

### Experiment No: 3

**Student's Name:** Rajiv Paul

**UID:** 20BCS1812

**Section/Group:** 702/A

**Subject Code:** 20CSP-338

**Subject Name:** WMS\_Lab

**Date of performance:** 31/8/2022

**Branch:** CSE

**Semester:** 5<sup>th</sup>

**Aim:** Working of CSRF (cross site request forgery) attack/ Vulnerability.

**Objective:** To understand how to find CSRF Vulnerability.

**Software/Hardware Requirements:** Windows 7 & above version

**Tools to be used:** OWAPP website, Notepad, Burpsuite

#### Discussion:

CSRF : Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

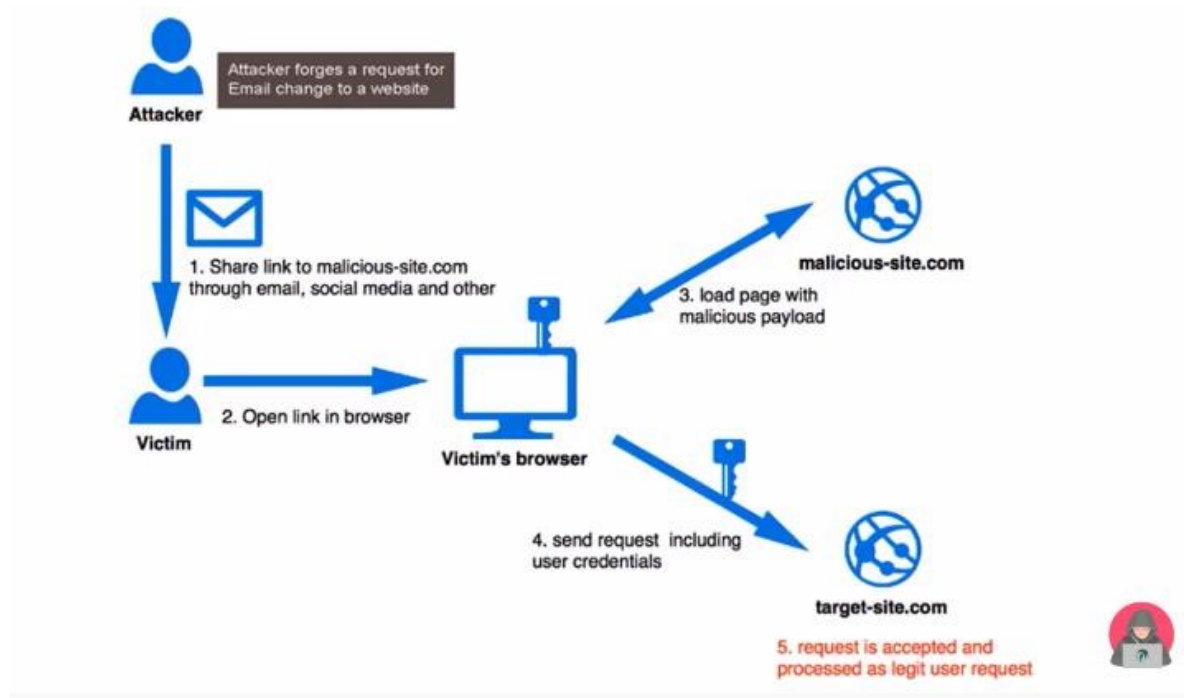
#### Attack Surfaces:

The attack surfaces for CSRF are mostly HTTP requests that cause a change in something related to the victim, for example: name, email address, website and even password. It is sometimes used to alter the state of authentication as well. (Login CSRF, Logout CSRF) which are less severe but can still be problematic in some cases

#### Synonyms

CSRF attacks are also known by a number of other names, including XSRF, "Sea Surf", Session

Riding, Cross-Site Reference Forgery, and Hostile Linking. Microsoft refers to this type of attack as a One-Click attack in their threat modeling process and many places in their online documentation.



### Exploitation:

Consider a website example.com and the attacker's website evil.com. Also assume that the victim is logged in and his session is being maintained by cookies. The attacker will:

- Find out what action he needs to perform on behalf of the victim and find out its endpoint (for example, to change password on target.com a [POST request](#) is made to the website that contains new password as the parameter.)
- Place [HTML code](#) on his website evil.com that will imitate a legal request to target.com (for example, a form with method as post and a hidden input field that contains the new password).
- Make sure that the form is submitted by either using "autosubmit" or luring the victim to click on a submit button.

When the victim visits evil.com and that form is submitted, the victim's browser makes a request to target.com for a password change. Also the browser appends the cookies with the request. The server treats it as a genuine request and resets the victim's password to the attacker's supplied value. This way the victim's account gets taken over by the attacker.

### **Prevention:**

- **On user side:**

User side prevention is very inefficient in terms of browsing experience, prevention can be done by browsing only a single tab at a time and not using the “remember-me” functionality.

- **On Server Side:**

There are many proposed ways to implement CSRF protection on server side, among which the use of CSRF tokens is most popular. A CSRF token is a string that is tied to a user's session but is not submitted automatically. A website proceeds only when it receives a valid CSRF token along with the cookies, since there is no way for an attacker to know a user specific token, the attacker can not perform actions on user's behalf.

### **Prevention measures that do NOT work**

A number of flawed ideas for defending against CSRF attacks have been developed over time. Here are a few that we recommend you avoid.

#### **Using a secret cookie**

Remember that all cookies, even the *secret* ones, will be submitted with every request. All authentication tokens will be submitted regardless of whether or not the end-user was tricked into submitting the request. Furthermore, session identifiers are simply used by the application container to associate the request with a specific session object. The session identifier does not verify that the end-user intended to submit the request.

#### **Only accepting POST requests**

Applications can be developed to only accept POST requests for the execution of business logic. The misconception is that since the attacker cannot construct a malicious link, a CSRF attack cannot be executed. Unfortunately, this logic is incorrect. There are numerous methods in which an attacker can trick a victim into submitting a forged POST request, such as a simple form hosted in an attacker's Website with hidden values. This form can be triggered automatically by JavaScript or can be triggered by the victim who thinks the form will do something else.

#### **Multi-Step Transactions**

Multi-Step transactions are not an adequate prevention of CSRF. As long as an attacker can predict or deduce each step of the completed transaction, then CSRF is possible.

## URL Rewriting

This might be seen as a useful CSRF prevention technique as the attacker cannot guess the victim's session ID. However, the user's session ID is exposed in the URL. We don't recommend fixing one security flaw by introducing another.

## HTTPS

HTTPS by itself does nothing to defend against CSRF.

However, HTTPS should be considered a prerequisite for any preventative measures to be trustworthy.

## Examples

### How does the attack work?

There are numerous ways in which an end user can be tricked into loading information from or submitting information to a web application. In order to execute an attack, we must first understand how to generate a valid malicious request for our victim to execute. Let us consider the following example: Alice wishes to transfer \$100 to Bob using the *bank.com* web application that is vulnerable to CSRF. Maria, an attacker, wants to trick Alice into sending the money to Maria instead. The attack will comprise the following steps:

1. Building an exploit URL or script
2. Tricking Alice into executing the action with [Social Engineering](#)

### GET scenario

If the application was designed to primarily use GET requests to transfer parameters and execute actions, the money transfer operation might be reduced to a request like:

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

Maria now decides to exploit this web application vulnerability using Alice as the victim. Maria first constructs the following exploit URL which will transfer \$100,000 from Alice's account to Maria's account. Maria takes the original command URL and replaces the beneficiary name with herself, raising the transfer amount significantly at the same time:

http://bank.com/transfer.do?acct=MARIA&amount=100000

The [social engineering](#) aspect of the attack tricks Alice into loading this URL when Alice is logged into the bank application. This is usually done with one of the following techniques:

- sending an unsolicited email with HTML content
- planting an exploit URL or script on pages that are likely to be visited by the victim while they are also doing online banking

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

as a 0x0 fake image:

```

```

If this image tag were included in the email, Alice wouldn't see anything. However, the browser *will still* submit the request to bank.com without any visual indication that the transfer has taken place.

A real life example of CSRF attack on an application using GET was a [uTorrent exploit](#) from 2008 that was used on a mass scale to download malware.

## POST scenario

The only difference between GET and POST attacks is how the attack is being executed by the victim. Let's assume the bank now uses POST and the vulnerable request looks like this:

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

```
acct=BOB&amount=100
```

Such a request cannot be delivered using standard A or IMG tags, but can be delivered using a FORM tags:

```
<form action="http://bank.com/transfer.do" method="POST">
```

```
<input type="hidden" name="acct" value="MARIA"/> <input type="hidden" name="amount" value="100000"/> <input type="submit" value="View my pictures"/>

</form>
```

This form will require the user to click on the submit button, but this can be also executed automatically using JavaScript:

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

<form...
```

### Other HTTP methods

Modern web application APIs frequently use other HTTP methods, such as PUT or DELETE. Let's assume the vulnerable bank uses PUT that takes a JSON block as an argument:

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

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

Such requests can be executed with JavaScript embedded into an exploit page:

```
<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>

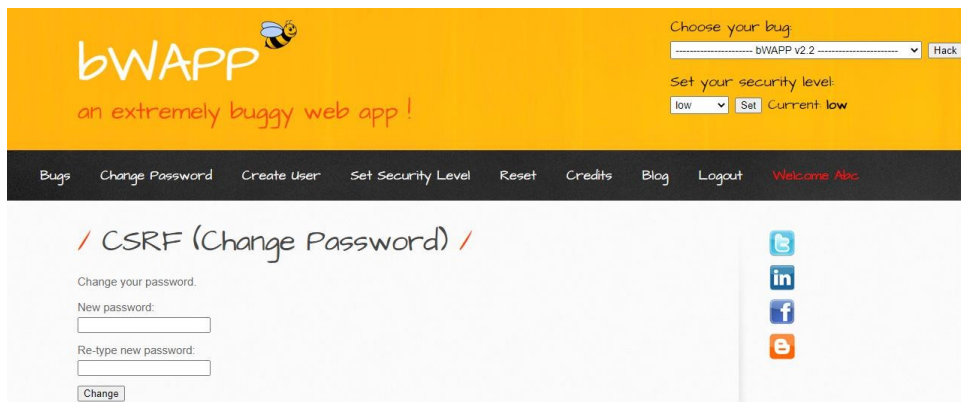
<body onload="put()">
```

Fortunately, this request will **not** be executed by modern web browsers thanks 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:

### Steps/Methods/Code:



1. Open a website bwapp Login page. Login with credentials given. If not getting login then create new user with a temporary id and then re-login with new id.
2. Now choose “csrf-change password” category and click on hack. (GET Method)

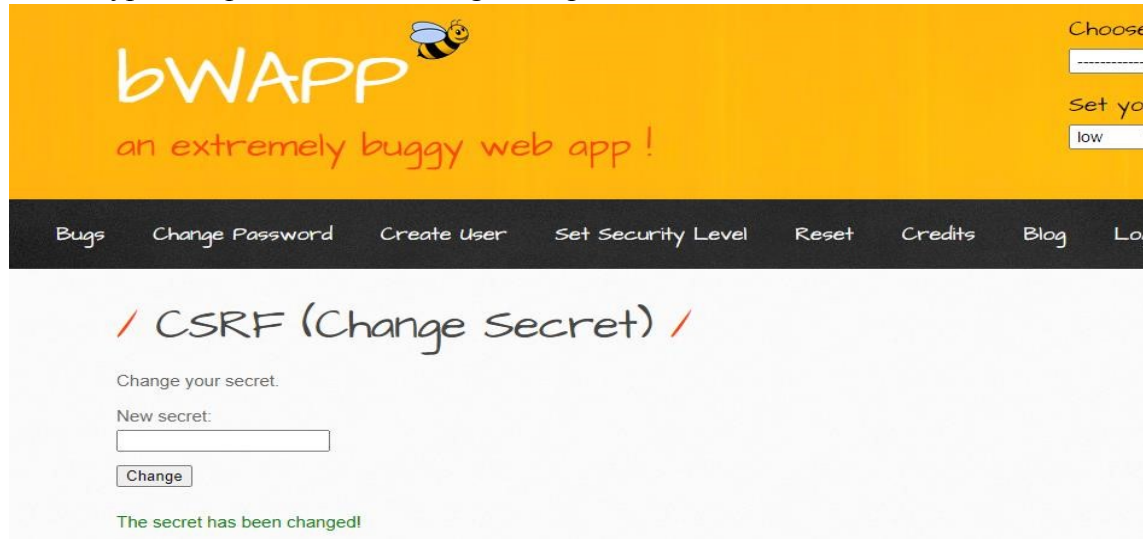


3. Type new password and login again with new password. You will see message that password is changed. You can see the changed password parameters on URL also. Attacker can send the URL to victim and can have access to the password or even can change the password by changing in URL.



4. Paste URL in notepad, change the password in URL and then copy paste it in new tab. Page will get opened. Then login again with this new password. It will execute successfully.

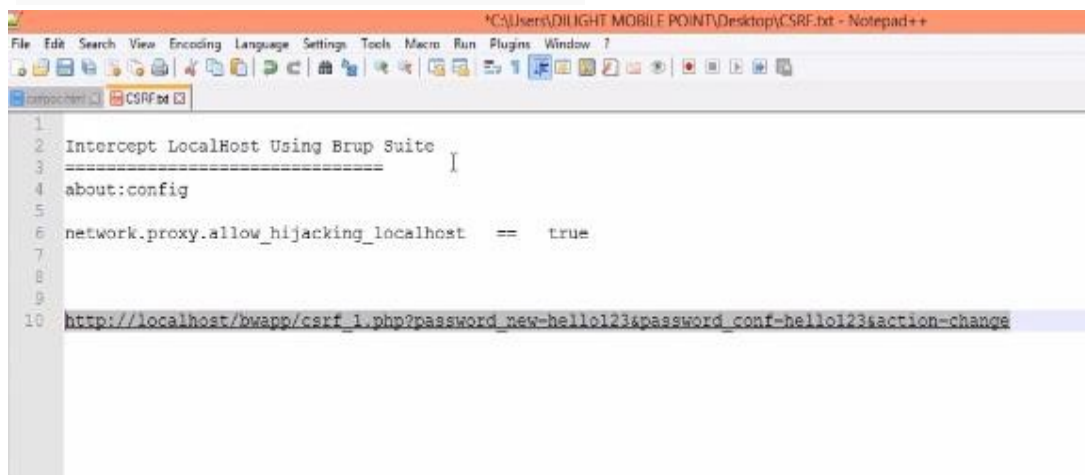
- Now choose another option CSRF-Change secret and click on Hack. This is POST based CSRF. Type new password and change it. Input not reflected in URL.



- To exploit POST based CSRF, we will use burp suite tool. Configure our browser so that burp proxy can intercept local host.

**Intercept LocalHost Using Brup Suite =====  
about:config**

**network.proxy.allow\_hijacking\_localhost ==  
true**

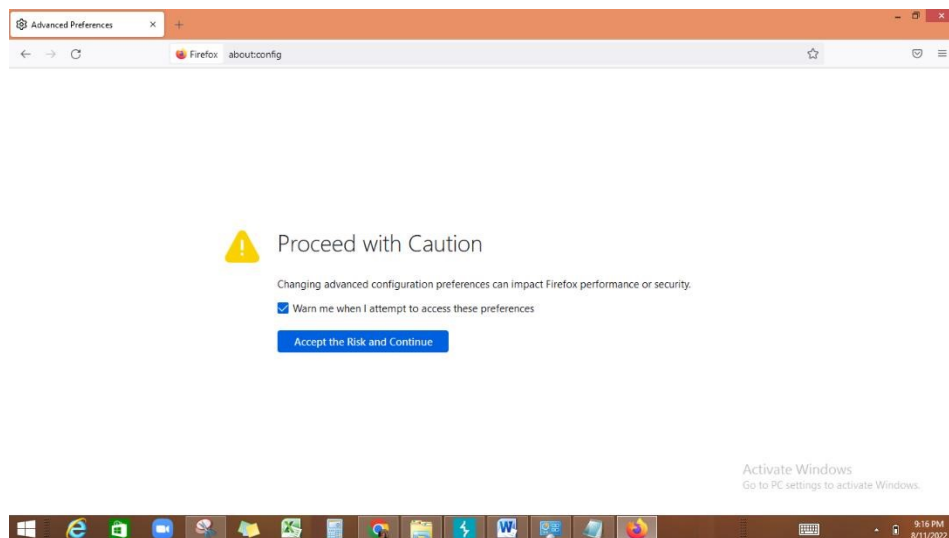
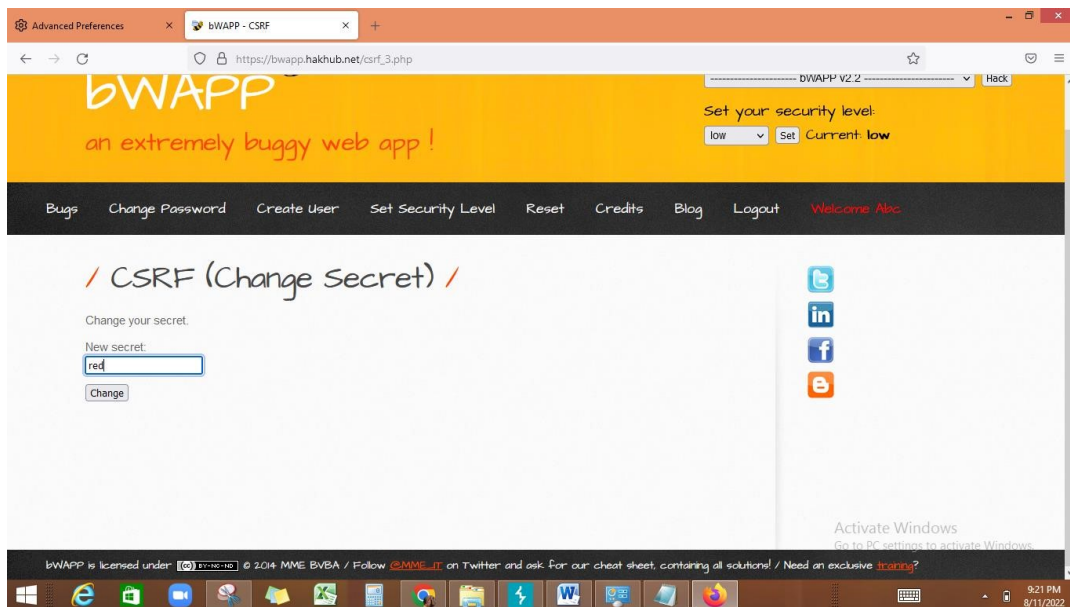


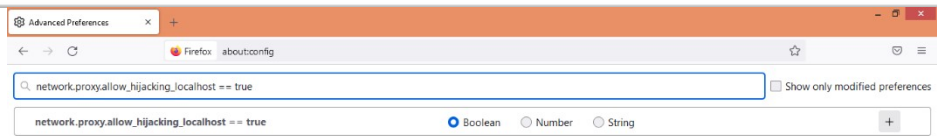
- Copy and paste: about: config(shown in screenshot) in MOZILLA FIREFOX browser. Click on accept risk as shown in screenshot. Write **network.proxy.allow\_hijacking\_localhost** in search field. And make it true(double



click). Then write any secret password in bwapp login and make **burpsuite** intercept On then click on change. Request will go to burpsuite tool.

Config burpsuite on Mozilla : <https://youtu.be/o3gsVWhacjk>





Login with changed secret in bwapp app. It will be run

## Learning Outcomes

1. Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated.
2. CSRF attacks exploit the trust a Web application has in an authenticated user.