Sécurité du code, une mission du développeur

Programmation avancée – Java

F5 – ISIMA 2020/2021

Olivier Goutet
o.goutet@openium.fr

01 décembre 2020

Introduction

- Les livrables JAVA ne sont pas ce qu'on peut appeler sûrs
- Il y a beaucoup de "failles" que vous pouvez laisser si vous n'y faites pas attention
- Un nombre important de règles à appliquer existent
  - OWAP
  - Différents livres évoquent le sujet : Effective Java en fait partie
  - Oracle diffuse un listing complet ici
- La plus part des règles s'appliquent à des serveurs Java
- Mais les principes de basent restent commun à tout programme JAVA

Plan

Règles de sécurité à appliquer lors du développement

Principe 0-2 / FUNDAMENTALS-2 : Avoid duplication

- Pleins de problèmes peuvent arriver avec du code dupliqué
- Traitement des données pas identiques
- Corrections pas appliquées sur toutes les copies

Principe 0-6 / FUNDAMENTALS-6 : Encapsulate

- N'exposer que ce qui est nécessaire
- Les attributs doivent être privés et les accésseurs évités !
- "The interface of a method, class, package, and module should form a coherent set of behaviors, and no more."

Principe 0-8 / FUNDAMENTALS-8 : Secure third-party code

- Se maintenir à jour sur les bibliothèques que l'on utilise
- Suivre les mises à jour de sécurité
- Outils de vérifications de bibliothèques existent

Plan

Règles de sécurité à appliquer lors du développement

Principe 1-1 / DOS-1 : Beware of activities that may use disproportionate resources

- Paramètres d'un programme hors limites
- Zip bombs
- Désérialisation d'un objet énorme

Principe 1-2 / DOS-2 : Release resources in all cases

- Toujours fermer les ressources qu'on ouvre
- Fichiers, Streams, Bases...
- S'assurer qu'on flush bien les streams en écriture

```
1  public void bufferedWriteGzipFile(OutputStreamHandler handler) throws IOException {
2      try (
3          final OutputStream rawOut = Files.newOutputStream(path);
4          final OutputStream compressedOut =  new GzipOutputStream(rawOut);
5      ) {
6          final BufferedOutputStream out = new BufferedOutputStream(compressedOut);
7          handler.handle(out);
8          out.flush();
9      }
10 }
11 }
```

Principe 1-3 / DOS-3 : Resource limit checks should not suffer from integer overflow

- Vérifier les paramètres d'une méthode
- On pourrait avoir un paramètre beaucoup trop grand pour les usages normaux de la méthode
- Création d'étudiants par exemple

```
1     private void generateStudiant(long count) {
2           if (count < 0 || count > max) {
3                       throw new IllegalArgumentException();
4           }
5     }
```

Plan

Règles de sécurité à appliquer lors du développement

Principe 2-1 / CONFIDENTIAL-1 : Purge sensitive information from exceptions

- Une exception donne des informations de debug, mais aussi des informations d'attaque
  - Valeur attendue
  - Fichier (révèle l'arboressence)
  - ...
- Valide pour les serveurs, mais aussi pour vos applications

Principe 2-2 / CONFIDENTIAL-2 : Do not log highly sensitive information

- On ne log pas des données utilisateur
- No Exceptions
- Valide pour les serveurs, mais aussi pour vos applications

Plan

Règles de sécurité à appliquer lors du développement

Principe 3-1 / INJECT-1 : Generate valid formatting

- Lorsque votre programme lit des données venant d'un utilisateur, vous devez être robuste à des données invalides
- Plantages, injections...
- Valide pour les serveurs, mais aussi pour vos applications

Principe 3-2 / INJECT-2 : Avoid dynamic SQL

- Injections SQL
- Toujours préférer les `PreparedStatement` et `CallableStatement` aux `Statement`
- Préférer les bibliothèques haut niveau plutôt qu'une implémentation "root"

```
1 String sql = "SELECT * FROM User WHERE userId = ?";
2 PreparedStatement stmt = con.prepareStatement(sql);
3 stmt.setString(1, userId);
4 ResultSet rs = prepStmt.executeQuery();
```

Principe 3-9 / INJECT-9 : Prevent injection of exceptional floating point values

- Attention à l'utilisation de flottants
- Plusieurs valeurs sont invalides NaN, Infini
- Code sanitization, defencing programming

```
1 if (Double.isNaN(untrusted_double_value)) {
2     // specific action for non-number case
3 }
4
5 if (Double.isInfinite(untrusted_double_value)){
6     // specific action for infinite case
7 }
8
9 // normal processing starts here
```

Plan

Règles de sécurité à appliquer lors du développement

0 Fundamentals
1 Denial of Service
2 Confidential Information
3 Injection and Inclusion
4 Accessibility and Extensibility
5 Input Validation
6 Mutability

Principe 4-1 / EXTEND-1 : Limit the accessibility of classes, interfaces, methods, and fields

- Utiliser les packages
- Publique si vous devez exposer la classe ou l'interface
- private package ou privé sinon
- Attention par défaut les méthodes d'une interface sont publiques

Principe 4-5 / EXTEND-5 : Limit the extensibility of classes and methods

- Si votre classe ou méthode ne doit pas être héritée, mettez la final

```
1          // Unsubclassable class with composed behavior.
2  public final class SensitiveClass {
3
4      private final Behavior behavior;
5
6      // Hide constructor.
7      private SensitiveClass(Behavior behavior) {
8          this.behavior = behavior;
9      }
10
11     // Guarded construction.
12     public static SensitiveClass newSensitiveClass(
13         Behavior behavior
14     ) {
15         // ... validate any arguments ...
16
17         // ... perform security checks ...
18
19         return new SensitiveClass(behavior);
20     }
21 }
```

Plan

Règles de sécurité à appliquer lors du développement

Principe 5-1 / INPUT-1 : Validate inputs

- Vérifiez les paramètres de vos méthodes (defensive programming)
- Vérifiez tous les paramètres de votre main
- Publique si vous devez exposer la classe ou l'interface
- private package ou privé sinon
- Attention par défaut les méthodes d'une interface sont publiques

Principe 6-9 / MUTABLE-9 : Make public static fields final

- Mettez toujours vos constantes final
- Aucune modification possible

```
1 public class Files {
2     public static final String separator = "/";
3     public static final String pathSeparator = ":";
4 }
```

Principe 6-10 / MUTABLE-10 : Ensure public static final field values are constants

- Toutes données qui devrait être non modifiable doit l'être
- Aucune modification possible

```
1 import static java.util.Arrays.asList;
2 import static java.util.Collections.unmodifiableList;
3 ...
4 public static final List<String> names = unmodifiableList(asList(
5     "Fred", "Jim", "Sheila"
6 ));
```

Conclusion

- De nombreuses règles existent, à appliquer en fonction de la sensibilité de votre programme
- Elles doivent être connues pour savoir ce que vous faites lorsque vous choisissez telle ou telle implémentation
- Vous ne pourrez pas toutes les appliquer, mais être sensibilisé fera de vous un meilleur défenseur, ou attaquant ;-)

Références

- https ://www.oracle.com/java/technologies/javase/seccodeguide.html
- Effective Java Second Edition (Joshua Blosh)