done if the attacker tries to send any HTTP request [7].

*d) Damage limitation:* To reduce the damage from CSRF those steps are followed by the Destruction restriction. To perform CSRF by an attacker on a website an authentication was required for every usage to limit the damage [7].

*e) Force user to use your form:* Force the user always to use the form of website. For this purpose a hidden fields are used which a helpful one. It is one of the protection and easy to bypass [7].

*f) Auto log off: If a user moves to some other site (untrusted) means it will automatically log off. So again the user wants to login.*

*g) Don't start new task while sensitive task running:* If the user is using sensitive task means don't start new application or task (untrusted).

*h) Never store the password to anywhere:* The user should never store the password in anywhere or anyplace.

*i) Single login access at a time:* A user should use single login access. If more than one access is present means it will intimate to the user.

*j) Divert to some other page (honeypot):* If the user or system find any malicious activities means it will automatically divert to some other page.

*Some Of the Tool are*

i. OWASP CSRF Tester [9]
ii. Ratproxy [4]

## V. RECENT WORKS ON CSRF

Ramarao R, Radhesh M, Alwyn R Pais [12] was presented a client-side proxy solution that detects and prevents CSRF attacks using IMG element or other HTML elements which are used to access the graphic images for the webpage. This proxy is able to inspect and modify client requests as well as the application's replies (output) automatically and transparently extend applications with the secret token validation technique.

William Zeller and Edward W. Felten [16] implemented a client side browser plug-in that can protect users from certain types of CSRF attacks. They implemented their tool as an extension to the Firefox web browser. Users will need to download and install this extension for it to be effective against CSRF attacks. Their extension works by intercepting every HTTP request and deciding whether it should be allowed. This decision is made using the following rules. First, any request that is not a POST request is allowed. Second, if the requesting site and target site fall under the same-origin policy, the request is allowed. Third, if the requesting site is allowed to make a request to the target site using Adobe's cross-domain policy, the request is allowed. If their extension rejects a request, the extension alerts the user that the request has been blocked using a familiar interface (the same one used by Firefox's popup blocker) and gives the user the option of adding the site to a white list.

Sooel Son [14] was proposed PCRF is a dynamic token generating defense scheme against CSRF. PCRF's basic goal is to prevent CSRF attacks by adding a fresh token to every web request whose target page should be protected one way to efficiently prevent CSRF attacks toward PHP web applications. This defense system is called PCRF: Prevent Cross-site Request Forgery attack. PCRF provides an automatic robust solution again CSRF threats by using a CSRF token. Due to the property of cryptographically secure hash function, it used to verify whether the token has been previously issued from servers.

Nanad jovanovic et.al [8] proposed a mitigation mechanism for CSRF that provides only partial protection by replacing GET requests by POST requests or relying on the information in the Referer header of HTTP requests. Also proposed a solution that provides a complete automatic protection from XSRF attacks. More precisely, his approach is


ISBN: 978-1-4673-4948-2 © 2013 IEEE

based on a server-side proxy that detects and prevents CSRF attacks in a way that is transparent to users as well as to the web application itself. (Orthogonal proxy).

Johns and Winter [6] introduced RequestRodeo,a client side solution to counter this threat. With the exception of client side SSL, RequestRodeo implements protection against the exploitation of implicit authentication mechanisms. This protection is achieved by removing authentication information from suspicious requests. They proposed a client side solution to enable security conscious users to protect themselves against CSRF attacks. Their solution works as a local proxy on the user's computer.

TABLE I. ADVANTAGE AND DISADVANTAGE FOR EXIXTING SOLUTIONS

| ***Existing Solution*** | ***Advantages*** | ***Disadvantages*** |
|---|---|---|
| Browser Plug-in | It's Simple | It may sometime Crushed. May be Some user aware of this plug-in. |
| Clint Side Proxy | It is easy way to monitor and find attack. | If proxy Compromised means all sensitive information will lost. It won't detect the login CSRF |
| Secret Token Validation | Computational is low | Requires Dynamic generation |
| Random Validation Token | It is one of the best solutions. Simple to implement. | It needs SSL to be implemented in all applications. It won't Detect Login CSRF |
| Cryptographic token | Very Strong Protection and it requires no additional memory | Requires Dynamic Generation and requires a small amount of system resources to check tokens and big database tables to manage tokens and sessions. |
| Referrer Header | Simple to implement | Many browsers disable this Header |
| Origin Header | It one of the way to find the same site request and cross site request. | Many browsers disable this Header |
| Custom Header | It also one of the way to find the same site request and cross site request. | Many browsers disable this Header |
| Captcha | Easy to Store in the memory. It is better Solution for auto submitting forms. | Requires more memory. It is expensive |
| Code Verifier | It will detect the login CSRF | It will take long time to verify the code |

Tatiana Alexenko et.al [15] were developed a Mozilla extension that integrates with the Firefox web-browser to protect the user's browsing history. The extension generates HTTP requests to random URLs from the user's browsing history. The extension allows the user to specify how often the requests get sent as well as giving users the option of adding a random URL to the Referrer field of the extension-generated HTTP request. The latter option is bound to initiate discussion, because the pairing of the requested URL and Referrer is random which can lead to combinations that should not exist during normal browsing. This can affect online advertising and raise red flags for web administrators.

They implemented a client-side defense measure that previews the HTML code before each page load and detects potential CSRF attacks. The detector would first find all form tags and check the "action" attribute of the "form" tags for deep linking. If such forms are found, the CSRF detector will prompt the user if they want to add the pairing of the URL of the website the code is located on and the URL of the form action to a white list.

Burns and Schreiber [5] provide comprehensive introductions to CSRF attacks. To prevent the CSRF attack, they used following methods. Use cryptographic tokens to prove the Action Formulator knows a session specific secret, use secret tokens to prove the Action Formulator knew an Action and user specific secret, use the optional HTTP referrer [sic] header to verify Action Formulators, require changes to application state to be done only with HTTP POST operations and use a simplified CSRF Prevention Token.

Drawback of their proposed work is that the attackers can adjust their attacks to be form based like CSRF, Submit forms automatically or though tricking users by making huge, mislabeled submit buttons. The header is optional and may not be present, some browsers disable this header and it is not available when interactions occur between HTTPS and HTTP served pages. The risk of header spoofing exists, and tracking the valid sources of invocations may be difficult in some applications.

## VI. FUTURE WORK AND CONCLUSION

Cross Site Request Forgery is one of the top vulnerabilities in the internet. It remains challenging for the researchers to provide a better solution for mitigating this attack. There were many organizations which affected by this cross site request forgery attack. Defense mechanisms and existing solutions for cross site request forgery are working in some extend only. The above work can be extended to provide suitable solutions for the cross site request forgery attack by means of applying parsing techniques to identify the attacking spots before the attackers attack. Some pattern for img, script, form, iframe tags can be designed to identify the attack.

## REFERENCES

[1] A.Barth, C.Jackson, and J.C.Mitchell. "Robust defenses for cross site request forgery". In Proc. ACM Conference on Computer and Communications Security (CCS), Oct, 2008.

[2] Cross-Site Request Forgery. www.owasp.org/index.php/CrossSite_Request_Forgery, May, 2009.

[3] Hossain Shahriar and Mohammad Zulkernine." Client-Side Detection of Cross-Site Request Forgery Attacks",IEEE International Symposium on Software Reliability Engineering, 2010.

[4] http://code.google.com/p/ratproxy/downloads/detail?name=ratproxy-1.58.tar.gz&can=2&q=

[5] J. Burns. Cross Site Reference Forgery: An introduction to a common web application weakness. http://www.isecpartners.com/documents/XSRF_Paper.pdf, 2005.

[6] M. Johns and J. Winter, "RequestRodeo: Client Side Protection against Session Riding," In Proc. of the OWASP Europe Conference, Leuven, Belgium, May 2006.

[7] Mohd. Shadab Siddiqui and Deepanker Verma,"Cross Site Request Forgery: A common web application weakness", IEEE Conference and white paper, 2011.

[8] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. “Preventing cross site request forgery attacks”.In IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), 2006.

[9] OWASP CSRFTester Project. http://www.owasp.org/ index.php/Category OWASP_CSRFTester_Project, May, 2009.

[10] OWASP. Open web application security project (OWASP) top ten project, http://www.owasp.org/index.php/Category:OWASP_ Top_Ten_Project; 2010 Cross-Site Request Forgery. http://www.owasp.org/index.php/Cross-Site_Request_Forgery.

[11] P.W.Cross Site Request Forgeries.www.securityfocus.com/archive/1/191390,2001.

[12] Ramarao R. Tool “preventing image based CSRF attacks”. http://isea.nitk.ac.in/rod/csrf/PreventImageCSRF/. May, 2009.

[13] RSnake. ¨XSS Irony¨. ha.ckers.org. 15/06/2007. http://ha.ckers.org/blog/20070615/xss-irony/

[14] Sooel Son, “Prevent Cross site Request Forgery: PCRF”userweb.cs.utexas.edu/~samuel/PCRF/Final_PCRF_paper.pdf.

[15] Tatiana Alexenko Mark Jenne suman Deb Roy and Wenjun Zeng,” Cross-Site Request Forgery:Attack and Defense”.In Proc.IEEE Communications Society (CCNC), 2010.

[16] W. Zeller and E. W. Felten, “Cross-Site Request Forgeries: Exploitation and Prevention,” Technical Report, Princeton University, 2008.


ISBN: 978-1-4673-4948-2 © 2013 IEEE