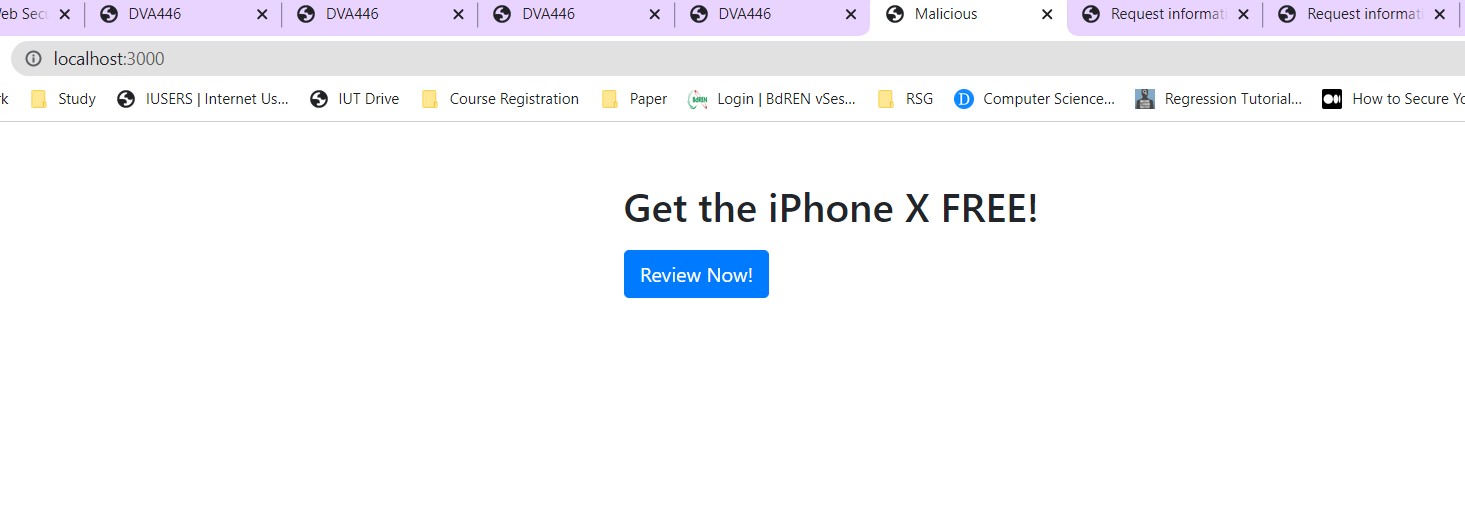
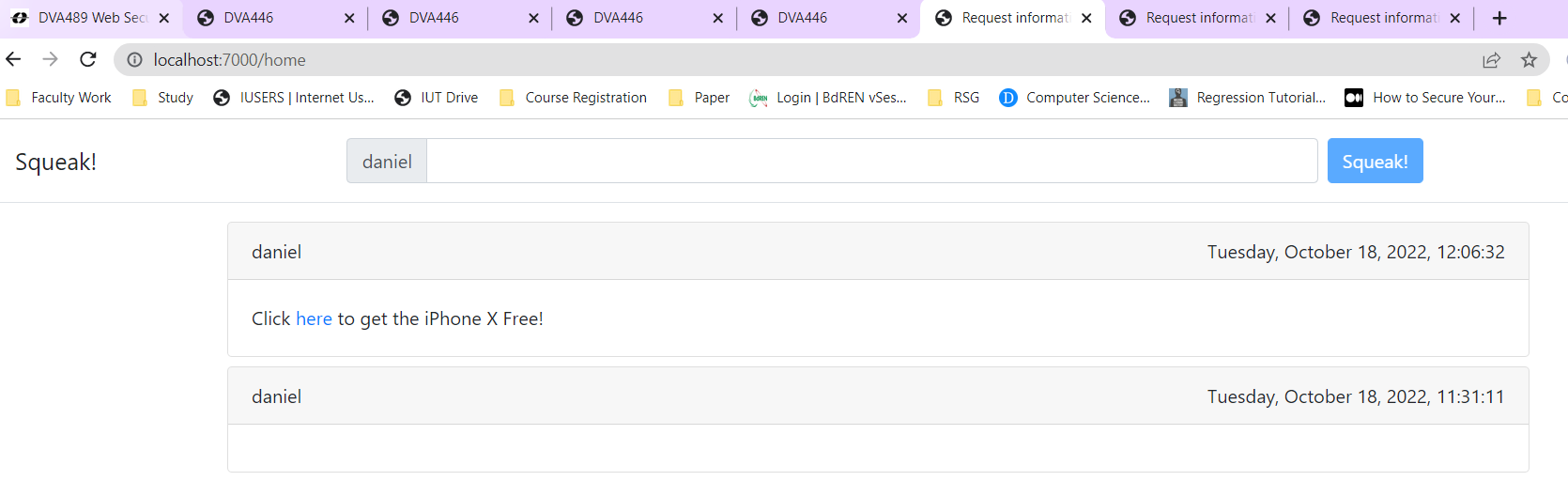
**Cross Site Request Forgery (CSRF)**

The CSRF attack is an attack launched by malicious site to another site where the user is currently authenticated. This attack is made possible because of automatically sending the authorization information (such as cookies) with every request. The unpatched version of our application is intentionally made vulnerable to CSRF attacks. We host the unpatched version at **localhost:7000**. To demonstrate the CSRF attack, we craft another attacker-controlled homepage and host it at **localhost:3000**. The malicious site contains a single button labelled “Review Now” which automatically posts a squeak containing a link to the malicious site in the Squeak! homepage when an authenticated user clicks on it.





Clicking the link contained in the post will redirect any user to the malicious site where they may again fall victim to the attack.

**Protecting the application**

The patched version of our application is hosted in **localhost:8000**. In order to protect our application from the CSRF attack, we associate each session with a randomly generated CSRF token. This token is created every time a user successfully logs in from the Sign In/Sign Up page, so basically the **/signin** route initiates the creation of the random CSRF token. In our implementation, the **/signup** route does not create a valid session, rather it only leads to adding a new user in our “passwd” file so we have not associated any CSRF token with this route. Next, this CSRF token is stored in SESSION\_IDS that is declared within the server and it is stored in an index that is based on the cookie.

  var cookie = req.cookies.squeak\_session;

  var sessionID = cookie.sessionid;

  req.session = {};

  if (!req.session.csrf) {

    req.session.csrf = uuid.v4();

  }

  SESSION\_IDS[sessionID] = req.session.csrf;

The CSRF token is also embedded in a hidden field with the squeak form page. When a new squeak is about to be posted, we check the embedded token with the value that is stored in the server and if it does not match, then we simply prevent the post from being added to our “squeaks” file.

  if(req.body.csrf == SESSION\_IDS[cookie.sessionid] && req.body.csrf)

  {

    flag = true;

  }

The **/signout** route clears the cookie, which makes our current CSRF token unlocatable, thus making our application ready to create a new CSRF token and store it at the new location based on the new cookie that is created for the new valid session whenever we encounter the **/signin** route again.