SCENARIO

The application contains a reflected cross site scripting vulnerability in the URL as it reflects our input in JavaScript URL but the application is blocking some characters in order to prevent any attack. We will try to trigger an alert message by injecting a payload into the application.

**PROCEDURE**

1. Go to the vulnerable application and navigate to any blog.
2. Observe in the source as well as in the URL that the URL is encoded into a JavaScript URL.
3. We will try to use exception handling to call the alert function with arguments and the throw statement is used, separated with a blank comment in order to get round the no spaces restriction. The alert function is assigned to the onerror exception handler.
4. As throw is a statement, it cannot be used as an expression. Instead, we need to use arrow functions to create a block so that the throw statement can be used. We then need to call this function, so we assign it to the toString property of window and trigger this by forcing a string conversion on window.

**PAYLOAD**

/post?postId=5&'},x=x=>{throw/\*\*/onerror=alert,1337},toString=x,window '',{x:'

**REMEDIATION**

1. **Input Validation:** This is the first line of defense against XSS attacks. Implement strict validation on all inputs. Use regular expressions to ensure the input adheres to the expected format.
2. **Output Encoding:** Whenever user-controlled input is reflected back to the user, it should be properly encoded. For example, HTML entities should be encoded before being rendered back to ensure they are treated as data and not executable code.
3. **Avoid eval():** The eval() function in JavaScript is notorious for its potential security issues. Avoid using it, and similarly avoid functions like setTimeout, setInterval, and Function() which can execute strings as code.
4. **URL Encode Data:** If data is being passed via URLs, ensure that it's URL encoded. This helps in protecting the application from malicious payloads being delivered via the URL.