» AMELIORER LA PERFORMANCE
» AUGMENTER LA SOUPLESSE
» ASSURER LA TRANSPARENCE
» REDUIRE LES COUTS
» AMELIORER LA RELATION CLIENT
» ACCELERER LA MISE SUR LE MARCHE
» INNOVER
» AMELIORER L'EFFICACITE
» ACCROITRE L'ADAPTABILITE
» GARANTIR LA CONFORMITE

Atos Origin™

CONSULTING > SOLUTIONS > OUTSOURCING

Macaron

Une porte dérobée pour JavaEE

Philippe PRADOS

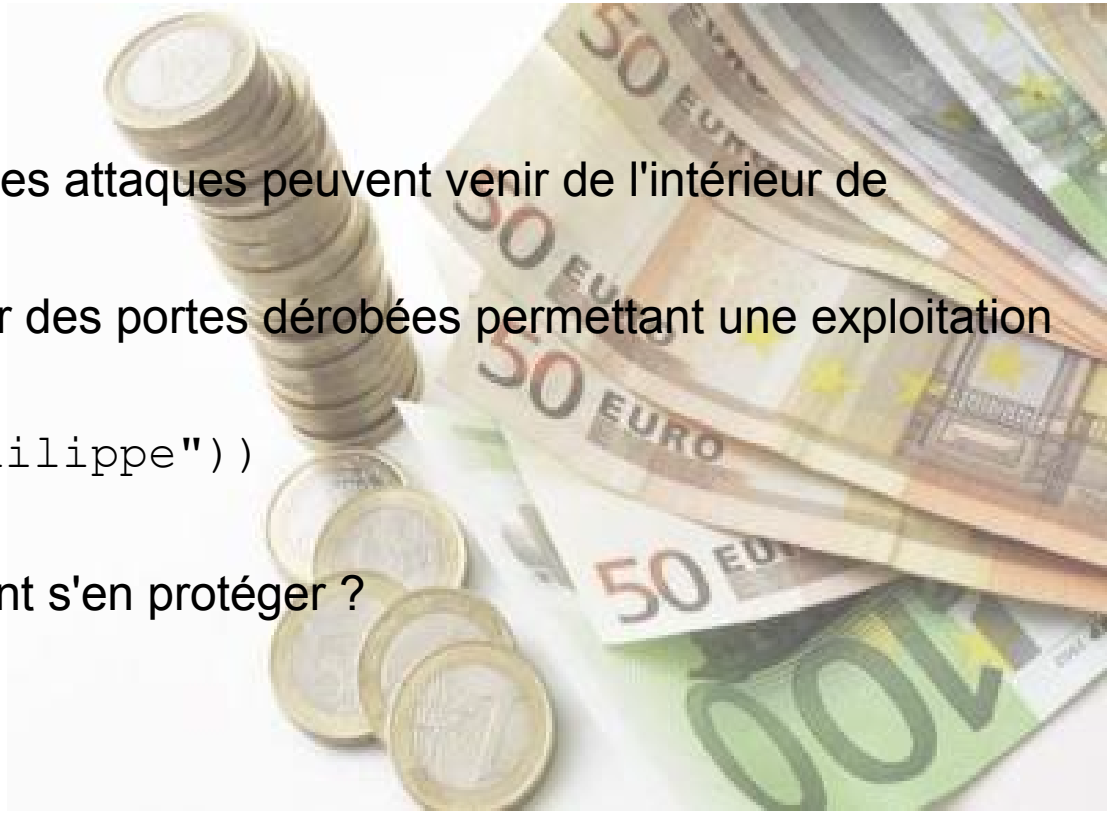
SSTIC – 5 Juin 2009

ADVANCE YOUR BUSINESS »

Curriculum Vitae

- Philippe Prados
- Architecte, Consultant
- Expert sécurité applicative

Backdoor JavaEE



- L'affaire Kerviel à démontré que les attaques peuvent venir de l'intérieur de l'entreprise.
- Les développeurs peuvent laisser des portes dérobées permettant une exploitation inhabituelle de l'application

```
if (contribuable.equals("philippe"))
     impot=(byte)impot;
```

- Quels sont les risques et comment s'en protéger ?

Sommaire

- **Objectif du pirate**
- Démonstration
- Attaques
- Solutions

À la place du pirate

- Pour la bonne cause, nous somme dans l'optique d'un pirate.
- Quels sont les objectifs ?
- Quels sont les moyens de détection à contourner ?

Une porte dérobée

- Doit résister à un audit du code de l'application
- Doit résister à un audit de configuration
- Doit résister à un pentest
- Doit résister aux évolutions futures du code
  - Il ne doit pas y avoir de dépendance avec le code existant.
- Doit contourner le firewall réseau et le firewall applicatif
  - La communication avec la porte doit être *invisible*
- Doit contourner la séparation entre les développements et la production

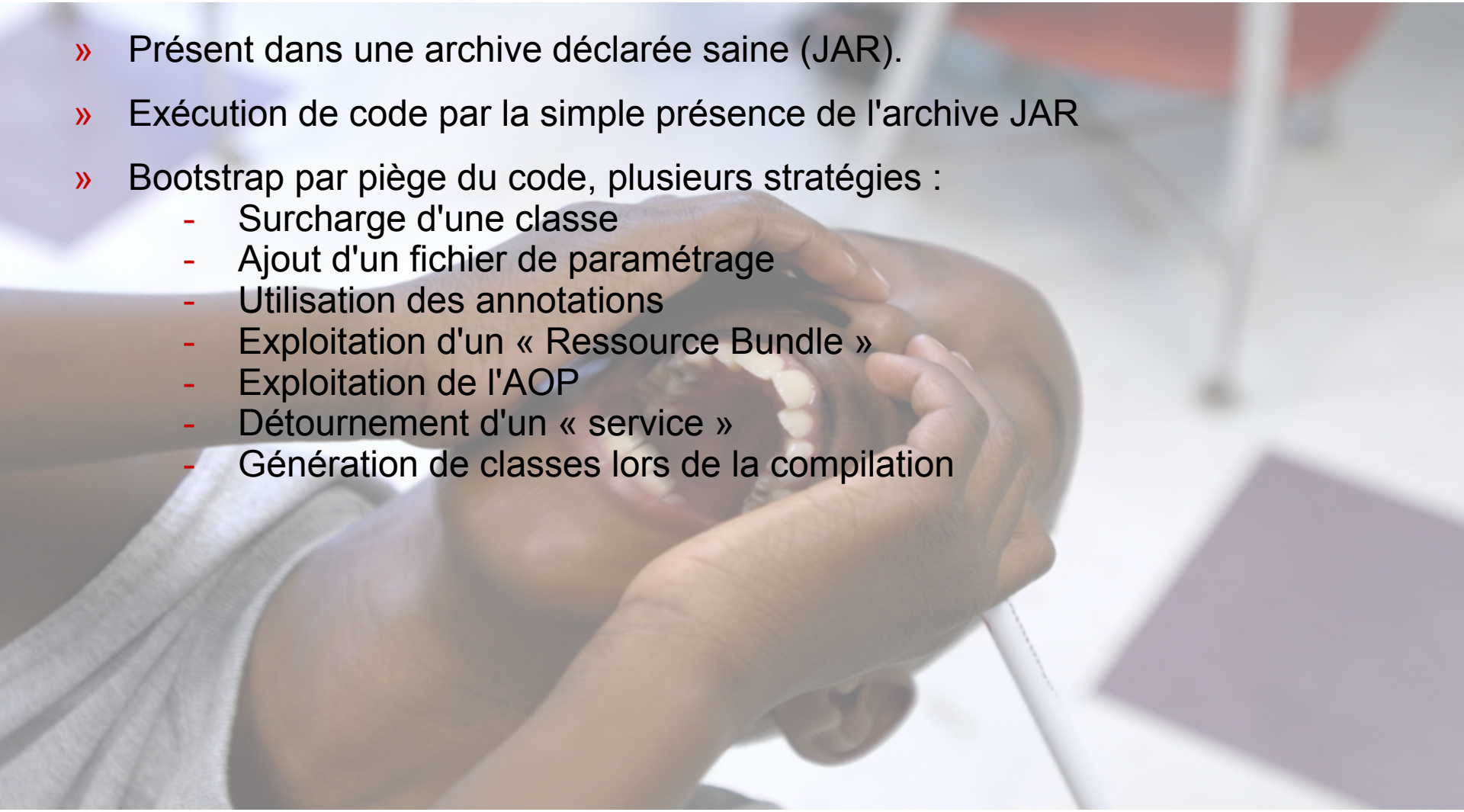
Sommaire

- Objectif du pirate
- **Démonstration**
- Attaques
- Solutions

Sommaire

- Objectif du pirate
- Démonstration
- **Attaques**
- Solutions

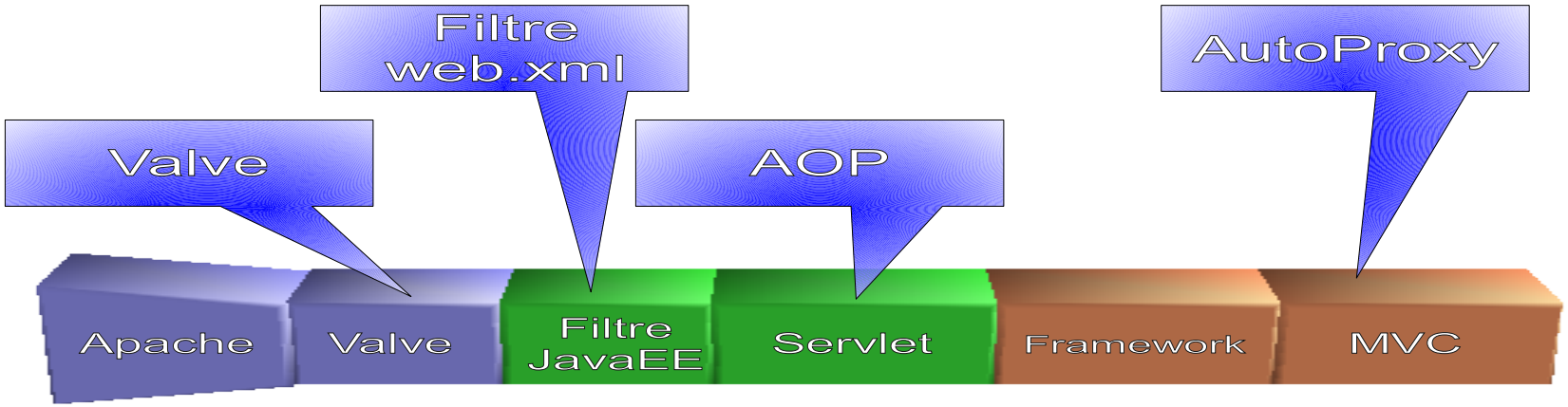
Techniques d'attaques



- Présent dans une archive déclarée saine (JAR).
- Exécution de code par la simple présence de l'archive JAR
- Bootstrap par piège du code, plusieurs stratégies :
  - Surcharge d'une classe
  - Ajout d'un fichier de paramétrage
  - Utilisation des annotations
  - Exploitation d'un « Ressource Bundle »
  - Exploitation de l'AOP
  - Détournement d'un « service »
  - Génération de classes lors de la compilation

Contourner les firewalls réseaux et applicatifs

- Il faut utiliser les canaux de communications de l'application (HTTP)
- Ainsi que ces requêtes
  - Même URL, mêmes paramètres, mêmes formats des valeurs des champs
- Solution
  - S'injecter dans le flux de traitements des requêtes HTTP pour les capturer



Détection de l'ouverture et agents

- Sur la présence d'une clef dans un champ quelconque d'un formulaire :
  - détournement du traitement vers la porte dérobée
- La clef est le mot Macaron en écriture « Elit » : M4c4r0n
  - Les logs serveurs peuvent révéler l'utilisation de la clef (généralement en GET, rarement en POST).
- Différents agents d'analyse sont disponibles
  - History (les dernières requêtes HTTP)
  - JMX (Outils de gestion dynamique du serveur)
  - JNDI (Annuaire d'objets : DB, files, etc.)
  - SQL
  - Java/javascript (Compilation et exécution dynamique de code)
  - Shell

Diffusion

- Le code est un bon exemple de ce que peut faire un développeur inconvenant
- Pour vérifier que les protections sont en place, il faut les *challenger*
- Le code proposé sert à cela (Proof Of Concept)
- Il vous permet de qualifier votre environnement
- Il est diffusé en archive non hostile, pas en source.

Le code à vocation « preuve de concept »

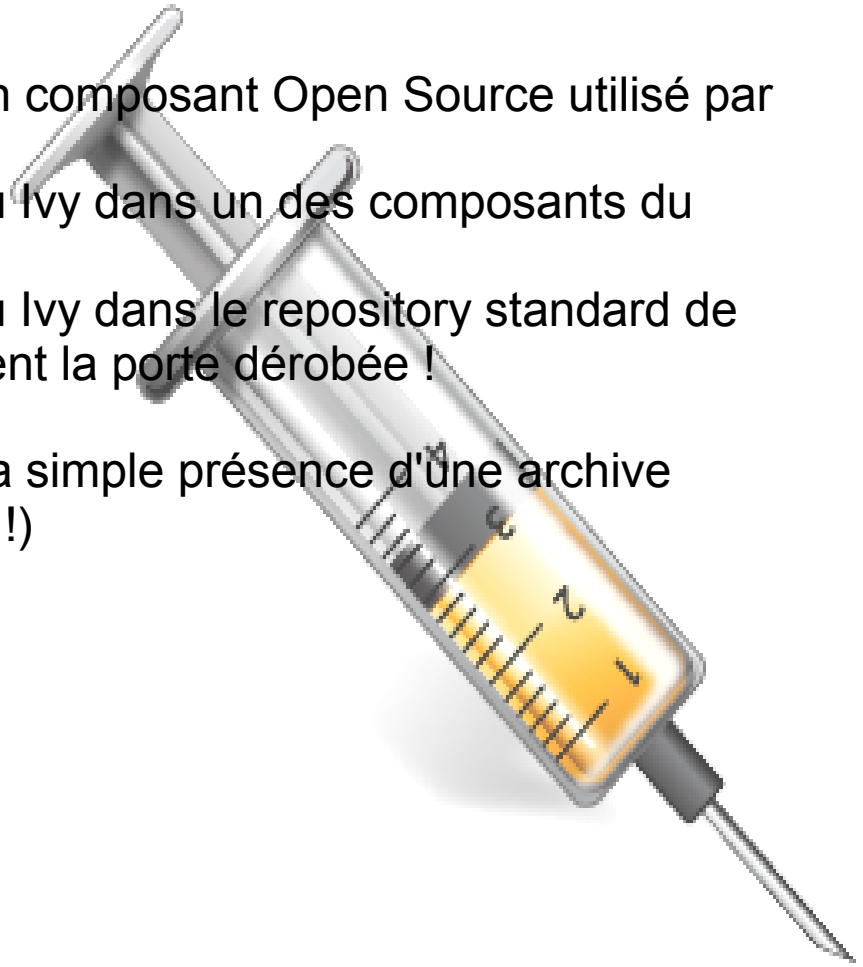
- Il peut être utilisé pour qualifier l'environnement d'exécution
- Les sources n'étant pas publique, rien ne garantie qu'il n'y a pas de deuxième porte dérobée dans le code (bon réflexe !).
- J'affirme que ce n'est pas le cas, mais vous ne pouvez le vérifier sans une étude approfondie.
- Donc, ne pas l'utiliser en production ;-)
- Vous avez alors saisi le message important de cette présentation

Diffusion = risque ?

- « *Vous diffusez un code de pirate !* »
- Un pirate étant capable de le décompiler est capable de l'écrire, et cela plus rapidement
- Il est volontairement très verbeux. Les logs présents reçoivent un message signalant clairement sa présence et ce qu'il fait.
- Une variable système est nécessaire à son exécution
  - `-Dmacaron-backdoor=i-take-responsibility-for-my-actions`
- La clef est codée en « dur » et ne peut pas être modifiée
- Une simple règle du firewall permet alors d'interdire son usage depuis le réseau. Il suffit de détecter le mot « M4c4r0n » pour couper la communication.

Injection de la porte dérobée dans un projet

- Un développeur inconvenant peut :
  - Injecter une archive
  - Injecter le code dans une archive d'un composant Open Source utilisé par le projet
  - Déclarer une dépendance MAVEN ou Ivy dans un des composants du projet
  - Déclarer une dépendance MAVEN ou Ivy dans le repository standard de MAVEN. Ainsi, tous les projets injectent la porte dérobée !
  - Véroler les repository Maven ou Ivy
  - Véroler la compilation du projet, par la simple présence d'une archive (injection possible dans une javacard !)

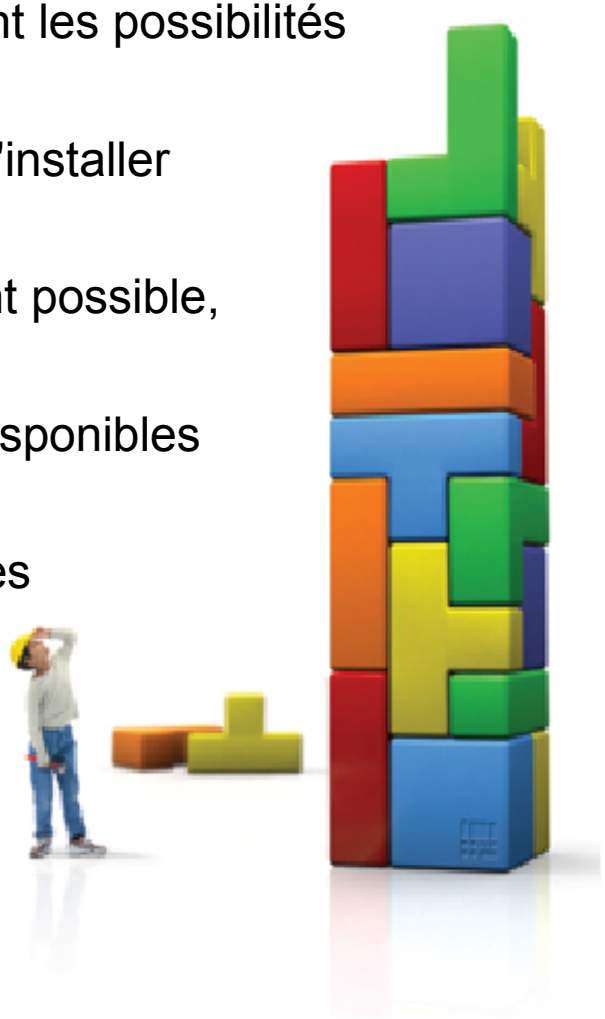


Sommaire

- Objectif du pirate
- Démonstration
- Attaques
- **Solutions**

Les solutions Java

- Java possède un mode sécurisé limitant fortement les possibilités de la porte dérobée
- Correctement utilisée, la porte dérobée ne peut s'installer et capturer tout le trafic.
- D'autres techniques de manipulation du code sont possible, mais plus spécifiques à chaque projet.
- Lorsque la sécurité Java2 est activée, les APIs disponibles sont limitées
- Les classes du serveurs d'applications ont tous les privilèges, mais pas les applications hébergées
- (Choix de Google App Engine)



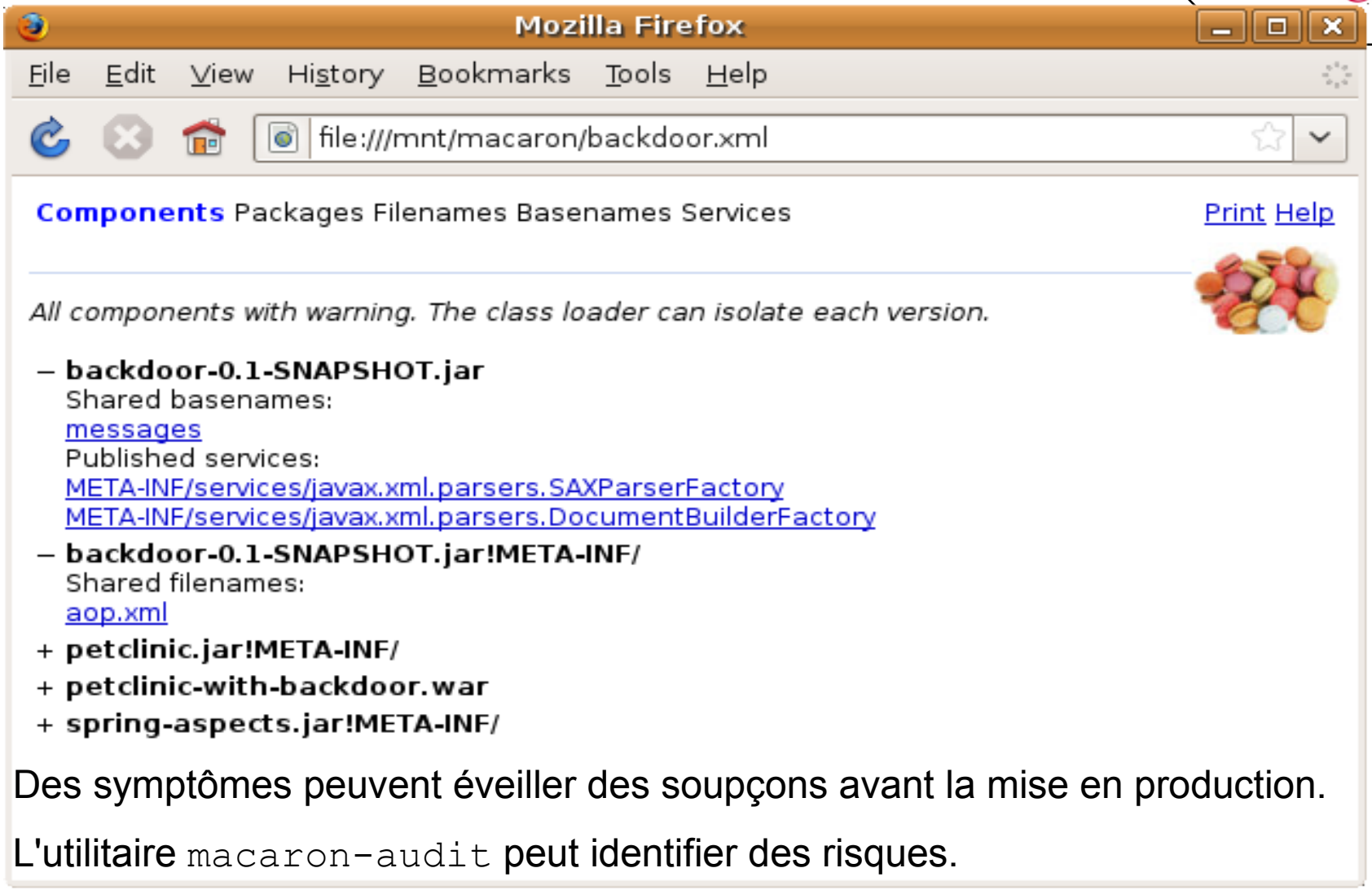
Dans la vraie vie

- Les serveurs d'applications utilisent rarement la sécurité Java2
- Pourquoi ?
  - Les développeurs n'ayant jamais testé ce mode, ils ne savent pas les droits minimums nécessaires.
  - Dans le doute, pour ne pas faire planter l'application, tous les privilèges sont ouverts
  - Cela complexifie l'installation des composants car il faut modifier un fichier global du serveur d'application (`*.policy`).
- L'utilitaire `macaron-policy` aide à gérer les privilèges à accorder à chaque composant.

Archives scellées

- Java permet de sceller les packages dans les archives. Ainsi, si la sécurité Java2 est active, toutes les classes d'un package doivent venir de la même archive.
- L'utilitaire `macaron-seal` se charge de sceller les archive.
- Il peut sceller Tomcat en une ligne de commande, pour le renforcer.

Audit



» Des symptômes peuvent éveiller des soupçons avant la mise en production.

» L'utilitaire `macaron-audit` peut identifier des risques.

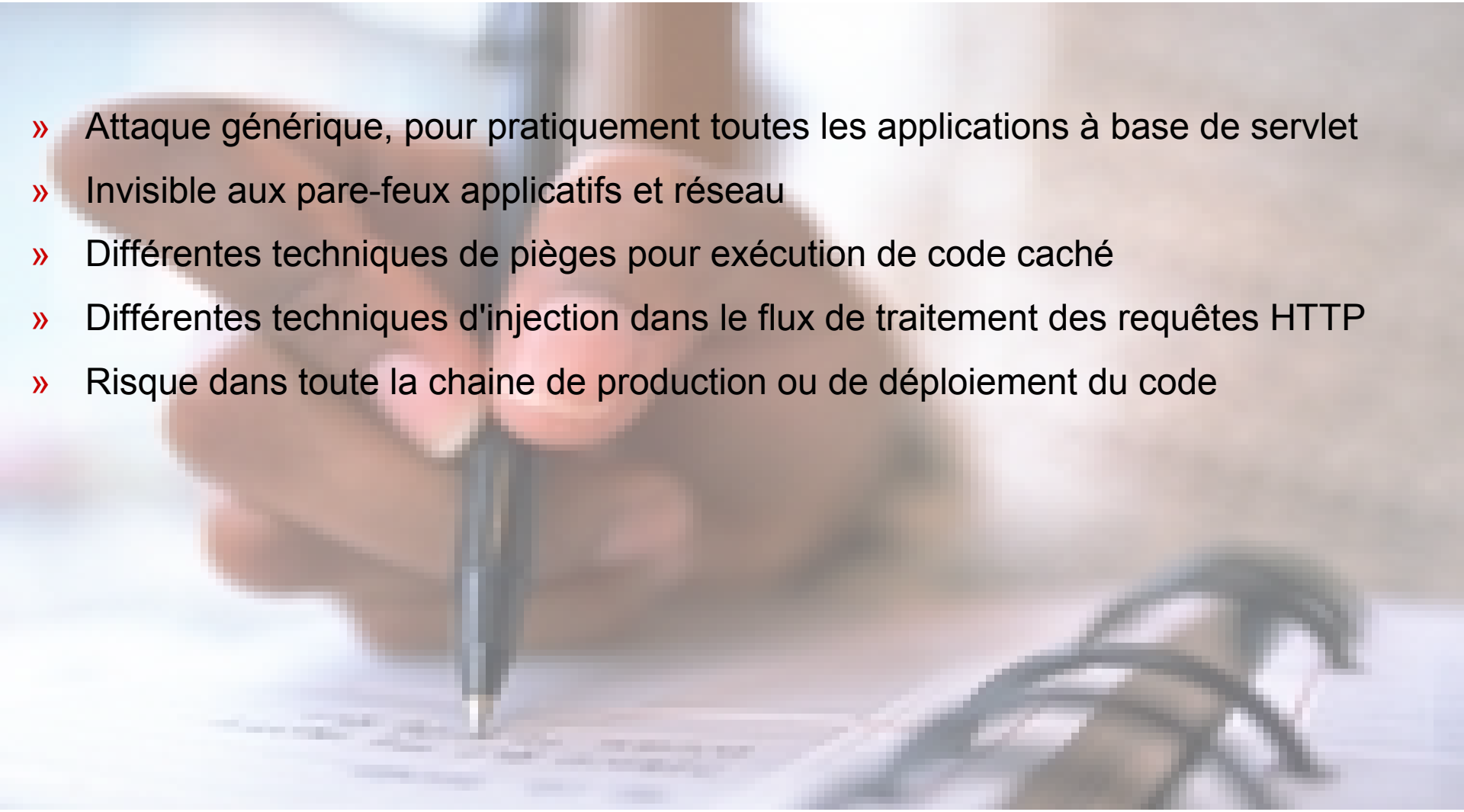
Deux patchs à OpenJDK 6 sont proposés

- Pour contrer les attaques
  - RessourceBundle
  - ServiceLocator
- Deux patchs sont proposés.
- SUN et Tomcat travaillent sur ces vulnérabilités, sur la base de mes patchs

Conclusion

- Vous n'êtes pas obligé de faire confiance aux prestataires
- Imposez l'utilisation de la sécurité Java2 dès les phases de développement
- Ne faite pas confiance aux *repositories*
- Limitez au maximum les droits ouverts aux projets
- Testez l'utilisation de la porte dérobée proposée dans votre projet.
- Tous est ici : http://macaron.googlecode.com
  - Vidéo de démonstration
  - Les utilitaires pour renforcer la sécurité et détecter les portes
  - La porte dérobée de démonstration
  - Le rapport de recherche

Synthèse



- Attaque générique, pour pratiquement toutes les applications à base de servlet
- Invisible aux pare-feux applicatifs et réseau
- Différentes techniques de pièges pour exécution de code caché
- Différentes techniques d'injection dans le flux de traitement des requêtes HTTP
- Risque dans toute la chaine de production ou de déploiement du code

Atos Origin™

CONSULTING > SOLUTIONS > OUTSOURCING

Pour plus d'informations, contacter :

Philippe PRADOS
+33 (0)6 70 03 89 60
macaron@philippe.prados.name

Atos Origin
18 avenue d'Alsace
92926, Paris la Défense Cedex
www.atosorigin.fr

ADVANCE YOUR BUSINESS »