Weaver
Fall 2017

CS 161
Computer Security

Homework 3

Due: Tuesday, October 31st, at 11:59pm

Instructions: This homework is due **Tuesday, October 31st, at 11:59pm** on Gradescope. No late homeworks will be accepted unless you have prior accommodations from us. This assignment must be done on your own.

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. Please don't forget to assign pages to questions on your submission.

Hope your semester is going well and good luck with the midterm!

Problem 1 ***Web Security Warm-Up*** **(20 points)**

(a) In class we learned about the Same-Origin Policy and the cookie policies for the browser and the server. Nicholas Dirks says that you should be careful of visiting any untrusted site, because the site's owners can read cookies from *any* other websites they want. Is Nick right? Explain in 1–2 sentences why or why not. For this question, only focus on cookies.

(b) 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]`?

(c) 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.

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

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

Problem 2 ***Yum, cookies!*** **(20 points)**

Your favorite 161 classmate Outis is creating a web hosting service to compete with Oski Bank and Services. Outis has been serving users' websites from

`https://users.WhatIsOutisHiding.com/[USERNAME]/`

Anyone can sign up with a chosen username and build their own websites and applications with arbitrary HTML and JavaScript. Currently, they're discussing changing their service to serve the websites instead from

`https://[USERNAME].users.WhatIsOutisHiding.com/`

(a) Under the **current** system, a malicious site may be able to **read** some[2] cookies from another user's site. Describe a cookie that a site could set such that another user's site would **not** be able to read it.

(b) Under the **proposed** system, could a site still prevent other users' sites from reading its cookies? If so, how?

(c) Under the **current** system, a malicious site can **write** a cookie that will be sent to other users' sites. Describe a cookie that a malicious site could set such that it would be sent to other users' sites as well.

(d) Under the **proposed** system, could a malicious site still write cookies for other users' sites? If so, how?

(e) One user, who runs a password-protected blog, claims that, under the **current** system, readers who visit other users' sites while signed in to the blog can cause the secret content to be disclosed to an attacker.

They report that malicious site open a window with the blog, access the blog's DOM with JavaScript, and send the results to its own server. State why browsers allow this to happen.

(f) Under the **proposed** system, could a malicious site still steal secret content this way?

[2] Corrected. Previously said "any", which is inconsistent, because you're asked to find a counterexample.

Problem 3 ***SQL Injection*** **(30 points)**

Outis has taken his business to the next level and has joined the online banking industry. Because he is too busy answering 161 questions on Piazza, you have been hired to test the security of his site. You start off reviewing the code for 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 user_id 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* `user_id`. If this check fails, the bank fails the request. Otherwise the request is issued as the user represented by `user_id`.

Note: if there are 0 `user_id`s 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, what's important is that the request goes through iff the `ResultSet` contains exactly one `user_id`.

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

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

Note: regarding the use of four backslashes in the source code above: this arises due to there being two separate instances of escaping going on. First, the text:

```
"\\\\'"
```

is a string literal *in the Java source code*. The Java compiler interprets it as specifying a string whose literal *data* value is:

`\\'`

Second, the `replaceAll` method takes a regular expression for its second argument. It *parses* that regular expression, including processing any escape sequences in it. Given the above *data* value, it then interprets the regular expression as:

`\'`

and so it will change any instance of a single `'` to `\'`.

(You might wonder *why* `replaceAll` treats its second argument as a regular expression. It does so because it's looking for "backreferences" like "\1", which means "in this part of the replacement use the first sub-match in the original regular expression." That detail isn't important for this problem.)

(d) Explain how you would rewrite the `checkPassword` function. (Like tell us what technique should you apply to prevent injection attacks) Explain why your code is safe from SQL injection attacks.

Problem 4 ***Cache Money*** **(20 points)**

You are the founding developer for a new fancy payments startup called Cache Me Ousside, competing with Outis, and you are developing the web-based payment interface. You have set up a simple payment page, with two inputs: the amount to be paid and the recipient of the payment. When a user clicks a "Pay now" button, the following request is made to complete the transfer:

```
GET https://www.cacheMeOusside.com/payment?amount=[dollars]&recipient=[name]
```

The server then transfers the specified amount from the logged in user (determined by a session cookie) to the specified recipient.

(a) You show this to your friend Eve, and she thinks there is a problem. She later sends you this message:

```
Hey, check out this funny cat picture.  tinyurl.com/z2k52gs
```

You click on this link, and later find out that you have paid Eve $1. (TinyURL is a url redirection service, and whoever creates the link can choose whatever url it redirects to.)

What kind of attack did Eve use to steal $1 from you? What did the tinyurl redirect to? Write the link in your solution.

(b) Web browsers send a `Referer` [sic] HTTP header when navigating (such as submitting a form), which tells the server the URL from which the user came from.

Describe a way to use this feature to prevent Eve from stealing any more money from you in this way.

(c) A defense using the Referer header is promptly implemented.

Another developer adds a new feature: the payments page now shows a list of the user's friends, with custom avatars. A user sets up an avatar by specifying a URL, which will be put in an `<img>` tag next to their name. (This fancy startup only hires the best, so there are no XSS vulnerabilities with the implementation.)

You later find out that all of Eve's friends have paid her $1. Describe how Eve used this new feature to bypass the defense and steal money from her friends.

(d) You decide to add a defense that uses secret tokens to stop Eve's latest attack: the payments page includes a secret value in a hidden field that must be sent with the payment request. That way, fixed URLs from unscrupulous users won't result in a fraudulent payment.

Around the same time, the startup rolls out new "Pay me" buttons that can be embedded on other sites. These are implemented as `<iframe>`s which load a small payment page from

```
https://www.cacheMeOusside.com/payme/[recipient]/[dollars]
```

This small payment page only has a button; the recipient and dollar amount are taken from the `<iframe>`'s URL.

As you tell Eve about these recent updates, she proudly announces her own new website, which displays two random cat pictures and which invites the visitor to click on which one is cuter. You check it out—just for a few minutes, you swear. But an hour later you realize you're still clicking away when you receive a message from your bank saying your account is overdrawn.

You find that you've sent Eve all your money in $1 increments. What kind of attack did Eve use to steal all your money? Describe her attack.

Problem 5 ***True/False*** **(10 points)**

Answer true or false and explain your reasoning.

(a) TRUE or FALSE: Randomizing the DNS query identifier prevents an on-path attacker (sitting between the client and the DNS server) from spoofing DNS responses.

(b) TRUE or FALSE: One defense against the Kaminsky Attack is to randomize the destination port along with randomizing the identifier field.

(c) 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.

(d) TRUE or FALSE: The httpOnly flag of a cookie mitigates XSS attacks because it ensures the browser sends the cookie only over https.

(e) TRUE or FALSE: A website that rejects all user input that contains `<script>` tags may still be vulnerable to XSS attacks.

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

Optionally, feel free to include feedback. What's 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.