Discussion 6: Data Sensitivity In Java

Discussion Topic:

Data sensitivity can have a detrimental effect on the performance of a Java application.

- What are some factors that affect the sensitivity of data in a Java application?
- What might be an example of an application that contains sensitive data in a noncompliant manner?
- Be sure to provide an appropriate source code example to illustrate your points.

My Post:

Hello class,

Sensitive data is information that individuals or organizations want to protect from public exposure as if its unintentional release or stolen could result in harm to the person or the organization in the form, for example, of identity theft or other criminal intentions (Baig, 2021). For individuals, this may include personal details like payment information or birth dates, and for organizations, it could be proprietary corporate information.

Java, as a programming language, incorporates several abstractions to secure sensitive data. However, data security can still be compromised, in an application, by different factors such as improper handling of sensitive information and vulnerabilities to data injection attacks, as well as insufficient input validation and the unsafe handling of mutable objects.

Oracle (n.d.), the corporation that owns the rights to Java, provides coding guidelines for Java SE, The following is a list of these guidelines.

Guideline 2 Confidential Information (Oracle, n.d.).

- Guideline 2-1 / CONFIDENTIAL-1: Purge sensitive information from exceptions
  Sensitive information in exceptions should not reveal internal states or paths.
- Guideline 2-2 / CONFIDENTIAL-2: Do not log highly sensitive information
  Logs should exclude sensitive details like passwords or security tokens.
- Guideline 2-3 / CONFIDENTIAL-3: Consider purging highly sensitive information from memory after use
  Clearing sensitive data from memory reduces its exposure window.

If sensitive information is logged or stored insecurely, it becomes vulnerable to unauthorized access.

Code examples:

Unsafe code, an application that logs sensitive user passwords in clear text violates the principle of purging sensitive information from logs.

```
public class PasswordLogger {
```

```
    public void logPassword(String password) {
        // Logs sensitive data—violates secure coding guidelines
        System.out.println("Password: " + password);
    }
}
```

Safe code, to comply with secure coding guidelines, sensitive data should be sanitized or excluded from logs entirely.

```
public class SecurePasswordLogger {
    public void logPassword() {
        System.out.println("Password logging is not permitted.");
    }
}
```

Guideline 3 Injection and Inclusion (Oracle, n.d.).

- Guideline 3-1 / INJECT-1: Generate valid formatting
  Input should always be sanitized to prevent incorrect formatting issues.
- Guideline 3-2 / INJECT-2: Avoid dynamic SQL
  Always use parameterized SQL statement queries to eliminate SQL injection risks.

These vulnerabilities may allow attackers to manipulate queries and access, modify, or delete sensitive data.

Code examples:

Unsafe code, using dynamic SQL queries to process user inputs without sanitization is a common mistake.

```
String query = "SELECT * FROM users WHERE username = '" + username + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Safe code, instead, parameterized queries should be used to prevent injection attacks:

```
String query = "SELECT * FROM users WHERE username = ?";
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, username);
ResultSet rs = pstmt.executeQuery();
```

Guideline 5 Input Validation (Oracle, n.d.).

- Guideline 5-1 / INPUT-1: Validate inputs
  Input from untrusted sources should be sanitized and validated.
- Guideline 5-2 / INPUT-2: Validate output from untrusted objects as input
  Output from untrusted sources should be revalidated before further processing.

These vulnerabilities may allow attackers may exploit improperly validated inputs to execute malicious code or access restricted data.

Code example:

Safe code, proper input validation ensures that malicious code is not injected.

```
public void validateInput(String userInput) {
    if (userInput == null || userInput.isEmpty() || userInput.contains("..")) {
        throw new IllegalArgumentException("Invalid input detected.");
    }
    System.out.println("Validated input: " + userInput);
}
```

Guideline 6 Mutability (Oracle, n.d.).

- Guideline 6-1 / MUTABLE-1: Prefer immutability for value types
  Immutable objects avoid unintended modifications in shared contexts.
- Guideline 6-2 / MUTABLE-2: Create copies of mutable output values
  Return copies of mutable objects to ensure encapsulation.

These vulgarities can lead to inconsistent object states or security vulnerabilities, especially when mutable objects expose sensitive information.

Code example:

Safe code, creating immutable objects or making safe copies reduces risks associated with mutable objects.

```
public class ImmutableExample {
    private final List<String> items;

    public ImmutableExample(List<String> items) {
        this.items = new ArrayList<>(items); // Creates a safe copy
    }

    public List<String> getItems() {
        return Collections.unmodifiableList(items); // Returns an immutable view
    }
}
```

To summarize, sensitive data is information that individuals or organizations want to protect from public exposure as if it is exposed could result in harm to the person or the organization. Factors such as improper handling of sensitive information, vulnerabilities to data injection attacks, the unsafe handling of mutable objects, and insufficient input validation can compromise an application's integrity. However, by adhering to secure coding guidelines such as avoiding the logging of sensitive information, using SQL parameterized queries to prevent injection attacks, validating all inputs, and handling mutable objects correctly, developers can build Java applications that are secure and keep sensitive data protected.

-Alex

References:

Baig, A. (2021, May 17). *What is sensitive data?* Securiti. https://securiti.ai/blog/what-is-sensitive-data/

Oracle (n.d.). Secure coding guidelines for Java SE. Updated May 2023. Oracle. https://www.oracle.com/java/technologies/javase/seccodeguide.html