SCENARIO

The application possesses a NoSQL Injection Vulnerability and is powered by a MongoDB NoSQL database. We will try to exploit the vulnerability in order to gain administrator privileges using MongoDB operators.

**PROCEDURE**

1. Open the web application and navigate to the user login page.
2. Login using the given credentials.
3. Try the following testcases:
   * Change the value of the username parameter from "wiener" to {"$ne":""}, then send the request.
   * Change the value of the username parameter from {"$ne":""} to {"$regex":"wien.\*"}, then send the request.
   * With the username parameter set to {"$ne":""}, change the value of the password parameter from "peter" to {"$ne":""}, then send the request again.
4. Based on the responses of the above requests we will craft a Payload and will try to inject it into the username and password field as mentioned in Payload 1 and Payload 2.
5. Then, right click on the response and select Show in browser and we’ve got access to the administrator panel.

**PAYLOAD**

# {"$regex":"admin.\*"}

1. {"$ne":""}

**REMEDIATION**

1. **Avoid Dynamic Query Construction with User Input:** One of the fundamental causes of NoSQL injections is constructing dynamic queries using unsanitized user input. If possible, avoid using dynamic queries. If you must, ensure that user input is properly sanitized and validated.
2. **Implement Input Validation and Sanitization:** Ensure all user inputs are validated against a set pattern, type, or value range. Sanitize them to make sure they don't contain any characters or patterns that might be interpreted as code or operators.
3. **Bind Variables:** Just like with SQL, use parameterized queries or API functions that allow for variable binding. This ensures the input is treated strictly as data and not executable code.
4. **Limit Database Permissions:** The account that the web application uses to connect to the database should have minimal necessary permissions. Don't use an account that has write or administrative access for standard application queries.
5. **Avoid Exposing Detailed Error Messages:** Detailed error messages can provide attackers with clues about the database structure and potential vulnerabilities. Instead, provide generic error messages to the user and log the details for internal review.
6. **Database Version:** Always ensure you are using the latest stable version of your NoSQL database. New versions often contain important security patches.
7. **Implement a Web Application Firewall (WAF):** A WAF can help detect and block NoSQL injection attacks. Set up rules that detect and block requests containing NoSQL operators or patterns commonly associated with injection attacks.