Cross Site Request Forgery: Lab Exercise
IN618 Security

Introduction

Last time we saw how an XSRF vulnerability can be exploited. In this exercise we will take the code for the vulnerable application and modify it to remove the XSRF vulerability by adding an extra token to our forms. The key idea is that we will add and track a bit more state as the user traverses the application.

All work will be done the the file `public_html/xsrf/home.php`. This file and all others needed to run and test your work have already been placed in the `xsrf` directory inside your own `public_html` directory.

1 Tasks

First we need to be able to track user sessions and store information about them on the server. Fortunately, we do not have to implement this ourselves, since PHP includes this as a core feature. Just add the line

```
session_start();
```

at the very beginning of the file. This will cause the server to send a session cookie to the user's browser and to store session information on the server.

Next, we need to add a special session token to the web forms we produce on this page. Add the code below after line 31 (assuming you've added only the one line above so far).

```
$xsrf_token = md5($token . rand() . microtime());
$_SESSION['saved_xsrf_token']=$xsrf_token;
```

The first line creates a reasonably unpredictable token that we can add to our forms. We take the user's login token and concatenate it with a random number and the server's system time in microseconds. We then MD5 hash that value. On the second line we save that value in the user's session. We can retrieve that value when the page loads again later.

Now we need to add the token value to the HTML forms we write. There are two such forms on this page. To each one, add the following:

```
<input type="hidden" name="xsrf_token" value="<?php echo($xsrf_token) ?>" />
```

So now, in a way that is largely transparent to the users, we include this token value with their form submission to prove that they submitted this form properly. If an attacker could intercept this value along with the user's session cookie and login cookie she could spoof the form submission, but only for a short time since the XSRF token changes every time the user loads the page.

Now we just need to verify the XSRF token when we process the form submissions. We do this at about line 26. Find that part of the code, where we look at the `$action` value. Wrap that entire `if-elsif` block with an outer if test like this:

```
if($_SESSION['saved_xsrf_token'] == $_REQUEST['xsrf_token']) {

  # form handling stuff that's already there

}
```

We compare the token submitted in the request (`$_REQUEST['xsrf_token']`) with the one stored in the user session (`$_SESSION['saved_xsrf_token']`) to see that they match.

This is all the code we need to add. Test your modifications to see that they work. Be sure to view the HTML source of your forms to see that the new XSRF token is included. Both forms submit GET requests so that you can see the form fields that are submitted. Check to see that you can copy those urls and try submitting them directly, but they don't work. Do you know why that is?

2 Submission

In addtion to testing your code on the server, submit a copy of your code to your SVN repository by Thursday, 26 March.

3 Resources

The files you need to start this assignment have been copied to your `public_html` directory on the `sec-student` server.

4 Marking

There are 10 marks for this exercise:

- Modifying and testing your code on the `sec-student` server: 5 marks;
- Committing correct and well-formatted code to SVN : 5 marks;