Nick Weaver
Fall 2018

CS 161
Computer Security

Homework 3

Due: Friday, 19 October 2018, at 11:59pm

Instructions. This homework is due **Friday, 19 October 2018, at 11:59pm**. No late homeworks will be accepted unless you have prior accomodations from us. This assignment must be done on your own.

Create an EECS instructional class account if you have not already. To do so, visit https://inst.eecs.berkeley.edu/webacct/, click "Login using your Berkeley CalNet ID," then find the cs161 row and click "Get a new account." Be sure to take note of the account login and password, and log in to your instructional account.

Make sure you have a Gradescope account and are joined in this course. The homework *must* be submitted electronically via Gradescope (not by any other method). Your answer for each question, when submitted on Gradescope, should be a single file with each question's answer on a separate page.

Problem 1 ***True-or-False Questions*** **(10 points)**

Answer each question. You don't need to justify or explain your answer.

(a) TRUE or FALSE: Prepared statements are a good defense against SQL injection.

True

(b) TRUE or FALSE: Setting the "secure" flag on a cookie (so it will only be sent over HTTPS) is a good defense against CSRF.

False

(c) TRUE or FALSE: Setting the "secure" flag on a cookie (so it will only be sent over HTTPS) is a good defense against XSS cookie-leaking.

False

(d) TRUE or FALSE: Setting the "HTTPOnly" flag on a cookie is a good defense against XSS cookie-leaking.

True

(e) TRUE or FALSE: Switching over all application requests to HTTP Post stops all CSRF attacks.

False

(f) TRUE or FALSE: SOP prevents XSS attacks.

False

(g) TRUE or FALSE: Two Javascript scripts embedded in pages running in two different tabs on a user's browser can never access the resources of each other.

False

(h) TRUE or FALSE: Two Javascript scripts embedded in pages running in two different tabs on a user's browser can never access the resources of each other.

False

(i) TRUE or FALSE: Browsers have a private browsing mode, which prevents websites from storing cookies on your computer altogether.

False

(j) TRUE or FALSE: Because of the cookie policy, you cannot be tracked across domains by cookies.

False

Problem 2 ***Web Security Warm-Up*** **(15 points)**

(a) Oski owns a conglomerate, OskiBankAndServices.com. He hopes to compete with Google by combining online banking together with web services, such as web hosting. As part of his business plan, Oski decides to host a website creation service at `oskiwebhosting.com/[SITENAME]`. This service allows you to choose your own `SITENAME` and upload any script or HTML that you desire. Why is this a better design than putting user sites on `OskiBankAndServices.com/sites/[SITENAME]`?

> This is a better design because this service allows any script of HTML to be uploaded, so a user could construct a site on this domain to steal cookies, and becaue the domain is the same, this user could steal these bank cookies and log in to the bank as you.

(b) Your friend Chad has decided to create a new microblogging service for aspiring presidential candidates but with the option to choose your intended audience. This way if you want to post something to pander to your base you can do so without offending another demographic! He informs you that he can handle the business side and tasks you with building the web-based sharing form, PresidentialTweets.gov. You have set up a simple form with just two fields, the text to share and the intended audience. When a user clicks submit, the following request is made:

```
https://www.presidentialtweets.gov/share?
    text=<the text to share>&audience=<the chosen demographic>
```

You show this to your bro Vladimir, and he thinks there is a problem. He later sends you this message:

```
Hey, check out this cute cat picture.  http://tinyurl.com/Cute161Kitty
```

You click on this link and later find out that you have created a post shared with "`voting-demographics`" with the text "I build the best aircraft carriers this country has ever seen, SAD". (TinyURL is a URL redirection service. Whoever creates the link can choose whatever URL it redirects to.)

How was this post created?[1] What URL would cause this to happen? Write the link in your solution.

[1] A reminder: in URLs, spaces are encoded as `%20`.

```
https://www.presidentialtweets.gov/share?
text=I%20build%20the%20best%20aircraft%20carriers%20this%20
country%20has%20ever%20seen%20SAD&audience=voting-demographic
```

This post was created by Vladdy using a CSRF attack, where when a user clicks submit, the link provided is the request made, without any authentication. By revealing this, Vlad can link this url, and hide the request using a tinyurl URL shortener, so when you click on this link, it runs the preloaded request that Vlad has customized to send the message he wants.

(c) Continuing from part (b), what attack is this and how could you defend your form from the sort of attack listed in part (b)? Explain in 1–2 sentences.

To defend against these sorts of CSRF attacks, we could use an unique CSRF token associated with the form and user, such that when a request is made, it needs the correct token to execute. This way, then you click on the tinyurl link and run Vlad's request, it would need a correct token to run, otherwise it would fail.

Problem 3 ***Attempted Web Security*** **(10 points)**

(a) When users of bank.com are logged in, a request to `bank.com/session.js` returns a Javascript file containing

```
let sessionID = "0123456789";
```

except that 0123456789 is replaced with the session ID for the user who made the request.

An attacker controls `evil.com` and would like to learn Alices session ID for `bank.com` . How can the attacker do this? Explain why the same-origin policy doesnt stop this attack. (Assume the attacker can get Alice to visit `evil.com` .)

send a link to alice that points to an external script, evilscript, that has :
let sessionID = "SOME PREDETEMINED KNOWN ID"
This way we can set the Alice's session ID to whatever we want, and know the seesion ID in the process. SOP doesnt help here because the origin is the same: bank.com.

(b) When bank.com learns of this problem, they fix it by beginning all Javascript files with

```
if
   (!document.location.includes("http://bank.com")) { while (1) {}
   // infinite loop

}
```

Explain why this doesnt work. How could an attacker defeat this defense?

This could be easily defeated if we feed

```
"http://bank.com"
```

somewhere in the fake link. This way, this script wont catch the fake script pointer, and we can set Alice's session id.

Problem 4 ***SQL Injection*** **(15 points)**

You are discouraged to find the following Java code in the client login section of an online banking website:

```
 /** * Check whether a username and password
combination is valid.  */ ResultSet checkPassword(Connection conn,
String username, String password) throws SQLException { String query
= "SELECT userID FROM Customers WHERE username = '" + username + "'
AND password = SHA256('" + password + "');"; Statement s =
conn.createStatement(); return s.executeQuery(query);

}
```

Assume that before issuing a request, the bank's server calls `checkPassword` and ensures that the returned `ResultSet` contains exactly one `userID`. If this check fails, the bank fails the request. Otherwise the request is issued as the user represented by `userID`.

Note: if there are 0 user IDs in the ResultSet then the username and/or password are wrong. If there are more than one then something went wrong somewhere on the bank's end since usernames should be unique (and consequently limit results to at most one). For the purposes of this question, whats important is that the request goes through iff the ResultSet contains exactly one user IDs.

(a) What username could an attacker enter in order to delete the Customers table?

```
"alice'; DROP TABLE Customers; --"
```

(b) What username could an attacker enter in order to issue a request as user "Admin", without having to know the password?

```
SELECT * FROM Customers WHERE username = 'admin'--' AND password = 'password'
```

(c) When you point this out to the development team, a junior developer suggests simply escaping all the single quotes with a backslash. For example, the following line could be added to the top of the function:

```
username =
   username.replaceAll("'","\\\\'");
```

This code replaces each ' in the username with \' before including it in the SQL query.

Modify your answer to part (b) above so it will work against this new code. Assume the database engine accepts either ' or " to enclose strings.

```
"alice\'; DROP TABLE Customers; --"
```

Problem 5 ***XSS Game*** **(15 points)**

Visit https://xss-game.appspot.com/ and complete the first 3 levels. This game is similar to Project 1, except youll be exploiting XSS vulnerabilities instead of buffer overflows. You may use the hints provided by the game.

For each level, describe the vulnerability and how you exploited it in 2-3 sentences. Show the code that you used or what you typed into the input fields.

We recommend using the Chrome browser for this.

(If you enjoyed this, check out http://overthewire.org/wargames/natas/ for more!)

Part1:
¡script¿alert("This is an alert");¡/script¿
Here we are basically just insertting script to be run. Our script tells the browser to make an alert.
Part2:
¡h1 onclick="alert("This is an alert)"¿1¡h1/¿
Here, we are making a clickable piece of text, that runs script to make an alert. Upon being clickedm the script is run.
Part3:
https://xss-game.appspot.com/level3/frame1' onclick='alert("This is an alert")' x='
Here we are changing the image into a clickable thing, that runs the script. In this case, it runs script making an alert.

Problem 6 ***Feedback*** **(0 points)**

Optionally, feel free to include feedback. Whats the single thing we could do to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better? If you have feedback, submit your comments as your answer to Q6.

Everything is great!