|  |  |
| --- | --- |
| 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 |
| 17-05-2023 | DAY 27 | 3.5 Hours |
| Activities done during the day:   1. Scanned an insecure application named “WebGoat-2023”. The scan result contains.  * Some Critical issues are found: * **Vulnerabilities** * Don't use the default "PasswordEncoder" relying on plain-text.   **File Name:** Make Sure this file name for **WebSecurityConfig.java**  **Description:**  Passwords should not be stored in plain-text or with a fast-hashing algorithm  **Explanation with Code snippets:**  src/main/java/org/owasp/webgoat/webwolf/WebSecurityConfig.java    **Risk / Undesirable impact:**  A cross-site request forgery (CSRF) attack occurs when a trusted user of a web application can be forced, by an attacker, to perform sensitive actions that he didn’t intend, such as updating his profile or sending a message, more generally anything that can change the state of the application.  The attacker can trick the user/victim to click on a link, corresponding to the privileged action, or to visit a malicious web site that embeds a hidden web request and as web browsers automatically include cookies, the actions can be authenticated and sensitive.  **Recommendations:**   * Protection against CSRF attacks is strongly recommended: * to be activated by default for all [unsafe HTTP methods](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Safe_methods). * implemented, for example, with an unguessable CSRF token * Of course all sensitive operations should not be performed with [safe HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Safe_methods) methods like GET which are designed to be used only for information retrieval..   **Solution**:  @EnableWebSecurity  public class WebSecurityConfig extends WebSecurityConfigurerAdapter {  @Override  protected void configure(HttpSecurity http) throws Exception {  // http.csrf().disable(); // Compliant  }  } Replace this persistent entity with a simple POJO or DTO object **File Name:** Make Sure this file name for MailboxController.java  **Description:**  Persistent entities should not be used as arguments of "@RequestMapping" methods  **Explanation with Code snippets:**  src/main/java/org/owasp/webgoat/webwolf/mailbox/MailboxController.java | | |
| **Risk / Undesirable impact:**   1. Object Persistence Vulnerabilities: The code receives an Email object through the request body and directly uses it in the method parameter. If the Email object is directly mapped to a persistent entity and stored in a database without proper validation and sanitization, it can introduce various security vulnerabilities. These include SQL injection, object-relational mapping (ORM) vulnerabilities, or other persistence-related attacks. 2. Data Exposure and Injection: If the Email object is not properly validated or sanitized before processing, it may expose sensitive information or introduce injection vulnerabilities. Attackers could potentially inject malicious content or manipulate data in unexpected ways, leading to security breaches or undesired behavior.   **Recommendations:**   1. Use a Data Transfer Object (DTO) or Value Object: Instead of using the Email object directly as a persistent entity, consider introducing a Data Transfer Object (DTO) or a Value Object to separate the representation of data received via the request from the underlying persistence mechanism. The DTO or Value Object should contain the necessary fields for the email, which can then be validated and sanitized before further processing. 2. Implement Proper Input Validation and Sanitization: Ensure that the data received in the Email object is properly validated and sanitized. Implement appropriate input validation mechanisms to prevent injection attacks and enforce data integrity. Validate fields such as email addresses, subject lines, and message content to avoid potential security vulnerabilities.   **Solution**:  public ResponseEntity<?> sendEmail(@RequestBody EmailDTO emailDto) {  // Method implementation  // Access emailDto.getRecipient(), emailDto.getSubject(), emailDto.getMessage() to retrieve the email data  // Perform the necessary operations to send the email  // Return the appropriate ResponseEntity  } | | |