CIS4930: Software Security Mini-Project 1

Website Overview:

As a team, we decided to build a site where you can both get data based on cities and input data based on cites as well. We are calling it Cityscape for that purpose.

Our attacks on site:

On this site, we will be using four different attack formats as follows: SQL Injection, XSS, IDOR, and Session Management. Below, the executions of each are explained.

SQL injection:

SQL injection is basically a code injection technique used to exploit websites via a data-driven avenue. In most cases, this is used to break or hack into sites that are data heavy or simply uncover a lot of information that could be personal. This is done through the insertion field of a website having SQL code copied and pasted into that field. The particular outcome is reached based on the attacker's hope.

XSS:

This is another attack that includes the insertion of code via a web page, but it is different from SQL because this type of attack is more complex in nature. XSS refers to cross-site-scripting, where an attacker will run what could be malicious javascript in another user's web application or their browser. The javascript written by the attacker is utilized by getting the user to access this separate page by placing or embedding it somewhere in the original web page.

IDOR:

Like other cyber attacking methods, this method allows users to access data that is obviously not meant to be accessed. What is really unique about this one is that this exploit happens when there is a reference within the site that points to an IIO, an internal implementation object. But the problem is that the IIO has no other access level while exposed making this an extremely obvious exploit when it is doable.

(non exploit)

Session Management:

Session management is pretty straight forward. It is basically a way for a website to handle multiple requests to a server. This becomes important when you have multiple server requests from the same user or other site trying to send or retrieve data from the server at the same time.

What is the difference between password hashing and encryption?

Hashing:

- One-way function
- Randomizes text
- Creates unique text each time
- Can't be reversed (back to the OG PW)
- One input - one output
- If you modify input, you modify the hash

Encryption:

- Two-way function
- Can be decrypted
- Origin of the message can be traced
- Limited to what users can see

What is salting and peppering?

Salting:

This is when a value is randomly generated and makes it so when the value is entered into the database it can't possibly be hashed to get the password. This leaves each password only crackable by sheer brute force. The salt is always stored in the database and corresponds to its password hash.

Peppering:

This is when a value that also corresponds to a specific passcode is stored outside of the database, like in the source code, for example. This is so that the value is actually kept secret and also enforces that the passcode table(where all the passwords are stored) is not able to be brute forced.

IDOR:

When using this exploit method, we compared the value from the form with the ones listed in the if-else statements and displayed the corresponding message. We improved this by changing the codes so that they were not easy to guess. This is a classic way to prevent IDOR attacks. In doing so we were able to ensure an attacker couldn't embed information into our site to then use it to exploit our site functionality.

Below is what we used in order to evaluate what the user selected in the dropdown. We converted the insecure version from this:

```
if select == "A":
    message = "Address: 123 Apple Blossem Avenue, City 1, State of Confusion 45678"
elif select == "B":
    message = "Address: 456 Beautiful Butterfly Boulevard, City 1, State of Confusion 78910" ...
```

To this:

```
if select == "QRD":
    message = "Address: 123 Apple Blossem Avenue, City 1, State of Confusion 45678"
elif select == "AHE":
        message = "Address: 456 Beautiful Butterfly Boulevard, City 1, State of Confusion 78910" …
```

Note that the single letter A became an unpredictable code, QRD.

XSS:

The XSS insecurity took in user input and returned a page with the words: "Hello <user input>". The input was used directly without sanitation, which allowed XSS attacks to occur.

return "Hello %s" % enterdata;

We improved this by escaping the input: return "Hello %s" % escape(enterdata);

Session Management:

Attackers can perform a session management exploit by decoding the Base64 and changing it from "user" to "admin" and then encoding the word again. We improved the security by switching from the original Base64 to a random code as a stand in for "user" or "admin." That way, the attacker will not be able to easily switch to admin without guessing millions of random combinations of letters and numbers.

SQL injection:

We originally used c.execute("SELECT * FROM cities WHERE name = '%s'" % projectpath) to query. However, this was not sanitizing the data, so we improved this by changing it to val = (projectpath2,) c.execute('SELECT * FROM cities WHERE name = ?', val) in order to sanitize the data.

For Task 4, please see "Abuse Case Diagram.pdf" and "Use Case Diagram.pdf" on GitHub.