Tyler Jackson

CSE 7359 Software Security Final Exam
5/11/2015

1.)

a.

Cross site scripting is when a user inputs javascript code into somewhere in the web application that will then be displayed by the web application at some point. However, instead of the code being displayed, it is executed as javascript code. There are two types of XSS, stored and reflective. Stored XSS refers to if someone injected javascript code into a location that is stored on the server and then displayed to all users any time a page is loaded or button is pushed, etc. Reflective is when an input of some sort takes javascript and immediately spits it back out to the user. This XSS attack will therefore happen immediately. An example is a blog entry or a comment that the web application stores to be displayed whenever the page is loaded. If this page is loaded in the future the code will be executed every time. I would recognize a XSS vulnerability based on if an input from any point in the past was being printed to the screen anywhere without being properly encoded. This could be in a url, in a css attribute, in an html tag etc.

b.

This vulnerability is prevented by encoding the output contextually. For example if I am printing out a comment to a comments table I need to html encode it so that any malicious code will be interpreted as text, rather than executable code. Validating the input can also be useful, but sometimes you may want to allow special characters in an input field. For example, if stack overflow didn't allow users to input tag elements into comments and questions then the site would lose a lot of value. In this case, validating the input would be less useful as more characters would be acceptable. Output encoding however, would prevent XSS.

2.)

a.) CSRF is when you submit a form to a page as if the form originated from that page while in reality it was a form that you had created to look like the real thing. You would identify CSRF by attempting to submit a form that you had created and see if it is accepted. For example, I could create a form with malicious code, send it to another tester and allow him to click a link which would attempt to submit the form. If he had authenticated with the website in another browser tab then his cookies might think the request is coming from a safe place and accept this seemingly normal form as if it were normal safe.

b.) CSRF is an architectural issue because it has more to do with how a web application was designed than it does one programmer making a coding mistake. The web application would have to be adjusted to incorporate some sort of CSRF token or double cookie submission which is used application wide, not particular to one spot necessarily.

3.)

a.)

A SQL injection vulnerability is a query that directly injects a users input into a query string. An example is

$queryString = “SELECT * FROM users WHERE id = ‘“ . $User_Id . “‘ AND passwd = ‘“ . $Password . “‘;”;

As you can see the variables $User_Id and $Password are inserted directly into the query string. If these were read in from an input field then this would be vulnerable to SQL injection.

b.)

You fix SQL injection by validating the input and binding the parameters in your query string. For example, you can use regex to essentially white list the acceptable input values for each field. Many languages have frameworks already created that will bind variables to query strings. For example, in php you would use some sort of `prepare` statement and then a `bind_param` function and an `execute` function. The exact syntax varies based on which database management library you are using but I believe with mysqli the syntax is something like

```
$stmt = $mysqli_connection->prepare($queryString);
$stmt->bind_param(“ss”,$User_Id,$Password);
$stmt->execute();
```

The query string also needs to be adjusted to replace any dynamically added variables with questions marks. For example the query above would become.

$queryString = “SELECT * FROM users WHERE uid = ? AND passwd = ?;”;

Using some form of prepare, bind, and execute is the recommended fix. This can also be done if a stored procedure accepts input parameters, as they are also vulnerable to sql injection. The same type of fix would apply.

4.)

One thing that I was a little surprised on was just how little I knew on web application security. While I was playing around with Owasp - BWA Mutillidae it became really obvious how big of a monster web application security can be. It seems the more I know, the more I realize how much I still don’t know. I think I could play around with that site for hours without even really scratching the surface to all the potential vulnerabilities. I also was having a lot of fun which was refreshing as a lot of classes seem more like work than play. This felt more like play. I knew I liked security, but of all the security classes I’ve taken this was by far the most enjoyable.