INRIA

Sécurité de la plate-forme d'exécution Java : limites et proposition d'améliorations

Guillaume Hiet, Frédéric Guihéry, Goulven Guiheux,
David Pichardie, Christian Brunette

AMOSSYS-INRIA-SILICOM

10 juin 2010

Introduction

Java : une réponse à un besoin de sécurité ?

- Renforcer la sécurité au niveau applicatif
- Un langage et une plate-forme d'exécution « sécurisés »

En pratique

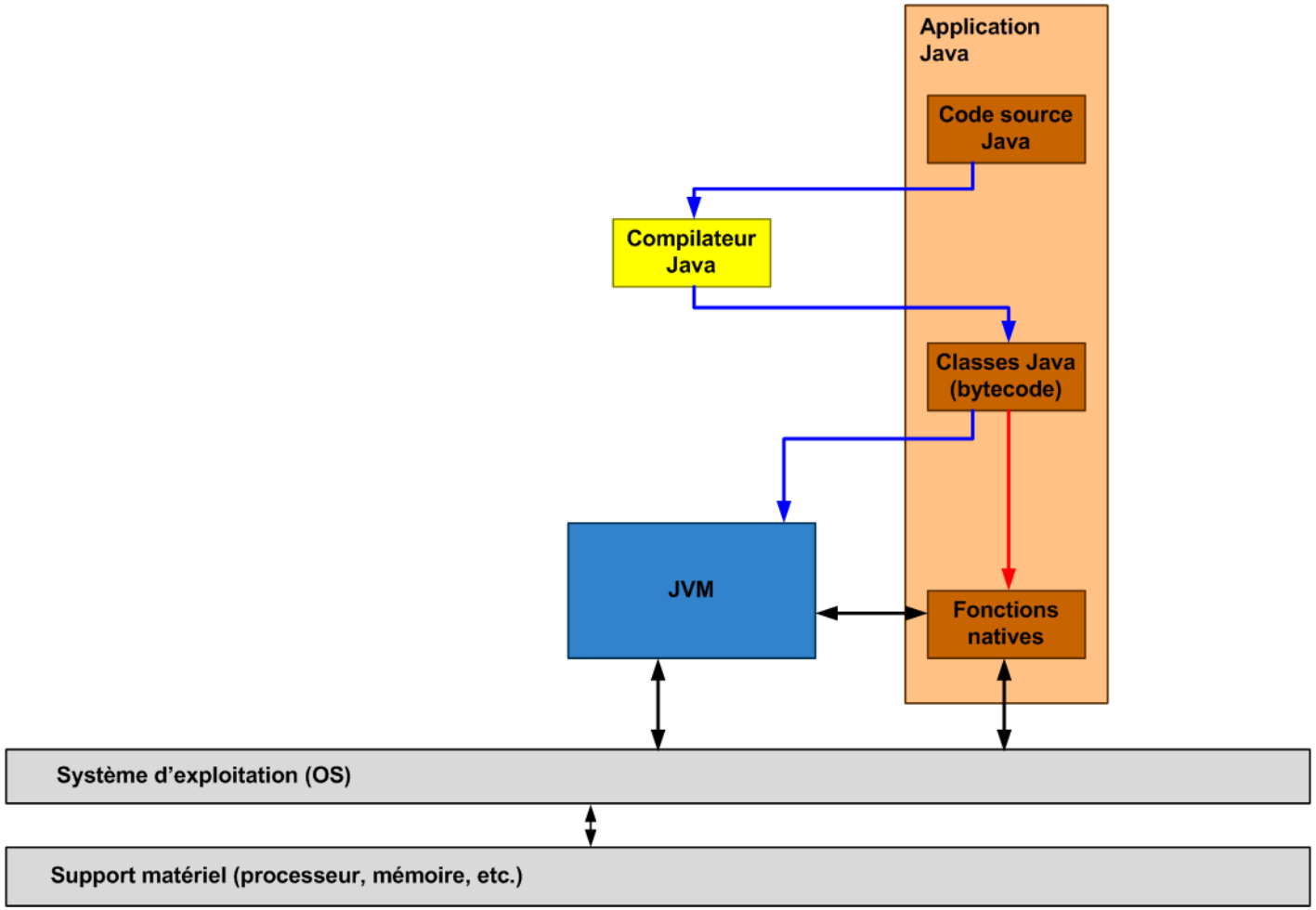
- Une implémentation de plus en plus complexe
- Des vulnérabilités récurrentes (cf les différents CVE)
- Un mécanisme de contrôle d'accès peu utilisé
- Une intégration pas toujours aisée avec les mécanismes de l'OS
- etc.

JAVASEC

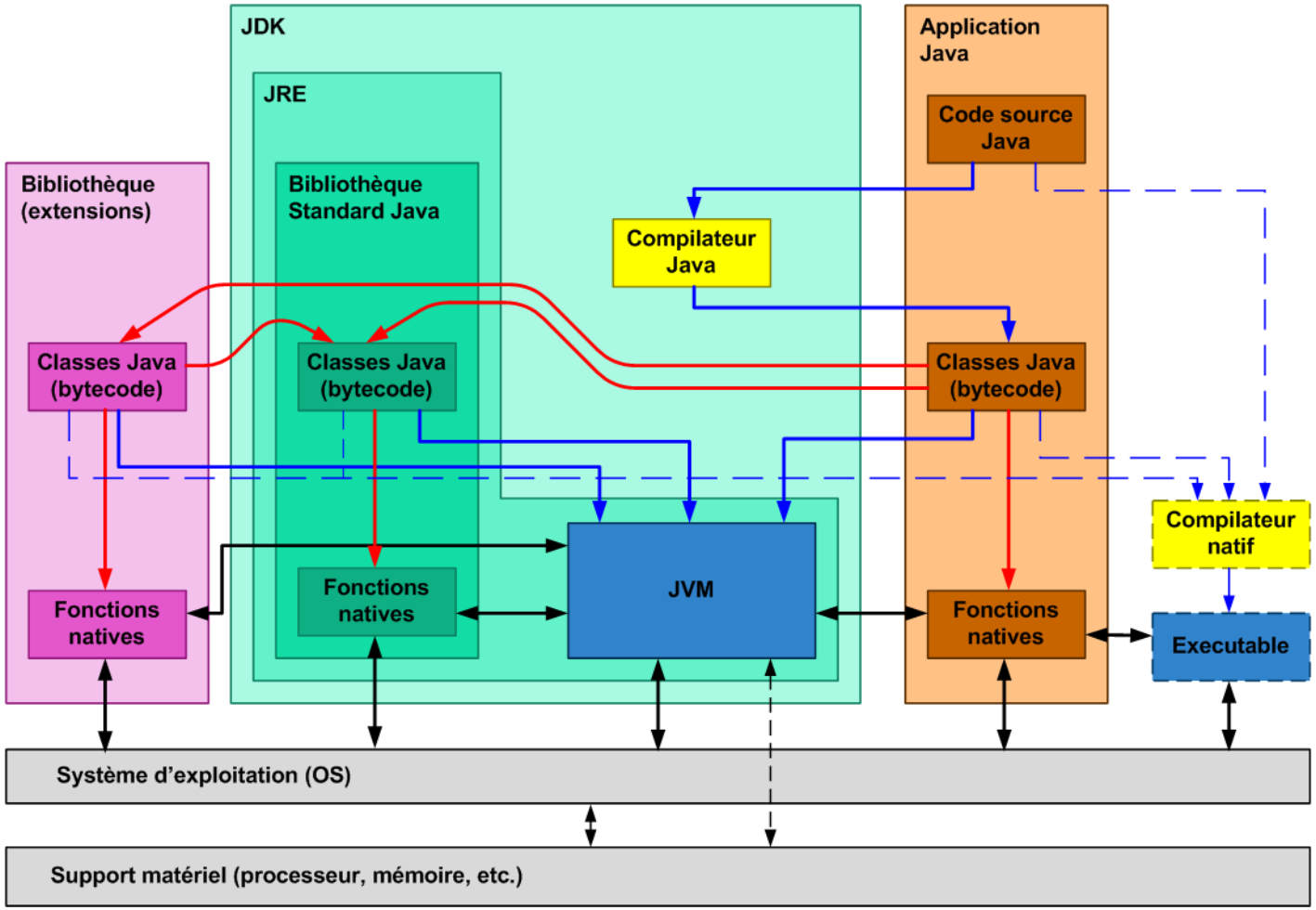
- Une étude financée par l'ANSSI réalisée par le consortium SILICOM-AMOSSYS-INRIA
- Java est-il un langage adéquat pour le développement d'applications de sécurité ?
  - Études (langage, propriétés, modèle d'exécution, etc.)
  - Recommandations à destination des développeurs
  - Propositions d'améliorations de la sécurité d'une JVM (développement d'un prototype)
  - Evaluation d'une JVM (à venir)
- Une partie des documents est disponible publiquement : `http://www.ssi.gouv.fr/site_article226.html`

Plan de la présentation

Architecture générale



Architecture générale



Propriétés de sécurité

Vérification en trois temps

- À la compilation (ex : JAVAC)
- Au chargement de classe par le vérificateur de bytecode (BCV)
- À l'exécution d'une instruction dangereuse (JIT)

Quelques propriétés

- Ni arithmétique, ni forge de pointeurs (COMP + BCV)
- Pas de dépassement des bornes d'un tableau (JIT)
- Champ `final` écrit une seule fois (COMP + $\sim$ JIT)
- Typage des instructions (COMP + BCV + JIT)
- Initialisation correcte des variables (COMP + BCV + JIT)

Mécanismes de sécurité

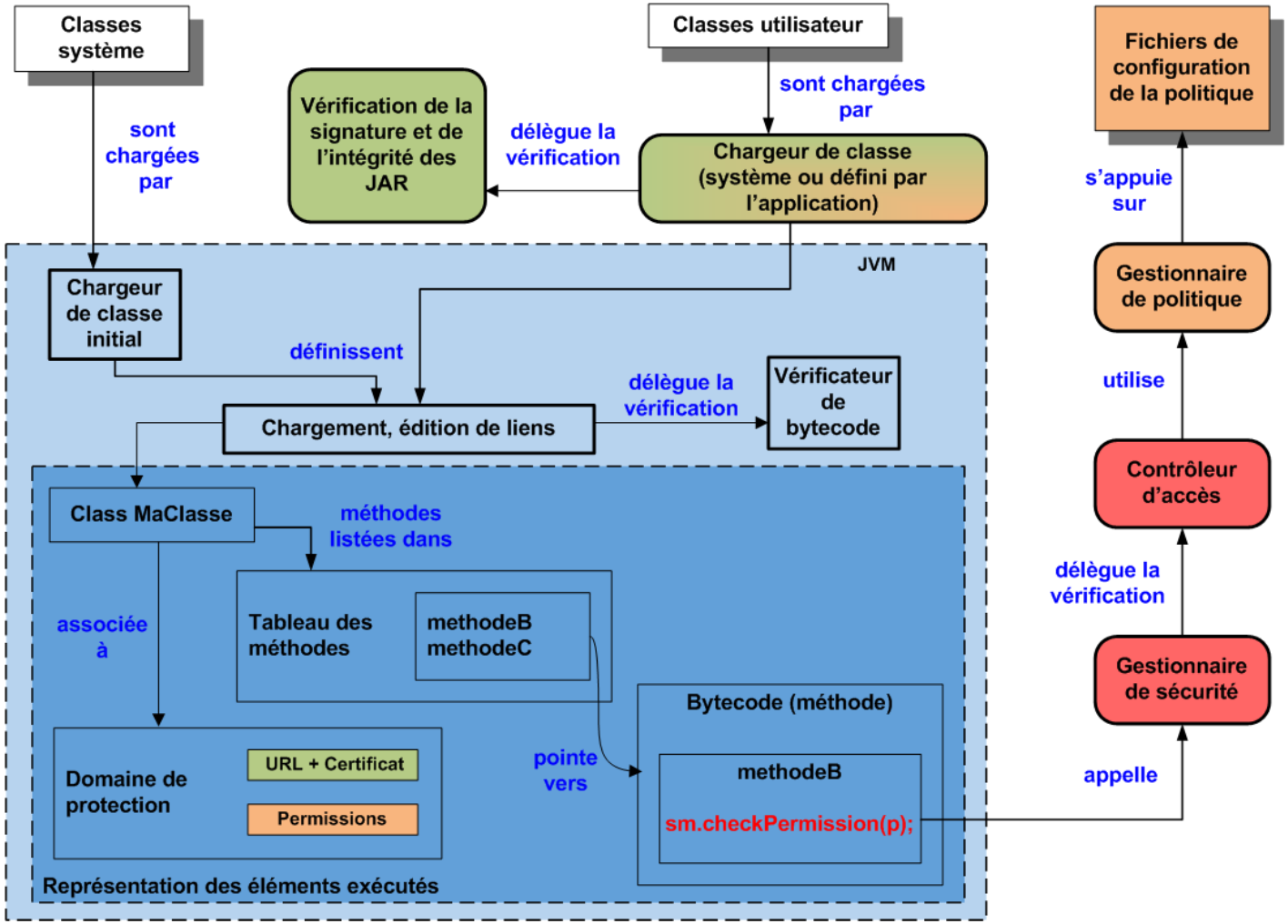
Bibliothèque standard

- Contrôle d'accès orienté code (JPSA)
- Contrôle d'accès orienté identité (JAAS)
- Chargement de classes (cloisonnement)
- Vérification de signature et d'intégrité des classes
- Primitives cryptographiques (JCE/JCA)

JVM

- Vérifieur de bytecode
- Gestion mémoire (garbage collector)
- Mécanismes d'exécution (vérification dynamiques)

Exemple : le contrôle d'accès JPSA



Problématiques de sécurité

Mauvaise utilisation des mécanismes

- Absence ou mauvaise utilisation de JPSA
  - Pas utilisé car trop complexe
  - Risques liés à la modification des éléments critiques

Langage et bibliothèque standard

- Mécanismes « dangereux » : sérialisation, réflexion
- Confiance dans l'implémentation de la bibliothèque standard (un seul niveau de privilège)
- Interfaces critiques : JNI, JVMTI, etc.

Problématiques de sécurité (suite)

JVM

- Rémanence des données confidentielles
- Echappement à la vérification de bytecode
- Confiance dans l'implémentation (« évaluabilité »)
  - Complexité (par exemple, du module d'exécution : JIT, AOT, compilation dynamique, profiling, etc.)
  - Transformations dynamiques, code intermédiaire
  - Suppressions des vérifications
  - Les propriétés sont-elles effectivement assurées ?
- Intégration avec les protections OS
- Pas de contrôle d'accès au sein de la JVM

Propositions d'amélioration

Développement et déploiement

- Guide de développement
- Guide de configuration et de déploiement

Bibliothèque standard

- Auditer, rechercher des vulnérabilités
- Limiter la surface d'attaque
- Appliquer le contrôle d'accès

Propositions d'amélioration (suite)

JVM

- Augmenter la confiance dans le module d'exécution
- Proposer un contrôle fin de la durée de vie des données confidentielles
- Améliorer l'intégration avec les mécanismes de sécurité de l'OS
- Implémenter le contrôle d'accès au sein de la JVM
- Étendre les vérification du bytecode

Vérification de bytecode étendue

Éviter la fuite d'objets partiellement initialisés

- Un objet partiellement initialisé doit être inutilisable par l'attaquant
- Système d'annotation + mécanisme de vérification statique au chargement d'une classe

```
class A {
    public A() { ...; securityManagerCheck(); }

    static void m1(@Raw A a) { ... }
    static void m2(A a) { ... }

}

class Attack extends A {
    void @RawThis finalize() { m1(this); m2(this); }
}
```

Conclusion

Bilan

- Java présente des avantages certains pour la sécurité
  - Les propriétés assurés limitent certaines classes d'attaques
  - Java propose de nombreux mécanismes de sécurité
- Mais
  - Des classes de vulnérabilités non couvertes « de base »
  - Des faiblesses (architecture et implémentation)
  - Complexité (vs sécurité) $\Rightarrow$ « château de cartes »

Perspectives

- Effort de sécurisation de la bibliothèque standard
- Augmenter la confiance dans la JVM (compromis performances vs sécurité)