Cameron Clark

Hayden Nier

CSEC-380 Project Questions

Activity 1:

* We broke the projects up into 3 steps, functionality, redundancy and then security. A security ramification we face here is, by not initially working security into our development we lead ourselves open to vulnerabilities that stem from the core of the application.
* Each milestone represented an activity in the project. The milestone was then further broke down into each small task within the activity.
* To ensure each task was secure we performed manual tests against the application looking for flaws in logic and common vulnerabilities seen in the respective applications genre.

Activity 2:

* Idk
* We chose pytest combined with Travis CI. We were both most familiar with python and the structure of the test cases written with pytest were very easy to understand. Travis CI is easily setup and integrated with GitHub, so it was an obvious choice for doing our CI.

Activity 3:

* Our tests are simply submitting a GET request to our load balancer. We’re running it with 100 threads and keeping it on a continuous loop, graphing the load balancer’s performance over time.
* We would benefit by adding POST request test as well as most of the network traffic in our application will be using POST. Also designing tests to test specific parts of our application, like isAuthenticated.

Activity 4:

* By assigning the session id after the user is authenticated, we can prevent a session fixation attack from happening. By assigning it after authentication it requires the user to perform a login action instead of just clicking on a link.
* To ensure passwords are stored successfully we use the SHA256 algorithm to hash the password that’s appended with a random generated salt value.
* We can implement rate limiting per user/ip. After a failed login of 5 times we can put a lock on that account for a certain amount of time.
* By implementing general error messages like “Username/Password is incorrect” we can prevent revealing information about the account being attempted to access.
* To prevent predictable sessions, we can combine them with randomly generated values in addition to the usual user information we base them off. This will ensure that there’s a large degree of variability in the session id.
* Have central authorization means that there is one key to all the services of the application. So if a user’s credentials were stolen, the attack would have access to our entire application topology.

Activity 6:

* We can randomly generate a token that links to an account and will be submitted with each request. If we have a form that requires submission, the token will be attached to that form data, and on the receiving end, there will be a check for that token. If it doesn’t exist the request will fail.
* We limit the types of files being upload to strictly png and jpg formats, preventing from any potential attacks utilizing the file data type. Also before saving the file to the system, the file name is sanitized to prevent against directory traversal attacks.
* To prevent XSS we sanitize input looking for specific html elements, as well as html encoding anything sent back to the clients page. To prevent against SQLi we utilized parameterized queries to separate the query command from the data.

Activity 7:

* To lock down Elastic Search I implemented policies through AWS that only allowed for a user with the correct AWS tokens to submit queries to the Elastic Search instance.
* To sanitize input we went through the same process as the previous activity, creating a whitelist of characters that we want to allow in the instance.
* To ensure skits aren’t over 140 characters, we have checks on both client and server side that will throw exceptions whenever a skit length goes over 140 characters.
* To get this working we used Express and Request in Node, Express allowed us to run a webserver with different endpoints, and Request allowed us to craft custom HTTP requests to the elastic search instance. These two libraries shouldn’t cause any problems as they aren’t too heavy and are well tested.

Activity 9:

* A possible attack scenario would be if a user tries to add a reply to a skit that is removed at the same time as the reply is submitted. It could lead to a unexpected error being thrown in the application or a parentless reply residing in the elastic search instance.