7.2D: Session management and CSRF attacks

- I implemented security solutions to address two critical vulnerabilities in a web application: some session management issues and possible CSCF attack threats. That is why the features I have introduced to the current Spring-based online apps are custom exception, cross-site the program I am working on now is protected against scripting attacks.

1) CSRF Attack Prevention

- ✓ Cross Site Request Forgery aka Cross-Site Reference Forgery is a web security flaw where an attacker can perform some activities on behalf of any authorized user without that user ever being knowledgeable about it. I implemented a security check into the code of the Controller class for the Coach to decrease the possibility of CSRF attacks. This check is part of Hitachi's HTTP inspection procedure and involves a check on the Referer portions of Web requests and the elimination of unauthorized requests from admittance to the Web.
- ✓ Referrer Header Comparison: Thus, by using the Referer header of the first request made, I performed the changeAge operation. Next, I extracted the Referer from the second request as Referer2, selected Referers 1 two Referers. I further thought that the request could be a kind of CSRF or attack since the session got cancelled if the data above did not equal something.

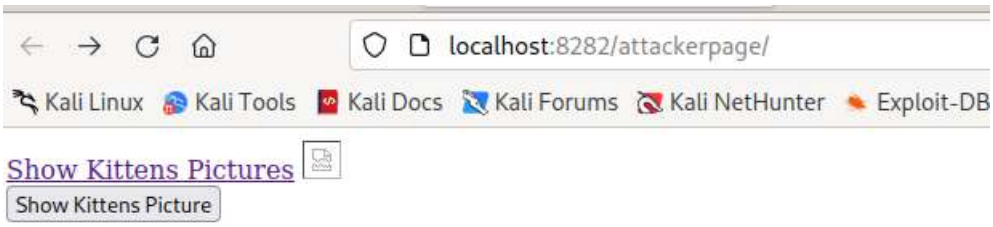
- ✓ This can be mostly prevented by matching the Referer field of a request with the first request that came through the initial page request. This ensure that the request originate from the same website safely and will also prevent the other unauthorized activities that are instigated fake websites.

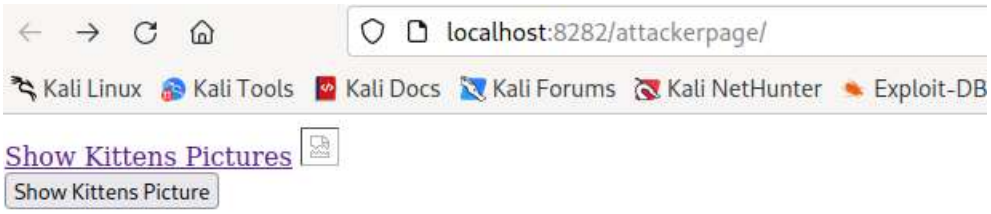
- **When the mitigating strategy will not work**: This security measure, however, will not be effective if the user's browser or the middle proxies modify or strip off the Referer header for personal reasons. When used in isolation this is inadequate because the attacker is capable of spoofing the Referer header and bypass this check. However, the two that are most effective and should be used together include the CSRF tokens and the referrer validation.

2) Improved Error Handling

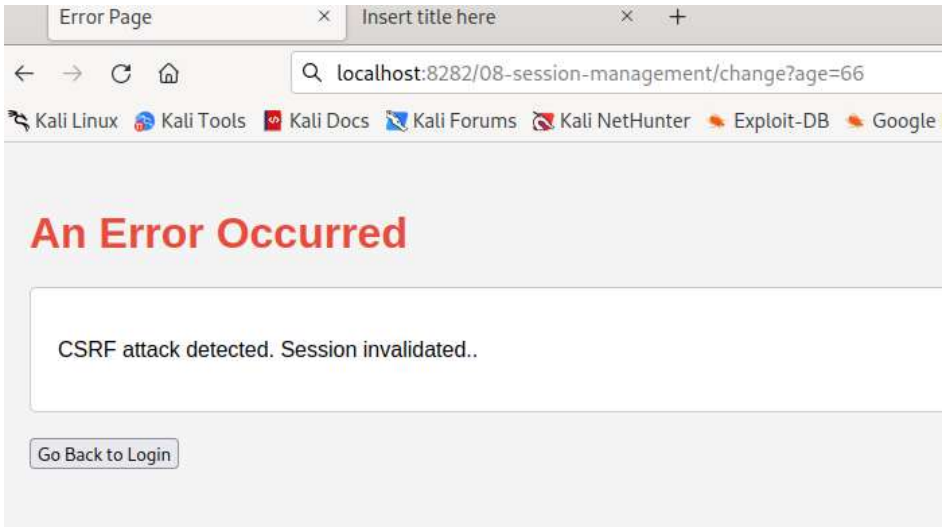
- ✓ I made the web application more robust when I developed a global error handler for it by using the @ExceptionHandler annotation on the surface class of the project known as CoachController.java. This one simply takes it and reports it at the highest level of severity until an exception is caught which in the end is caught.
- ✓ In order to log mistakes at a very high level, I used java.util.Logger. This warrants that critical concerns are recorded as they are later deliberated subsequently within the premises.

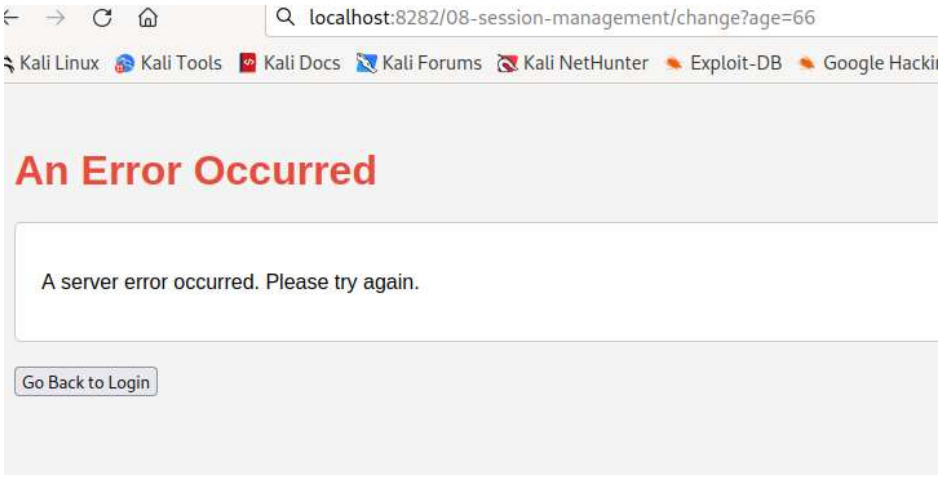
- ✓ What I only provided was the regular personalized error message that prevents the visitor from seeing the new/common 500 page and safeguard against any processed data if the visitor was logged in.





- ✓ A generic error page with the specific error message overlay shows instead of the internal server error 500 when the show kitten picture button is clicked.





Prevention logic

- ✓ Global exception handling system should encompass all the uncaught mistakes to provide visibility and lead to an occurrence of an exception in the SEVERE level while presenting an understandable message during mistaken operations. In this case mistakes that can lead to the revelation of how the program operates are suppressed and from leakage.

- **When the mitigating strategy will not work:** But if the logging is not controlled, or its security is breached, then things with this system can go horribly wrong. Standard logs indicate the occurrence of exception information, which is violation of security, however, good log file encryption and, above all, good encryption and control of access can reduce this risk.

3) Session Management Fix

- ✓ This is based on firsthand experience: this person found a blatant session management vulnerability in the current program; even if, one of the browser windows was closed, an authenticated session can still remain alive in other browsers something that it should not. This is a clear security threat as it will facilitate, for instance, a user logging out of one browser session but remaining logged in to another.
- ✓ To ensure the session of the user is cleared properly after logout, I added a logout form on workout.jsp and defined /logout in LoginController.java.

Form for Logout in Workout.jsp

The logout function within LoginController.java

Prevention Logic

- ✓ The session management ensures when a specific user logs out, then the session is destroyed in all the browsers in use. To protect the data, the system fully logs out the user stripping them of all access privileges to the protected resources by removing all of their sessions from the LoginController
  .

- **When the mitigating strategy will not work:** This tactic however can prove to be quite dangerous if the sessions are handled in the wrong manner; instance; the sessions may be client side cached or the attackers change session tokens in a commonly referred to as session fixation assault. To bolster up this defense, measures like; the use of secure cookies to reduce the risks of session hijacking, while the use of ID regeneration after each login should be established.