|  |  |
| --- | --- |
| Name Of The Student | Jaganathan G |
| Internship Project Topic | 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 |
| 15-05-2023 | DAY 25 | 3 Hours |
| Activities done during the day:     1. 1. Browsed the internet to study about topics. 2. 2. Scanned an insecure application named “WebGoat-2023”. The scan result contains.  * Some Critical issues are found: * Bugs: * "Random" objects should be reused (Security Misconfiguration)   **File Name:** Make Sure this file name for cryptography/EncodingAssignment.java  **Description:** [OWASP Top 10 2017 Category A6](https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration) - Security Misconfiguration  **Explanation with Code snippets:**  src/main/java/org/owasp/webgoat/lessons/cryptography/EncodingAssignment.java    **Risk / Undesirable impact:**  Using pseudorandom number generators (PRNGs) is security-sensitive.  When software generates predictable values in a context requiring unpredictability, it may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.  As the java.util.Random class relies on a pseudorandom number generator, this class and relating java.lang.Math.random() method should not be used for security-critical applications or for protecting sensitive data. In such context, the java.security.SecureRandom class which relies on a cryptographically strong random number generator (RNG) should be used in place.  **Recommendations:**   * Use a cryptographically strong random number generator (RNG) like "java.security.SecureRandom" in place of this PRNG. * Use the generated random values only once. * You should not expose the generated random value. If you have to store it, make sure that the database or file is secure.   **Solution**  SecureRandom random = new SecureRandom(); // Compliant for security-sensitive use cases  byte bytes[] = new byte[20];  random.nextBytes(bytes); Strings and Boxed types should be compared using "equals()". **File Name:** Make Sure this file name for ForgedReviews.java  **Description:**  Comparison of Object References Instead of Object Contents. Use of Wrong Operator in String Comparison. Do not use the equality operators when comparing values of boxed primitives. Do not confuse abstract object equality with reference equality.  **Explanation with Code snippets:**  src/main/java/org/owasp/webgoat/lessons/csrf/ForgedReviews.java    **Risk / Undesirable impact:**   False negatives: The condition may fail to detect a match between referer and "NULL" even when they are equal, leading to incorrect behavior or bypassing necessary security checks.   Unintended behavior: If the referer value is incorrectly treated as not equal to "NULL", it may cause unexpected application behavior or vulnerabilities.  **Recommendations:**  String Comparison: As mentioned in the comment, strings should be compared using the equals() method instead of the != or == operators.  Cross-Site Request Forgery (CSRF) Protection: The provided code is returning a "csrf-same-host" feedback message. This indicates that CSRF protection is being handled by the application.  **Solution:**  String firstName = getFirstName();  String lastName = getLastName();  if (firstName != null && firstName.equals(lastName)) { ... }; A "Map<WebGoatUser, Comments>" cannot contain a "String" in a "WebGoatUser" type **File Name:** Make Sure this file name for CommentsCache.java  **Description:**  The java.util.Collection API has methods that accept Object parameters such as Collection.remove(Object o), and Collection.contains(Object o). When the actual type of the object provided to these methods is not consistent with the type declared on the Collection instantiation, those methods will always return false or null. This is most likely unintended and hides a design problem.  **Explanation with Code snippets:**  src/main/java/org/owasp/webgoat/lessons/xxe/CommentsCache.java    **Risk / Undesirable impact:**  Type Mismatch and Potential Data Corruption   * Data integrity issues: Storing a string value as a key in a map that expects a different type (WebGoatUser) can lead to unexpected behavior, incorrect lookups, or loss of data. * Potential runtime exceptions: If the code attempts to use the mismatched key for map operations, it may throw runtime exceptions, such as ClassCastException or NullPointerException.   **Recommendations:**   * Ensure Consistent Key Types: The code attempts to use a String (retrieved from webSession.getUserName()) as a key in a Map<WebGoatUser, Comments>. To ensure consistency and avoid potential issues, it is recommended to use the appropriate key type (WebGoatUser) consistently throughout the code. * Validate and Sanitize User Input: When retrieving input from webSession.getUserName(), ensure that proper validation and sanitization are performed on the user input. This helps prevent common security vulnerabilities, such as SQL injection or cross-site scripting (XSS) attacks. Validate and sanitize the input before using it as a key in the map.   **Solution:**  WebGoatUser user = webSession.getUser();  var comments = userComments.getOrDefault(user, new Comments());  comments.add(comment);  userComments.put(user, comments);3   1. 3. Visited the digital discussion room and checked for any new announcements. 2. 4. Finally, I wrote the Activity Report | | |