La programmation sécurisée Java

Retours d'expérience

Axelle APVRILLE

Plan de la présentation

Retours d'expérience :

- **Architecture de JCE, JSSE et JAAS**
- **JSSE: comment initier des sockets SSL/TLS**
- **JCE: comment créer une paire de clés**
- **Les applets signées**
- **Décompilateurs & obfuscateurs**
- **Etude de cas: protection logicielle Java**

Programmation sécurisée avec Java

JCE
Java Cryptography Extension
javax.crypto.*

JSSE
Java Secure Socket Extension
javax.net.ssl.*
javax.security.cert.*

JAAS
Java Authentication Authorization Service
javax.security.auth.*
com.sun.security.auth.*

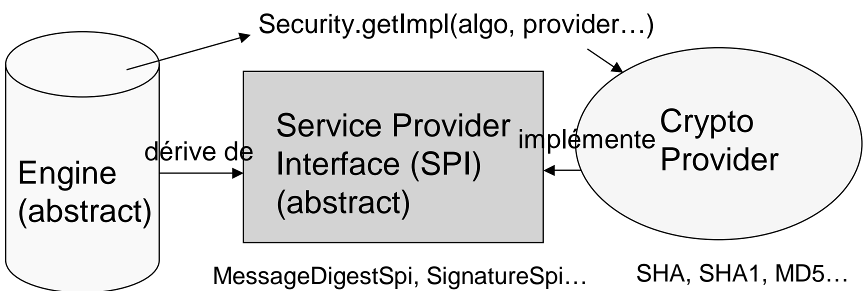
Certification Path API
java.security.cert.*

GSS
Global Security Services
org.ietf.jgss.*
com.sun.security.jgss.*

General Security
java.security.*, java.security.cert.*, java.security.interfaces.*, java.security.spec.*

JCE: Java Cryptography Extension

- **Contient les interfaces & algorithmes de chiffrement, scellement, signature, management de clés etc.**



MessageDigest, Signature, KeyPairGenerator
KeyFactory, SecureRandom…

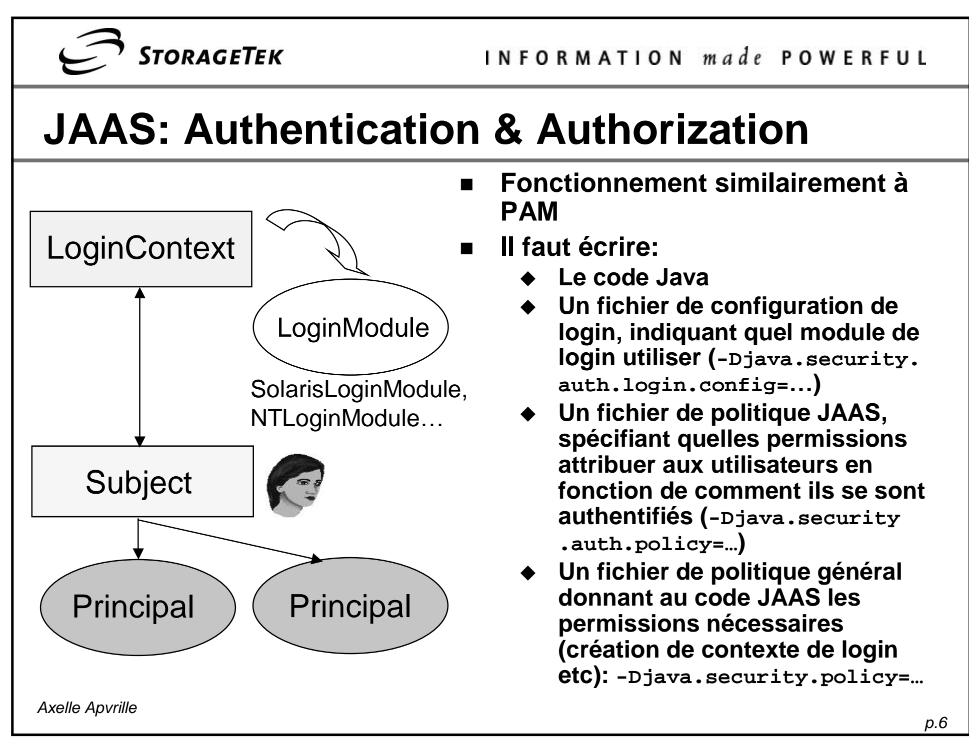
STORAGETEK INFORMATION *made* POWERFUL

JSSE: Java & SSL

- **Java Secure Socket Extension**
  - **Autrefois une extension**
  - **Inclus dans JDK 1.4**
- **Fonctionnalités**
  - **Permet de réaliser des connexions SSL/TLS en Java**
  - **SSL v3.0, TLS v1.0**
  - **RSA jusqu'à 2048 bits, DH/DSA jusqu'à 1024 bits**
  - **Chiffrement symétrique: DES, Triple DES, RC4 (« bulk encryption »)**
- **Utilisation**
  - **Création de sockets « SSL » client et serveur**

Axelle Apvrille

p.5



Exemple 1: JSSE: client / serveur TLS Java

Objectif:

1. **Créer une socket cliente TLS et une socket serveur TLS conversant ensemble sur le même port, de manière sécurisée**
2. **Vérifier que le flux des communications est sécurisé**

Initialisation de la socket serveur SSL (1)

- **Récupération du fabriquant de sockets serveurs SSLs:**

```
SSLServeurSocketFactory factory =
SSLServeurSocketFactory.getDefault();
```

- **Création de la socket:**

```
SSLServeurSocket socket = (SSLServeurSocket)
factory.createServerSocket(7200);
```

- **Paramétrages:** `socket.setNeedClientAuth(true); …`
- **Ecoute:** `Socket client = socket.accept();`

Initialisation de la socket cliente SSL (2)

- **Récupération du fabriquant de sockets clientes SSL:**

```
SSLSocketFactory factory =
SSLSocketFactory.getDefault();
```

- **Création de socket:**

```
SSLSocket socket = (SSLSocket)
factory.createSocket(colibri,7200);
```

- **Vérification des droits du serveur ?**

```
SSLSession session =
((SSLSocket)socket).getSession();
```

  - **Puis `getPeerCertificateChain(), getSubjectDN()`: est-ce bien le serveur qu'on attend ?**

- **Terminaison:** `socket.close();`

Un petit essai… (3)

```
$ java TLSserver 5000
Creating the server socket on port 5000
Listening...
Exception caught: javax.net.ssl.SSLException: No available
certificate corresponds to the SSL cipher suites which are
enabled.
-------- END --------
```

- **AIE ! Pourquoi ça ne marche pas ?!**

  SSL/TLS nécessite (au moins) une paire de clés et un certificat côté serveur… sinon comment pourrait-il y avoir un échange confidentiel et authentifié !

  Il faut les créer pour le serveur: keyStore.

  Le client doit accepter le certificat du serveur: trustStore.

STORAGETEK   INFORMATION *made* POWERFUL

Mise au point des paramètres client/serveur (4)

- **Création clés & certificat pour le serveur:**
  - **Génération d'une paire de clés:**
    - ❖ `Keytool –genkey –alias Test1 –keystore .keystore –keyalg RSA –sigalg SHA1WithRSA`
  - **Obtention d'un certificat:**
    - ❖ `Keytool –certreq –alias Test1 –file certreq.pem`
    - ❖ `Ou (triche): keytool –selfcert –alias Test1 –keystore .keystore –sigalg SHA1WithRSA`
  - **Importation d'un certificat dans le trustStore du client:**
    - ❖ `Keytool –import –alias Test1 …`
- **Passage des paramètres au serveur et au client:**

```
java –Djavax.net.ssl.keyStore=.keyStore –Djavax.net.ssl.keyStorePassword=**** TLSserver 5000
java –Djavax.net.ssl.trustStore=.trustStore TLSclient 127.0.0.1 5000
```

Axelle Apvrille

STORAGETEK   INFORMATION *made* POWERFUL

Essai: sockets « en clair » (5)

```
$ java ClearClient colibri 5000
Creating a socket on colibri:5000
Communicating... write your input to stdin
Coucou
Sending: Coucou
Expecting: 6
Receiving: 6
Communicating... write your input to stdin
```

```
$ colibri{apvria}> java ClearServer 5000
Creating the server socket on port 5000
Listening...
Connexion from client 129.80.164.112:1395
Listening for data coming from client
Reading: Coucou
Sending: 6
Listening for data coming from client
```

Axelle Apvrille

Essai: sockets TLS (6)

```
$ java -Djavax.net.ssl.trustStore=
"e:\\prog\\TLSsocket\\ressources\\.keystore"
TLSclient colibri 5000
Creating a socket on colibri:5000
Communicating... write your input to stdin
Coucou
Sending: Coucou
Expecting: 6
Receiving: 6
Communicating... write your input to stdin
```

Session: TLSv1
Echange: RSA
Bulk: RC4 128
Sceaux: SHA1

```
java -Djavax.net.ssl.keyStore=./ressources/.keystore
-Djavax.net.ssl.keyStorePassword=steo31 TLSserver 5000
Creating the server socket on port 5000
Listening...
Connexion from client 129.80.164.112:1398
Listening for data coming from client
Reading: Coucou
Sending: 6
Listening for data coming from client
```


Ecoute de SSL / TLS (7)

Sans TLS :

```
colibri{root}# ./tcpdump -X host colibri and tcp port 5000
tcpdump: listening on hme0
15:59:47.241580 129.80.164.112.1395 > colibri.5000:
P 2465375352:2465375360(8) ack 1176339639 win 17520 (DF)
0x0000   4500 0030 abfd 4000 8006 02e6 8150 a470       E..0..@......P.p
0x0010   8150 a4d3 0573 1388 92f2 a478 461d 84b7       .P...s.....xF...
0x0020   5018 4470 cee2 0000 436f 7563 6f75 0d0a       P.Dp....Coucou..
```

Avec TLS :

```
colibri{root}# ./tcpdump -X host colibri and tcp port 5000
tcpdump: listening on hme0
16:11:29.219015 129.80.164.112.1398 > colibri.5000:
P 2625216489:2625216522(33) ack 1347402001 win 16764 (DF)
0x0000   4500 0049 af92 4000 8006 ff37 8150 a470       E..I..@....7.P.p
0x0010   8150 a4d3 0576 1388 9c79 9fe9 504f b911       .P...v...y..PO..
0x0020   5018 417c 880c 0000 1703 0100 1cda 164b       P.A|..........K
0x0030   9d9b a478 1326                                ...x.&
```

STORAGETEK INFORMATION *made* POWERFUL

JSSE & SSL/TLS: récapitulatif (8)

- **J'aime:**
  - Support natif dans Java 1.4
  - Tout se trouve dans javax.net.ssl: SSLSocket, SSLSession, SSLServerSocket …
  - Empaquetage dans SSL / TLS complètement caché: les applications n'ont plus qu'à se soucier des données à transporter (+ vérification d'identité)
  - Paramétrage plus fin possible
- **Je n'aime pas :**
  - Pas facile à débugger, car on n'a pas accès aux trames de négociations, ni aux trames de données, sauf via `-Djava.security.debug=ssl`
  - Trop lié aux keyStore Java, ce qui n'est pas pratique.

Axelle Apvrille

STORAGETEK INFORMATION *made* POWERFUL

Exemple 2: JCE: génération de clés

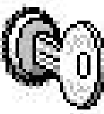
Objectifs:

1. Ecrire du code Java permettant de générer une paire de clés RSA 2048 bits
2. Sauvegarder cette paire de clés sur fichier
3. Etre capable de relire la paire de clés

Axelle Apvrille

INFORMATION *made* POWERFUL

Génération de clés (1)



- **Package: java.security.***
- **Initialisation:**
  - **Préciser le nom de l'algorithme (tel que référencé par le provider)**
    - ❖ `KeyPairGenerator kpg = KeyPairGenerator.getInstance(RSA);`
  - **Effectuer l'initialisation indépendante de l'algorithme**
    - ❖ `kpg.init(2048,new SecureRandom());`
  - **Ou effectuer une initialisation dépendante pour d'autres algorithmes (DSA, DH…) en passant des AlgorithmParameterSpecs (interface Java)**
- **Génération (long):**
  - ◆ `kpg.generateKeyPair();`
  - ◆ `RSAPrivateKey prvKey = (RSAPrivateKey) kpg.getPrivate();`
  - ◆ `RSAPublicKey pubKey = (RSAPublicKey) kpg.getPublic();`

Axelle Apvrille

STORAGETEK INFORMATION *made* POWERFUL

Un exemple d'implémentation (2)

KeyPairGenerator
generateKeyPair()
getAlgorithm()
getInstance()
getProvider()
initialize()

Classe abstraite

Generator
keySize : int = 0
algoName : String = ""
kpg : KeyPairGenerator = null
pair : KeyPair = null
prvKey : PrivateKey = null
pubKey : PublicKey = null
generate()
getPublic()
getPrivate()
getSize()
getAlgorithm()

Instantiation en passant
Un objet SecureRandom
Et une taille de clés

RSAKeyPairGenerator
PRNG : SecureRandom = new SecureRandom()
DEFAULT_KEYSIZE : int = 2048
RSA_ALGO_NAME : String = RSA

Axelle Apvrille

Sauvegarder les clés sur fichier (3)

- **Solution A: dumper les objets Java RSAPrivateKey et RSAPublicKey**

```
RSAPublicKey pubkey;
FileOutputStream fpub = new FileOutputStream(« key.pub »);
ObjectOutputStream opub = new ObjectOutputStream(fpub);
opub.writeObject(pubkey);
```

  RSAPublicKey est une interface implémentée par le provider… donc le dump est dépendant du provider !

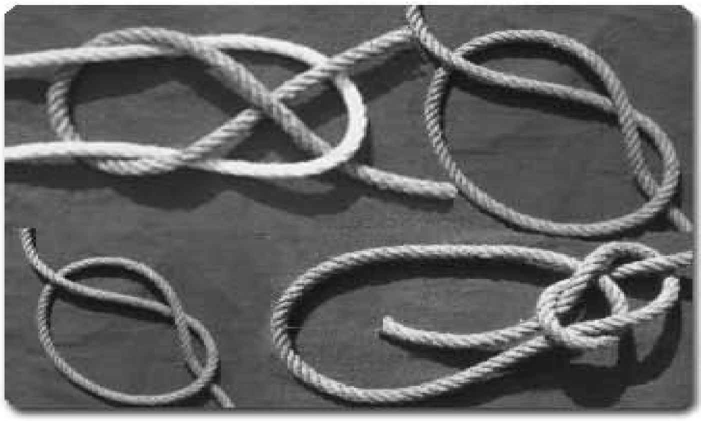
- **Solution B: les keyStores Java**
  - **Format « Sun » JKS, ou PKCS#12**
  - **En passant par la classe KeyStore**
- **Solution C: utiliser les encodages « standards » proposés par le provider**
  - **X509EncodedKeySpec pour la clé publique**
  - **PKCS8EncodedKeySpec pour la clé privée**
    - FileOutputStream fpriv = new FileOutputStream(« key.prv »);
    - fpriv.write(prvkey.getEncoded());

Relire les clés (4)

```
KeyFactory factory = KeyFactory.getInstance("RSA");
FileInputStream pubfile = new FileInputStream(filename+PUB);
byte [] byte_pub = new byte[pubfile.available()];
pubfile.read(byte_pub);
X509EncodedKeySpec pubspec = new X509EncodedKeySpec(byte_pub);
pubkey = (RSAPublicKey) factory.generatePublic(pubspec);
pubfile.close();
```

- **Lire les octets depuis le fichier**
- **Reconstituer la clé à partir des spécifications de son encodage**
- `generatePublic()` **est mal choisi: c'est plutôt une « reconstruction » qu'une génération.**

Les applets signées

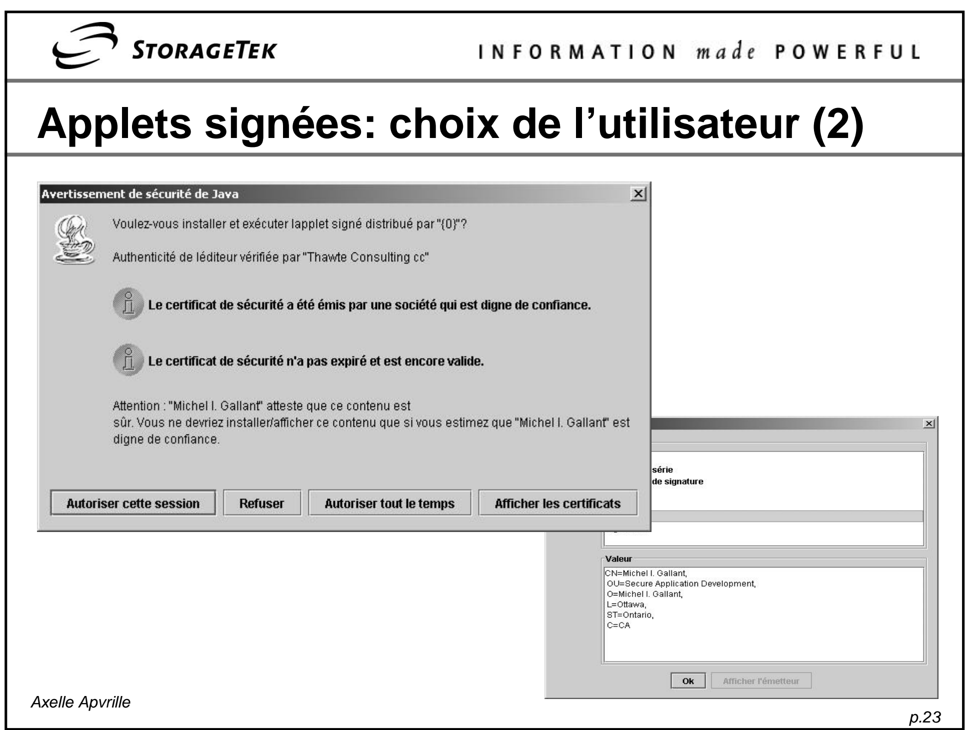


Les applets signées (1)

- Les applets jouent dans leur bac à sable…
  - Interdit d'accéder aux fichiers locaux de la machine,
  - Interdit de charger du code natif,
  - Interdit d'ouvrir une socket,
  - Interdit d'ouvrir une fenêtre sans la mention « applet window » …
- On peut requérir des permissions spéciales :
  - De manière **statique**, en modifiant l'environnement Java (`${JREHOME}/lib/security/java.policy`),
  - De manière **dynamique**, en **signant les applets** : l'utilisateur choisit alors de faire confiance ou pas, permettant à l'applet d'accéder à des privilèges ou non.

STORAGETEK INFORMATION *made* POWERFUL

Applets signées: choix de l'utilisateur (2)



Axelle Apvrille

p.23

STORAGETEK INFORMATION *made* POWERFUL

Jouer avec l'environnement Java… (3)

- **Global policy** file: `${JREHOME}/lib/security/java.policy`
- **User policy** file: ~/.java.policy, c:\windows\java.policy, c:\Documents and Settings\user\.java.policy
- Outil: **policytool**

  Permet d'assigner des permissions à un code donné (en provenance d'une URL donnée) ou signé par une personne donnée:

```
grant signedBy « Axelle » {
permission java.io.FilePermission « /tmp »,
    « read,write,delete »;
permission java.net.SocketPermission « *:5000 »,
    « accept,listen, connect »;
}
```

- Voir aussi `${JREHOME}/lib/security/java.security`

Evitez de jouer avec le feu !

Axelle Apvrille

p.24

STORAGETEK INFORMATION *made* POWERFUL

Signer des applets (4)

Trois méthodes différentes :

- La méthode « Netscape »
- La méthode « Sun » avec les Java Plug-In
- La méthode « Microsoft » pour Internet Explorer

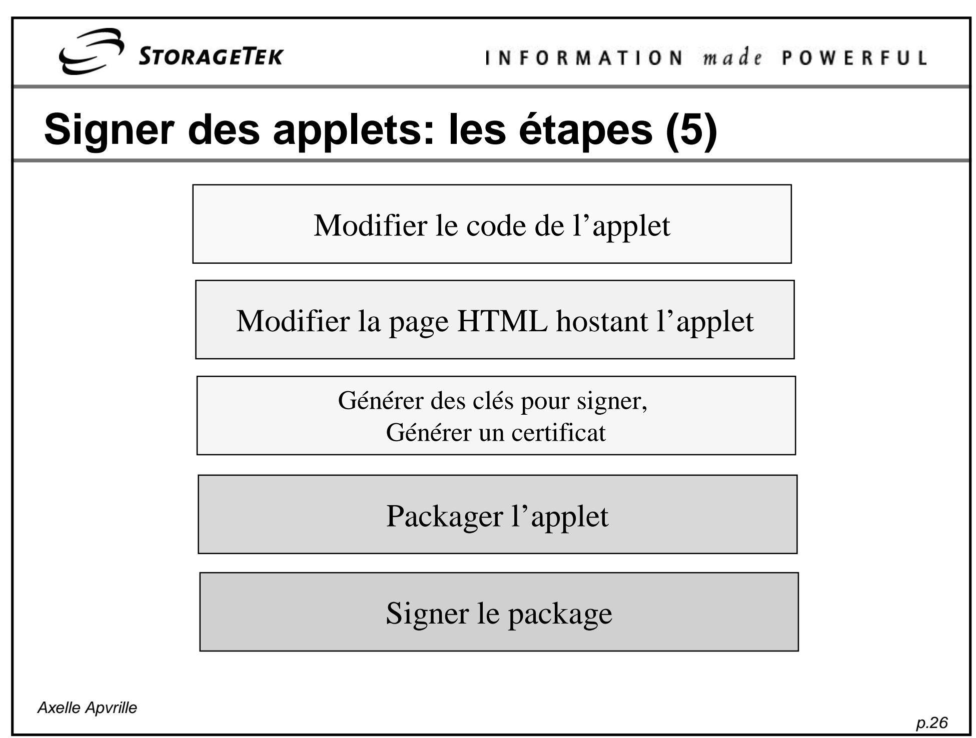
Hélas, toutes incompatibles !
C'est un vrai parcours du combattant pour faire une applet signée fonctionnant dans tous les cas…

Axelle Apvrille

p.25

STORAGETEK INFORMATION *made* POWERFUL

Signer des applets: les étapes (5)



Axelle Apvrille

p.26

Signer des applets: méthode « Netscape » (6)

- **Dans le code:**
  - **Requérir les privilèges nécessaires**
  - **Gérer les exceptions qui peuvent survenir si l'utilisateur n'accorde pas la permission à l'applet**
  - **Voir les « *Capabilities Classes* » http://developer.netscape.com/docs/manuals/signedobj/capabilities/index.html**

```
import netscape.security.*;
PrivilegeManager.enablePrivilege(« UniversalFileAccess
    »);
…
PrivilegeManager.revertPrivilege();
```


Signer des applets: méthode « Netscape » (7)

- **Utiliser signtool : http://developer.netscape.com/docs/manuals/cms/41/adm_gide/app_sign.htm#1007529**
- **Les clés et certificats sont contenus dans les bases de données du navigateur: cert*.db et key*.db (protégés par mot de passe).**
- **Signtool signe un répertoire avec des clés données, et place le tout dans un JAR signé :**

```
Signtool –k <NickName> -J <SignedJar> <DIR>
```

- **Signtool permet de générer des clés/certificats de test (option –G)**

Signer des applets: méthode « Microsoft » (8)

- **Récupérer les clés (clé privée *.pvk, et clé publique + certificat *.spc) et les importer (PVK File Importer: http://msdn.microsoft.com /vba/technical /pvk.asp)**
- **Packager les classes avec l'outil cabarc du Microsoft Java SDK :**
  - Cabarc -p n MyCab.cab Toto.class Tata.class
- **Utiliser les outils « Authenticode for Internet Explorer » pour signer l'applet :**
  - Signcode -jp Low -spc <public key>.spc -v <private key>.pvk <cabfile>.cab
  - **Le niveau « Low » permet à l'applet de tourner avec la sécurité affectée à la zone « Low » (celle qui a le moins de sécurité). On peut aussi choisir Medium, ou High.**
- **Les outils** makecert **et** cert2spc **permettent de générer des clés & certificats de test (voir outils Authenticode)**

Signer des applets: méthode « Java Plug- In » (9)

- Générer des clés pour signer, faire une demande de certificat et importer le certificat reçu:
  - keytool -genkey -keyalg rsa -alias MyNickName
  - keytool -certreq -alias MyNickName
  - keytool -import -alias MyNickName -file <File>.cer
- Packager les classes avec l'outil Jar
- Signer l'applet soit avec jarsigner (outil Sun), soit signtool (Netscape):
  - jarsigner <MyJar>.jar <MyNickName>
- Génère des fichiers :
  - META-INF/MyNickName.SF: hashage SHA de chaque classe du JAR (en prenant les lignes du MANIFEST.MF)
  - META-INF/MyNickName.RSA/DSA: signature du fichier *.SF au format PKCS#7
- On peut générer des clés & certificats de test avec keytool

Récapitulatif sur comment signer les applets (10)

	Java Plug In	Netscape	Microsoft Authenticode
Code		Requires modifications	
HTML	<applet code="Toto.class" archive="SignedApplet.jar">		<applet code="Toto.class"> <param name= « cabbase » value= « mycab.cab »>
Keys	Keytool	Use netscape Test: signtool	Use Internet Explorer Test: makecert & cert2spc
Certificate	Keytool, Test certificates : signtool		
Package	Jar	signtool	Cabarc
Sign	jarsigner		Signcode
Comments	Browser independent		

Référence sur les applets signées

- **« Java Science », Michel Gallant, http://home.istar.ca/~neutron/java.html**
- **« Java Security », Michael Thomas, http://www.michael-thomas.com/java/javaadvanced/security/**
- **« *Securing Java* », Gary McGraw, Ed Felten, http://www.securingjava.com/toc.html**
- **« *Java security* », Scott Oaks, 2nd Edition, O'Reilly**
- **« The Java Plug-In 1.4 Developer Guide », http://java.sun.com/j2se/1.4/docs/guide/plugin/developer_guide/contents.html**
- **Microsoft Java SDK, http://www.microsoft.com/java**
- **Object Signing Ressources, http://developer.netscape.com/docs/manuals/signedobj/overview.html**
- **« The original hostile applets », http://www.cigital.com/hostile-applets/**

Décompilateurs & obfuscateurs Java

- **Qu'est-ce que les décompilateurs révèlent du bytecode Java ?**
- **Comment s'en prémunir ?**
- **Les obfuscateurs sont-ils efficaces ?**

Décompilation sous Java (1)

- **En Java, la décompilation des *.class est très efficace: on pert les commentaires et parfois qq détails**
- **Cela pose de graves problèmes d'intégrité de packages**

```
java –cp « mocha.zip » mocha.Decompiler Sample.class
```

```
/**
 * This is a sample class we're going
 * to use to test the mocha Decompiler
 * @author Axelle Apvrille
 * @version 1.0
 */
public class Sample {
    public static final int MAX = 10;
    int counter = 0;

    public Sample() {
          for (int i=0;i<MAX;i++)
              counter++;
    }

    public static void main(String args[]){
          Sample s = new Sample();
    }
}
```

```
/* Decompiled by Mocha from Sample.class */
/* Originally compiled from Sample.java */
public synchronized class Sample
{
    public static final int MAX = 10;
    int counter;

    public Sample()
    {
        counter = 0;
        for (int i = 0; i < 10; i++)
            counter++;
    }

    public static void main(String astring[])
    {
        Sample sample = new Sample();
    }
}
```

Décompilateurs Java (2)

- **Mocha**
  - Simple, gratuit et le plus célèbre ?
  - N'est plus supporté
  - Repris dans Jbuilder
  - Ne fonctionne plus sur le bytecode généré par le JDK 1.4 ? (JDK 1.3.1 ok).
- **De nombreux autres décompilateurs**
  - beaucoup n'existent plus,
  - beaucoup sont payants…
  - mais il en reste encore quelques uns de gratuits,
  - notamment DJ Java Decompiler, gratuit et graphique (et marche pour JDK 1.4)

Les résultats de l'obfuscation: exemple (3)

```
public class A
{
    public A()
    {
        C = 0;
        false;
        int i;
        i;
          goto _L1
_L3:
        C++;
        i++;
_L1:
        i;
        10;
        JVM INSTR icmplt 26;
           goto _L2 _L3
```

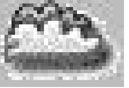
```
_L2:
            return;
            throw ;
    }

    public static void
main(String args[])
    {
        A a = new A();
    }

    public static final int B =
10;
    int C;
}
```

Obfuscateurs de code (4)

- **Principes d'obfuscation**
  - « Anonymiser » les noms des variables, méthodes, classes
  - Modifier les boucles (for, while, goto…)
  - Rajouter du code inutile (!)



- **Problèmes**
  - Il faut que le package reste facilement compilable et exécutable !
  - Attention aux méthodes des classes java.lang.Class et java.lang.ClassLoader

Références sur la décompilation & obfuscation

- Mocha (free) http://www.brouhaha.com /~eric/ computers/mocha.html
- Homebrew Decompiler, Pete Ryland (GPL) http://www.pdr.cx/projects/hbd/
- SourceTec Java Decompiler http://www.srctec.com/decompiler/
- DJ Java Decompiler http://members.fortunecity.com /neshkov/dj.html (free)

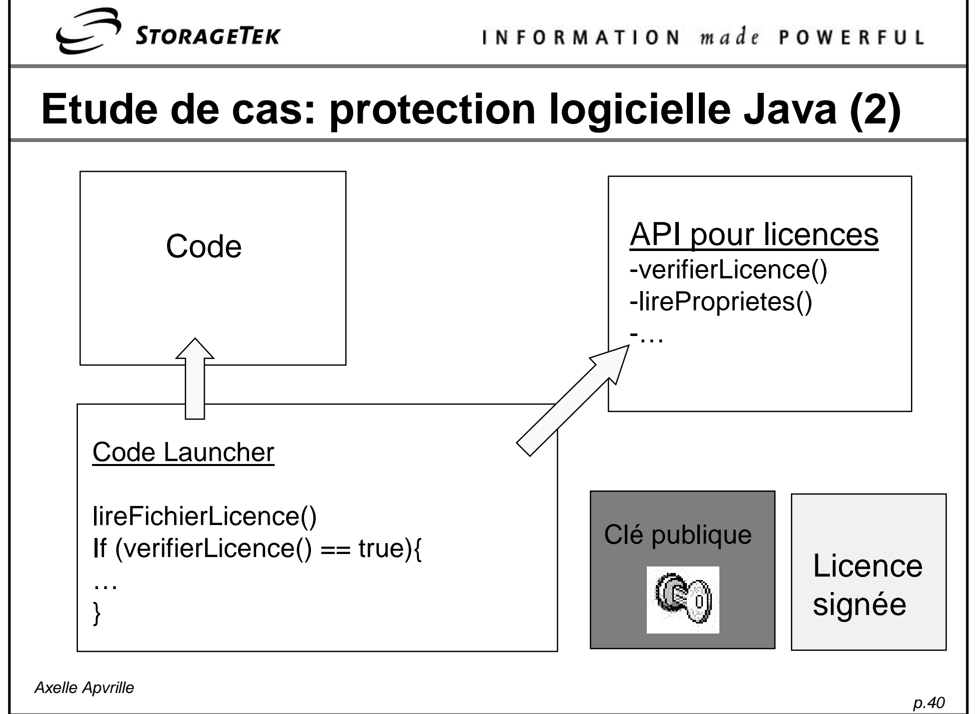
- Crema obfuscator http://www-sor.inria.fr/~java/tools/crema/
- SmokeScreen http://www.leesw.com/ (shareware)
- Jshrink http://www.e-t.com/jshrink.html
- Marvin Obfuscator http://www.drjava.de/obfuscator/#history (free)
- RetroGuard Bytecode obfuscator http://www.retrologic.com/retroguard-main.html (LGPL)

Etude de cas: protection logicielle Java (1)

- **Objectif: développer un système de contrôle de licence pour logiciel écrit en Java**
  - L'utilisateur doit posséder une licence valide pour pouvoir lancer le logiciel
  - Possibilité d'évaluation du logiciel pendant une période donnée
  - Possibilité de gestion de licences « site »
  - Réutilisabilité du code…
- **Stratagème étudié**
  - On dispose d'une clé privée qui signe le fichier de licence. La clé privée est conservée secrètement.
  - La clé publique de vérification du fichier de licence est contenue dans le JAR
  - L'API de vérification de licence vérifie la signature du fichier de licence fourni

STORAGETEK INFORMATION *made* POWERFUL

Etude de cas: protection logicielle Java (2)



Etude de cas: protection logicielle Java (3)

- **Attaques possibles:**
  1. **Le pirate dé-jarre le soft, remplace la clé publique par la sienne, et signe la licence avec sa clé privée**
  2. **Le pirate dé-jarre le soft, décompile les classes, repère l'appel au contrôle de licence et le met en commentaires !**

- **IL N'Y A PAS DE SOLUTION (SIMPLE) AU PROBLEME (ou alors je suis preneuse !) :**

  Si on met la clé publique en dur dans le code, le pirate peut la changer à la décompilation. Idem si on ne met qu'un hash de la clé.

  Sceller le JAR empêche un pirate de fournir un mauvais JAR, mais ne l'empêche pas de contourner la protection logicielle…

Etude de cas: protection logicielle Java (4)

Si on chiffre la licence, alors il faut fournir une clé secrète. Le pirate peut alors déchiffrer la licence et la modifier…

Si on chiffre les *.class, alors comment proposer une licence d'évaluation temporaire ? Dès que les classes ont été déchiffrées une fois, on peut les mémoriser… Quant à chiffrer/déchiffrer à la volée cela consommerait trop de temps.

Utilisation d'un système client / serveur (style FlexLM)

- **Conclusion:**

  Être sûr que le jeu en vaut la chandelle

  Faire un système simple et obfusquer partiellement le code. Complexité ne rime pas toujours avec sécurité !

  Ou opter pour un système plus radical client / serveur

Conclusion

- **Comment programmer en « toute » sécurité ?**
  - **Utiliser des algorithmes éprouvés,** style RSA, AES et non pas des inventions « maison » (et ne pas réinventer la roue),
  - **Ne jamais opter pour la paresse:** style laisser une brèche ouverte « temporaire », chiffrer/sceller seulement une partie des données…
  - **Se placer dans la peau de toutes les personnes intermédiaires pour trouver des failles:** l'utilisateur, le client, le serveur, le pirate, l'initialisation des paramètres, les traces du débuggage, la maintenabilité des logiciels, l'utilisation du logiciel à d'autres fins…
  - **Réfléchir toujours en termes (1) d'intégrité (2) de confidentialité (3) d'authentification,**
  - **Discuter avec soin de l'architecture avec d'autres,**
  - **Relire son code,**
  - **Dans l'impossibilité de faire bien (ou mieux), faîtes simple.**
- **Hélas, liste non exhaustive …**