SCENARIO

The application is vulnerable to server-side template injection due to the way it unsafely uses a template. 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. Inject Payload 1 within the **{}** which will throw an exception revealing the name of the template engine deployed at the backend.
4. After studying the Django documentation, we got to know that by injecting Payload 2 we can see the application’s debug information.
5. There we can notice that we have access to the **settings** object which contains some confidential information.
6. So, we will try to inject the Payload 3 into the field in order to do Remote Code Execution by going through the documentation of Django template engine.

**PAYLOAD**

1. {{2\*2}}
2. {% debug %}
3. {{settings.SECRET\_KEY}}

**REMEDIATION**

1. **Avoid Dynamic Template Generation:** Avoid the creation of templates that include user-controlled data. Ideally, templates should be static, and any dynamic data should be inserted using safe mechanisms such as context variables.
2. **Sandboxed Execution:** If your template engine supports it, consider executing templates in a sandboxed environment. This restricts the available operations, limiting potential damage.
3. **Disable Debug Mode:** Always run applications in production mode, not in debug mode. Debug modes often provide verbose error messages that can disclose sensitive information, like in this Django scenario.
4. **Limit Template Features:** Some template engines allow you to limit the features that are available for templates. By restricting these features, you can minimize the impact of an injection attack.
5. **Context Variables Only:** When working with Django templates specifically, always use context variables to insert dynamic content, and avoid the direct use of Django's ORM within templates.
6. **Validate, Filter, and Sanitize Input:** All user input should be treated as untrusted. It should be strictly validated, filtered, and sanitized to ensure that it doesn't contain malicious data.