10 Java Security Best Practices

Weihao Gao, Yangkai Lin, Zicong Lin,
Andrew White, Weikai Liang

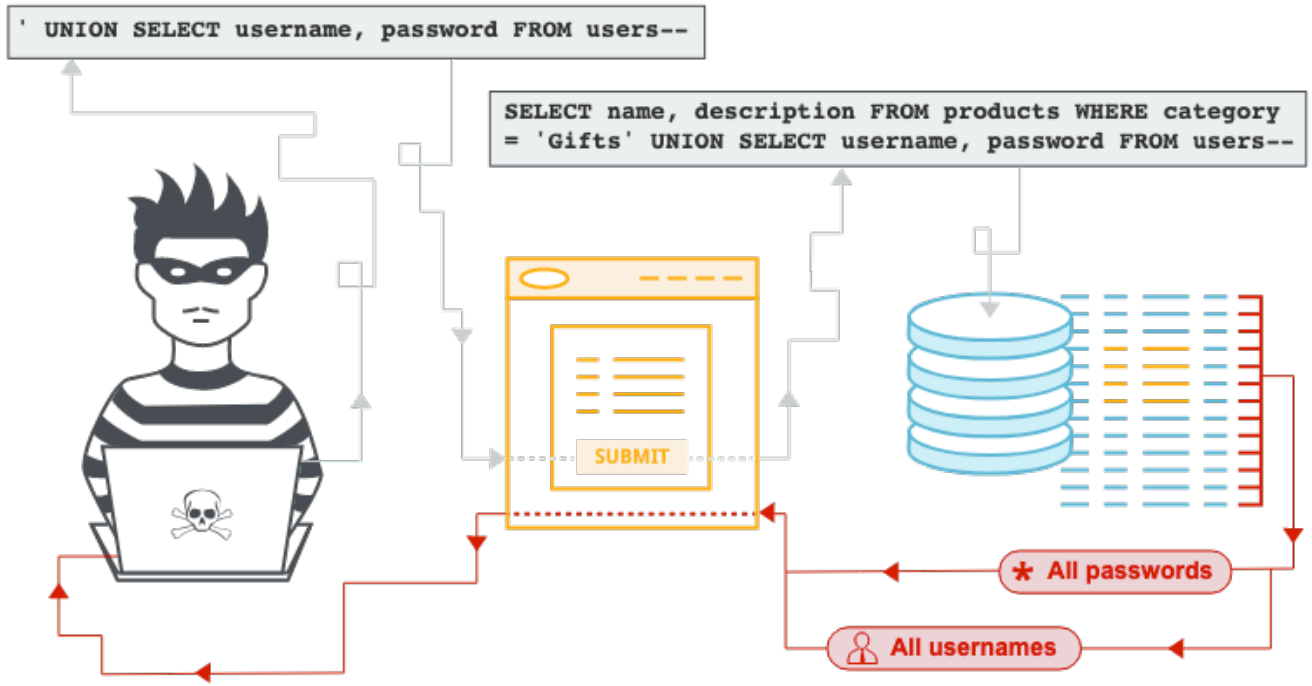
1. SQL Injection

- SQL stands for Structured Query Language. SQL is used to communicate with a database.
- SQL injection is a code injection technique that can destroy the database by executing malicious SQL statements.

1. SQL Injection

```
// The query can be hijacked by malicious input.
public void selectExample(String parameter) throws SQLException {
   Connection connection = DriverManager.getConnection(DB_URL, USER, PASS);
   String query = "SELECT * FROM USERS WHERE lastname = " + parameter;
   Statement statement = connection.createStatement();
   ResultSet result = statement.executeQuery(query);
   printResult(result);
}
```

1. SQL Injection



https://portswigger.net/web-security/images/sql-injection.svg

1. SQL Injection

Source: https://hackaday.com/wp-content/uploads/2014/04/18mpenleoksq8jpg.jpg

1. SQL Injection

```
// This technique prevents the parameter input from interfering with the SQL code.

public void prepStatmentExample(String parameter) throws SQLException {

   Connection connection = DriverManager.getConnection(DB_URL, USER, PASS);

   String query = "SELECT * FROM USERS WHERE lastname = ?";

   PreparedStatement statement = connection.prepareStatement(query);

   statement.setString(1, parameter);

   System.out.println(statement);

   ResultSet result = statement.executeQuery();

   printResult(result);

}
```

2. Use OpenID Connect with 2FA

- Leverage third-party clients for authentication.
- Identity management and access control is difficult and broken authentication is often the reason for data breaches.

3. Scan your dependencies for known vulnerabilities

Ensure your application does not use dependencies with known vulnerabilities. Use a tool like Snyk (www.Snyk.io) to:

- Test your app dependencies for known vulnerabilities
- Automatically fix any existing issues
- Continuously monitor your projects for new vulnerabilities over time

4. Handle sensitive data with care

- Sanitize the toString() methods of your domain entities.
- If using Lombok, annotate sensitive classes. @ToString.Exclude
- Use @JsonIgnore and @JsonIgnoreProperties to prevent sensitive properties from being serialized or deserialized.

```java
@ToString
public class LombokToStringDemo4 {

  private Long id;

  private static boolean defaultStatus = false;

  @ToString.Exclude
  private String name;

  @ToString.Exclude
  private LocalDate dateOfBirth;

}
```

https://javabydeveloper.com/lombok-tostring-examples/

5.Sanitize all input

Cross-site scripting(XSS) attack

XSS is an injection of JavaScript code executed remotely

OWASP Java encoding

```
<dependency>
        <groupId>org.owasp.encoder</groupId>
        <artifactId>encoder</artifactId>
        <version>1.2.2</version>
</dependency>
```

6.Configure your XML-parsers to prevent XXE

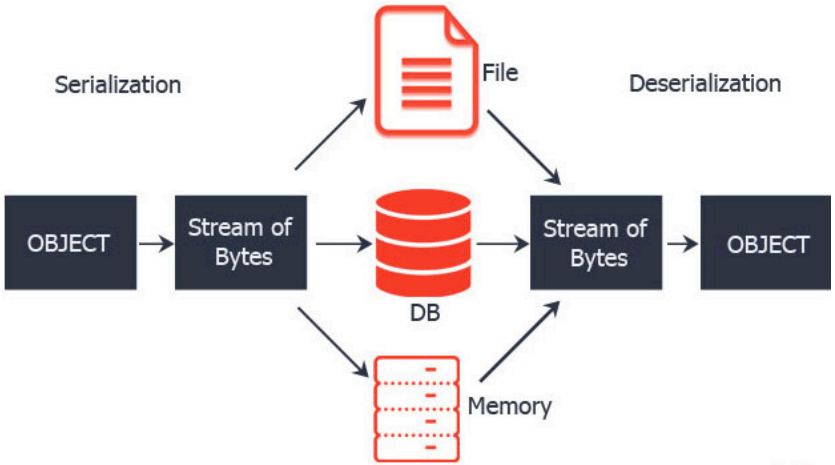
When XML external entities (XXE) are enabled, they can be exploited to create malicious XML.

The Java XML library is particularly vulnerable to XXE injection because most XML parsers enable external entities by default.

```
...

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

factory.setFeature("https://xml.org/sax/features/external-general-entities", false);

saxParser.getXMLReader().setFeature("https://xml.org/sax/features/external-general-entities", false);

factory.setFeature("https://apache.org/xml/features/disallow-doctype-decl", true);

...
```

7. Avoid Java serialization

Java Serialization and Deserialization



Source: https://byam.github.io/glossary,/data/2018/06/11/data-engineering-glossary.html

Insecure Deserialization

Insert **malicious** serialized object that can:

- Manipulate object data
- Manipulate program logic

8. Use strong encryption and hashing algorithms

- Symmetric encryption: AES using Google Tink

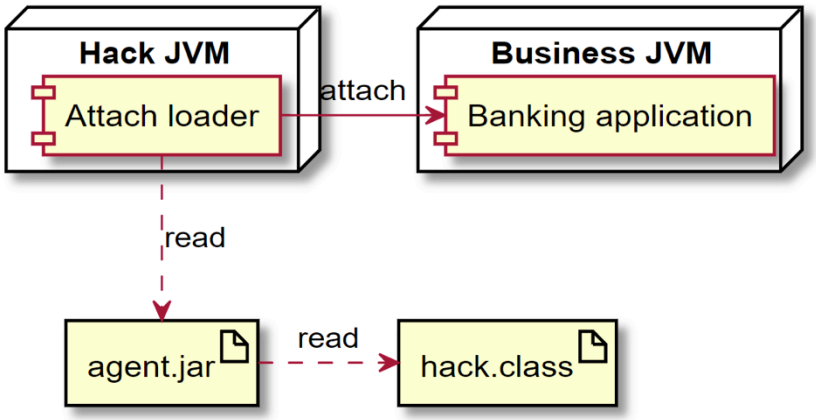
```
AeadConfig.register();
KeysetHandle keysetHandle = KeysetHandle.generateNew(AeadKeyTemplates.AES256_GCM);

String plaintext = "I want to break free!";
String aad = "Queen";

Aead aead = keysetHandle.getPrimitive(Aead.class);
byte[] ciphertext = aead.encrypt(plaintext.getBytes(), aad.getBytes());
String encr = Base64.getEncoder().encodeToString(ciphertext);

byte[] decrypted = aead.decrypt(Base64.getDecoder().decode(encr), aad.getBytes());
String decr = new String(decrypted);
```

- Asymmetric encryption: BCrypt using Spring Security

Source: https://snyk.io/blog/10-java-security-best-practices/

9. Enable the Java Security Manager

Attach API Hack:



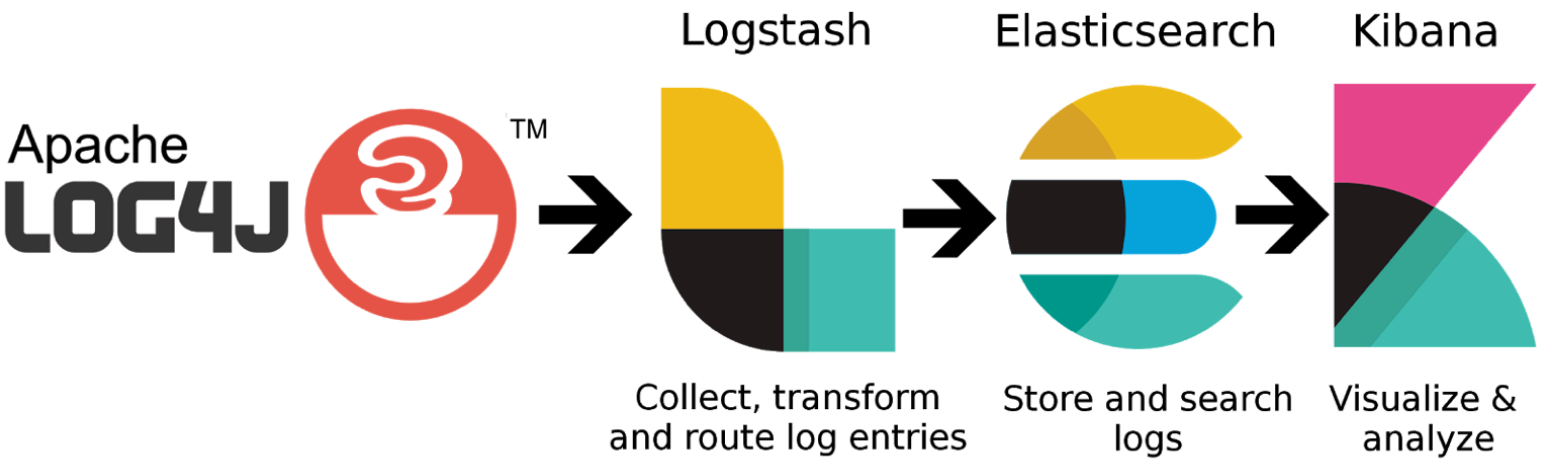
Souce: https://blog.frankel.ch/jvm-security/4/

Enable default: java -Djava.security.manager

Enable custom: java -Djava.security.manager -Djava.security.policy==/foo/bar/custom.policy

10. Centralize logging and monitoring

- Log all incoming requests
- Monitor for strange activity
  - CPU spikes, large load from a single IP



Souce: https://www.shortn0tes.com/2018/02/elk-stack-with-log4j.html

Discussion

Q: Do you think these practices are useful and how will you use them in your application development?