Zach Tsarnas, Cyan Perez, Jaxson Aldrich, Sam Adams

Mini Project Report 1

The overall intention of the web app we decided to make was a search database for cats. This would allow users to search thorugh the database and see their various physcial attributes as well as if they are availble for adoption. Currently the web app deployed is with only vulnerable exploits active, however comments within the code will indcate what was changed in order to protect these vulnerabilties. We as a group found it easier to find vulnerabilites, attack and exploit them, and then proceed to add additions in the code to prevent them.

The login page of our website, in its vulnerable state, allows users to login without means of protecting their credentials. This leaves them susceptible to brute-force and rainbow table attacks. However, to combat these types of attacks, we have implemented salting, peppering, and hashing of passwords. In order to achieve this, we stored unique “salt” values for each user in a database and held them within that database. Then, we hashed these values with the server-kept “pepper” and stored those as well in the database, making sure to keep the pepper away from the database in case it was breached. Finally, once a user attempted to logon to our website, if the hashed result of the entered password with the corresponding salt and pepper values was incorrect, they were not allowed into the Cat Database, but redirected to the login page instead.

In terms of URL manipulation, it is very easy to find the exploits and take advantage of them if one were a hacker. With our deployment on DigitalOcean, all of the different html/php pages are grouped together in the same “droplet” under a common IP address. Thus after entering the IP address into one’s web browser they could enter “/anything” to access any of our available html or php web pages. Thus, one could circumnavigate the login page and get directly to our dashboard. In order to prevent this, we used php sessions. These sessions allow a developer to create a universal variable that is accessible across all web pages. Thus when we confirmed that the login was correct and successful, we created a session variable called “login” and set it to ‘true’. Within the dashboard, an if statement would look to see if the variable was ‘false’ and if it was, it would redirect the user back to the login page. This would prevent any user from accessing the dashboard without first logging in and thus effectively preventing url manipulation.

Cross-Site Scripting is possible for a hacker when code utilizes user-given data without cleaning it up first. Attackers are able to run whatever JavaScript they want on a site that doesn’t protect itself against XSS. With our website, when a user searched a cat’s name using the search box, if there were no results found, the site would print “No results found for: “ and whatever the user had searched. This is a good example of a site that uses user-supplied data without screening it first. In order to fix this vulnerability, all we had to do was not use the variable that the user just defined by typing in the search box. Since we aren’t utilizing that variable in our php code, (other than the sql query) the malicious script isn’t run.

When considering CSRF Session attacks, the biggest concern is escalation of privilege. An unregistered user could gain a registered user’s permissions, or even worse, a registered user could gain an administrator’s permissions. However, a way to prevent this type of attack on a website is to insert logic surrounding sessions. To do this, we were able to implement temporary sessions that guide users while they browse but do not save in the browser, thus

making hackers unable to steal this information. This could be strengthened by creating Session tokens which are randomly generated and encrypted, however for our cat database purposes, realistically, boolean session management will suffice.

For SQL injection, the name of the game is “sanitizing” the inputs that come into your site so that they cannot manipulate the SQL commands to their advantage. Technically, its not really possible to have your website recognize that an attack is occurring, so you need to work around the standard SELECT * FROM “user_input” format. PDO (PHP Data Objects) are a great way to achieve a level of abstraction for your SQL queries so that the inputs will not be able to blindly access anything within a query’s reach. Basically, you prepare an SQL statement without the inputs filled in, instead with placeholder variables or simple ‘?’, next, you’d bind values to the placeholders, then execute the query simultaneously, often aided by an array if you’re searching for more than one output. You’ll notice in the code for CatPage_secure.php under the //PDO Stuff comment, it pulls on a lot of outside functionality to make this work.