|  |  |
| --- | --- |
| Name Of The Student | Jaganathan G |
| Internship Project Topic TCS iON | TCS iON RIO-125: Application of Static Application Security Testing (SAST) Tools – Find Defects in Insecure Web-based Applications |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Uma Devi |
| Name of the Institute | Government College of Engineering, Bodinayakkanur, Theni - 625583 |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 13-05-2023 | DAY 23 | 5 Hours |
| Activities done during the day:   * Scanned an insecure application named “WebGoat-2023”. The scan result contains. * 31 Bugs * 16 Vulnerabilities * 80 Security Hotspot * 568 Code Smells * 1.4% Duplications      * Some Critical issues are found: * Bug. * Use try-with-resources or close this "PreparedStatement" in a "finally" clause. * Change this condition so that it does not always evaluate to "false". * Vulnerabilities * Revoke and change this password, as it is compromised. * Don't use the default "PasswordEncoder" relying on plain-text. * The JWT signature (JWS) should be verified before using this token. * Code Smells * Remove this use of "convert"; it is deprecated. * Define a constant instead of duplicating this literal "classpath:/" 4 times. * Remove this 'public' modifier. * Some Critical Issues:   **File Name:** Make Sure using a PasswordRestLink.java  **Description:** A class which uses a Random in its constructor or in a static main function and nowhere else will be ignored by this rule. The Random () constructor tries to set the seed with a distinct value every time. However, there is no guarantee that the seed will be random or even uniformly distributed. Some JDK will use the current time as seed, which makes the generated numbers not random at all.  **Explanation with Code snippets**  The src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/PasswordResetLink.java    **Risk / Undesirable impact:**   * **Insufficient Randomness**: The Random object created with the default constructor (new Random()) may not provide sufficient randomness. As mentioned earlier, the seed initialization mechanism of Random is not guaranteed to be truly random or uniformly distributed. This can result in predictable sequences of random numbers. If the application relies on strong randomness, such as for cryptographic purposes, using a weak random number generator can lead to vulnerabilities. * **Security Vulnerabilities**: The usage of a week or predictable random number generator, especially in security-sensitive scenarios like user authentication, can introduce security vulnerabilities. In the provided code, if the username is compared in a case-insensitive manner and allows the username "admin" to bypass certain security measures, it can potentially enable unauthorized access or privilege escalation attacks.   **Recommendations:**   * Secure Random Number Generator: Instead of using the Random class, which may not provide sufficient randomness for security-sensitive operations, consider using a more secure random number generator, such as java.security.SecureRandom. This class utilizes cryptographic algorithms to generate secure random numbers.     SecureRandom random = new SecureRandom();   * Improved User Authentication: Instead of relying solely on a case-insensitive username comparison, it is recommended to implement a more robust user authentication mechanism. This typically involves using secure password hashing, implementing multi-factor authentication (MFA), and following secure coding practices for user input validation and authorization checks. Proper user authentication helps prevent unauthorized access and strengthens the security of the application. | | |