SCENARIO

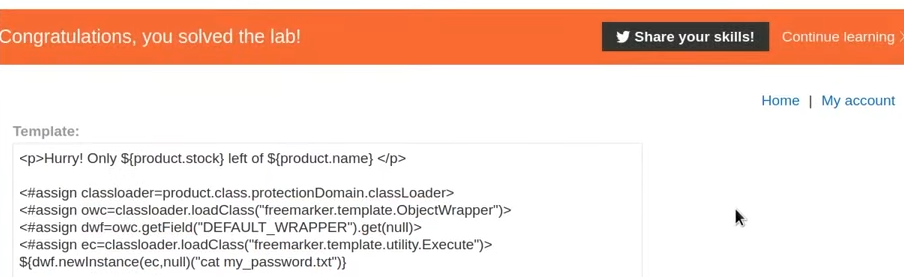
The application is vulnerable to server-side template injection due to the way it unsafely uses a template and we know that it uses the Freemarker template engine due to its poorly implemented sandbox. We will try to figure out a way to know the template used in order to execute arbitrary code on the backend server.

**PROCEDURE**

1. Open the application and log in using the provided credentials in order to act as the target.
2. Now navigate to any blog and click on the **Edit Template** button at the end of the page.
3. Load the JavaDoc for the Object class to find methods that should be available on all objects
4. After studying the Freemaker template documentation, we got to know that by injecting the Payload we can see that the application allows us access to a class with a static method that lets you read a file.
5. There we can notice that we have access to the **settings** object which contains some confidential information.
6. Now that we’ve got the output that contains the contents of the file as decimal ASCII code points.
7. Convert the ASCII back to characters and submit the solution in order to solve the lab.

**PAYLOAD**

${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlos/my\_password.txt').toURL().openStream().readAllBytes()?join(" ")}

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Avoid User-Controlled Templates:** The most effective way to prevent server-side template injection is to avoid letting users supply templates that are subsequently processed by the server.
2. **Sandboxing:** If user-supplied templates are an essential feature of the application, use a template engine that supports sandboxing. Ensure that the sandbox is correctly configured to restrict functionalities that can be abused.
3. **Strictly Define Allowed Methods/Functions:** If the template engine provides a way to define which functions or methods can be used in the templates, restrict them to the bare minimum, and do not allow methods that can execute arbitrary code or perform potentially unsafe operations.
4. **User Input Validation:** Ensure that all user inputs are strictly validated. Use allow-lists rather than deny-lists to ensure that only expected and safe input can be processed.
5. **Disable Debug Information:** Always run your applications in production mode, ensuring that verbose error messages that could provide useful information to an attacker are disabled.