A Report on CSRF Security Challenges & Prevention Techniques

Pratibha Yadav[1]
M. Tech (Cyber security)
Department of Information Technology
Raksha Shakti University
Ahmedabad, India
pratibhayadav1120@gmail.com

Mr. Chandresh D. Parekh[2]
Assistant Professor
Department of Telecommunication
Raksha Shakti University
Ahmedabad, India
cdp_tc@rsu.ac.in

Abstract— **Now a days, The world wide web is becoming the town square for the global village of today and we, the netizens of globe are becoming essential part of cyberspace. The use of Internet helps in making the journey of our life easy and comfortable but as we all know that every "pros" has some "cons" attached with it, therefore some precautions need to be taken to make the internet life more secure and fruitful in every aspect. In recent scenario, vulnerability is increasing by leaps and bound and web services are often deployed with critical application security faults that prone them to malicious attack and out of all these vulnerability, the most common web application vulnerability is Cross-Site Request Forgery (CSRF). In this paper, we explore the recent challenges faced in CSRF attack, different ways to launch CSR attack and its prevention techniques.**

Keywords*— *Cross-Site Request Forgery; CSRF; Sea Surf; Session Riding; One-Click attack; XSRF; application vulnerability; cross-origin request; security tokens; origin header; malicious URL; referrer header

I. Introduction

Cross-Site Request Forgery (CSRF) is an attack or technique to perform unwanted actions on a web application in which netizens are currently authenticated user. With a small help of social engineering techniques (chat, visiting malicious website, or sending a link via e-mail), CSRF attack can be executed. CSRF attacks mainly target the requests which will help in changing the state of application. While setting the target, the attacker will not steal any of the data and there is no way through which the attacker can view the response to the forged request [1].

Possibility of scenario: (1) If the victim is a regular user, a successful CSRF attack can be launched by the victim by just clicking, which will unknowingly of the user will perform state changing request like transferring funds, changing the email-address of user and many more. (2) If the victim has an administrator account, entire web application security can be compromised by CSRF attack.

CSRF is an attack that lures the user to submit a malicious link or request. In most of the sites, browser automatically sends requests which contains any of the authenticated credentials associated with the site, such as the user's IP address, session cookie, and Windows domain credentials. Therefore, if the user is currently authenticated to use the services of the site, the site has no available way to find the differentiation between the fake request originated by the victim and a valid request originated by the victim [1].

Synonyms: XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery, and Hostile Linking. CSRF attack is referred as One-click attack by Microsoft in its threat modeling process [1].

II. How does the attack work

There are many methods which can be used by the attacker to outwit the victim into submit information as well as loading information to a web application. In order to execute an attack, most importantly we need to understand how to generate a valid malicious request for our victim to execute. Take one scenario like:

Alice wishes to transfer $300 to Bob using the abcbank.com web application which is sensitive to CSRF. Hacki, an attacker, will trick Alice to send the money to him instead of Bob. The attacker will make the following steps:

1. Building script or exploit URL
2. Luring Alice to execute the action by social engineering

A. *GET scenario*

If the application was developed in a way to mainly use GET request to launch action and transfer argument, the money transfer request will be like:

```
GET http://abcbank.com/transfer.do?acct=BOB
&amount=300 HTTP/1.1
```

Hacki now decide to take advantage of this web application weakness by using Alice as his victim. Hacki will first construct the following exploit URL which will transfer $30,000 from Alice's account to his account. He takes the original command URL and replaces the recipient name with Hacki, and the desired amount at the same time:

```
http://abcbank.com/transfer.do?acct=HACKI&amount=30
000
```

By using social engineering this attack can be implemented with the help of following techniques:

1) Sending email with HTML content

Planting the malicious URL or script on web-pages that are likely to be visited by the victims while they are currently logged-in or doing online banking. The malicious URL will can be masked as an ordinary link, which will encourage the victim to click on it:

```
<a
href="http://abcbank.com/transfer.do?acct=HACKI&amo
unt=30000">View my Profile!</a>
```

2) fake image as a 0x0:

```
<img
src="http://abcbank.com/transfer.do?acct=HACKI&amou
nt=30000" width="0" height="0" border="0">
```

If the above mentioned image tag is included in the email, Alice can't see anything. However, the browser will still submit the request to abcbank.com without any visual indication which shows that the transfer has been taken place.

B. POST scenario

The execution process used by developer to execute the action is the only difference between GET and POST attacks. Let us imagine the bank is using POST method and the request which is vulnerable looks like this:

```
POST http://abcbank.com/transfer.do HTTP/1.1

acct=BOB&amount=100
```

The above mentioned type of request cannot be implemented successfully by using a standard IMG or A tag but can be delivered using FORM tags.

```
<form action="http://abcbank.com/transfer.do"
method="POST">
<input type="hidden" name="acct" value="HACKI"/>
<input type="hidden" name="amount" value="30000"/>
<input type="submit" value="View my Profile"/>
</form>
```

User interaction will be required in this form which means user have to click on the submit button, otherwise this can also be executed automatically by using JavaScript:

```
<body onload="document.forms[0].submit()">
<form...............  </form>
```

C. Other HTTP methods

Latest web application are frequently using other HTTP methods, such as DELETE or PUT. Let's assume this vulnerable bank web application uses PUT that takes a JSON block as an argument:

```
PUT http://abcbank.com/transfer.do HTTP/1.1

{ "acct":"BOB", "amount":100 }
```

With the help of JavaScript embedded into a vulnerable page, this type of request can be executed easily:

```
<script>
function put()
 {
          var x = new XMLHttpRequest();
          x.open("PUT","http://abcbank.com/transfer.do",t
rue);
          x.setRequestHeader("Content-Type",
"application/json");
          x.send(JSON.stringify({"acct":"BOB",
"amount":100}));
}
</script>

<body onload="put()">
```

This technique can be prevented by using same-origin policy restrictions. By default when the target website specially opens up cross-origin requests from the attacker's origin by using CORS with the following header, same-origin restriction is enabled:

```
Access-Control-Allow-Origin: *
```

III. CSRF Defense: Automated

We suggest two separate checks as the standard CSRF defense in which user interaction is not required. This technique is ignored for the command that deliberately allow cross origin requests. Your defenses will have to adjust according to the permission that is “permission allowed” or “permission not allowed” [1].

A. Check headers to verify the request is from same origin

This check will require two steps:

- ✓ Identifying the origin from which request is coming (Source Origin)
- ✓ Identifying the target where the request is going (Target Origin)

1. Identifying Source Origin:

For identification of source origin, use these two standard headers that almost include all requests, one or both of them [1]:

- Origin Header
- Referer Header

Checking the Origin Header:

If the Origin header is available in your header, verify the value of header and match this value to the target origin. CSRF and other Cross-domain attacks can be defended by using a method defined as the Origin HTTP Header standard. Dissimilar to the Referer header, the Origin header will be available in HTTP requests that begin from HTTPS URL. If the Origin header is present, then it should be verified to make sure it is same as the target origin [1].

```
Origin: http://abc.exapmle
```

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Origin: http://abc.exapmle
```

Checking the Referer Header:

If the Origin header is not present, then check the hostname in the Referer header which is same as the target origin. Referer checking is the most commonly used method for prevention of CSRF attack on embedded network device because it does not require any per-user state [1].

```
Referer: http://abc.ecample
```

2. Identifying the Target Origin:

Identification of the target origin is not an easy task. The first process is to simply grasp the target origin (i.e., its hostname and port #) from the URL in the request. However, the original URL is different from the URL which is actually received by the app server because the application server is usually sitting behind one or more proxies. If the user can directly access the application server then it is fine to use origin in the URL [1].

If you are behind a proxy, then there are a few options to be considered:

1. Configure your application to simply know its target origin
2. Use the Host header value
3. Use the X-Forwarded-Host header value

B. CSRF token

There are numerous ways available which can be specifically used to repel against CSRF. We recommend the following:

1. Synchronizer Tokens (it requires session state) No server side state is required in this approach [1].:
2. Double Cookie Defense[1]
   Encrypted Token Pattern[1]
3. Custom Header – e.g. X-Requested-With:XMLHttpRequest[1]

IV. CSRF Defense: Require User Interaction

Sometimes involvement of user while doing the transaction helps in prevention of unauthorized transactions

- One-time Pass code
- Re-Authentication
- CAPTCHA

Example: Use of security tokens in a transparent manner

Step 1: user provide email address and amount of money they want to send.

Response: server will response with conformation dialog that will contain a unique ID for this transaction.

Step 2: user confirms the transaction

Response: Transaction ID which is generated in step 1 is included in the confirmation request. CSRF attack can be prevented against this transaction flow without using a CSRF specific defense token, if the server checks that the transaction ID is authentic and this authentic transaction ID is a part of this step. If the attacker cannot predict the transaction ID then the transaction is safe from the exploitation of CSRF vulnerability. The transaction ID should be random/unguessable, unique and different from the next transaction number generated in list of transaction IDs.

V. Personal Safety

- Quickly Log-off after using website
- Don’t save username/passwords in browser
- Do not allow sites to remember your login
- Do not use same browser or (Tab) to access websites and surfing when your banking sites credential opened in another TAB.

CONCLUSION

At the end of this paper we have reached to the conclusion that CSRF is a website vulnerability which is less known to website developers so it is exist in many websites. There are many famous websites which is still vulnerable to this type of attack. It is really difficult to preserve against this attack, but there are many ways which can help to protect against this. The most effective way is use of random token generator. This is used by many web sites to protect against CSRF.

REFERENCES

[1] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[2] https://developer.mozilla.org/enUS/docs/Web/HTTP/Access_control_CORS

[3] Purnima Khurana and Purnima Bindal, "Vulnerabilities and D efensive Mechanism of CSRF", International Journal of Computer Trends and Technology (IJ CTT), volume 13, No. 4, 2231-2803, Jul 2014

[4] http//en.wikipedia.org/wiki/Cross-site-request forgery

[5] https://www.owasp.org/images/1/19/OTGv4.pdf

[6] Xiaoli Lin, Pavol Zavarsky, Ron Ruhl and Dale Lindskog, "Threat Modeling for CSRF Attacks", Internationa Conference on Computational Science and Engineering(ICCSE), 978-0-7695-3823-5/2009

[7] RadhaRani Sankuru and Madhubabu Janjanam, "Web application Security- Cross Site Request Forgery attacks", International Journal of Computer Science & Engineering Technology(IJCSET), volume 4, No. 08, ISSN: 2229-3345, Aug 2013

[8] Sentamilselvan.K, Lakshmana Padndian.S & Ramkumar.N, "Cross Site Request forgery: Prevention Measures", International Journal of Computer Applications, Volume 106, No. 11, 0975-8887, November 201