La sécurité dans Java

Claude Scarpelli – INFOBIOGEN

14 Mai 1996

- Java 1
- JavaScript 3
- attaques connues de JavaScript 5
- sécurité à plusieurs niveaux 6
- langage et compilateur 7
- *byte code verifier* 8
- La machine virtuelle Java 10
- *Le ClassLoader* 15
- restrictions d'accès 17
- exemples de configuration 19
- récapitulatif 22
- attaques connues de Java 24
- pointeurs, crédits 27

Java

Avant tout, une machine virtuelle.
Mais aussi un langage de programmation et une bibliothèque (graphique, réseau, threads…) orientés vers les applications clients / serveurs dans l'entreprise (Intranet) et les applications chargeables à la demande sur le Web (Applet).

Le langage Java est:

simple

> *Java, c'est C++ sans les couteaux ni les revolvers — James Gosling*

orienté objet

distribué le code réseau TCP/IP ainsi que des protocoles de plus haut niveau (HTTP, FTP, NNTP, SMTP…) sont intégrés dans la bibliothèque

robuste

indépendant de l'architecture Le *byte code* Java est indépendant de l'architecture de la

plateforme d'exécution. Des implémentations existent aujourd'hui sur UNIX, Mac et Win 32.

portable Le langage est hautement portable, sans ambiguïté. Les caractéristiques des types de bases sont connues (taille des entiers par exemple).

performant

multi-threadé

dynamique Les classes sont chargées à la demande, dynamiquement :

```
proto://domain.com/...
```

Le browser va tenter de charger le fichier `proto.class`, sous le contrôle de la classe `SecurityManager`

sûr

JavaScript

- un *scripting language*, avec une syntaxe proche de celle de Java mais sans compilation → *pas de machine virtuelle*
- NetScape only
- Security through Obscurity
- Les problèmes connus seraient limités à des violations de l'intimité (listes des pages lues, hotlist). Il serait possible de lire, voire d'écrire dans un fichier.
- NetScape Communication Corp bouche les trous au fil de l'eau

- LiveConnect (Netscape 3) : intégration de JavaScript, de Java et des plug-ins :

  JavScript peut appeler des méthodes Java et des plug-ins

  Java peut appeler des fonctions JavaScript et des plug-ins

  plug-in peuvent définir des classes Java, les appeler, ainsi que les fonctions JavaScript

attaques connues de JavaScript

Quelques trous révélés par
John Robert Loverso, OSF dans NetScape 2.01. Voir `http://www.osf.org/~loverso/javascript`.
Corrigés partiellement dans 2.02, totalement (?) dans 3.0PR2.

- HTTP upload. Chargement d'un fichier présent sur le disque local. L'utilisateur doit cliquer sur un bouton, mais le label peut être modifié pour apparaître innocent
- lecture de l'arborescence
- capture de tous les URL visités. Nécessite une interaction de l'utilisateur, mais il est facile de la déguiser de façon à la rendre innocente.

Les bugs liés à l'envoi de mail de la part de l'utilisateur contre sa volonté auraient été corrigés. De même pour l'attaque par `gopher://localhost:25/EHLOxxxx`

sécurité à plusieurs niveaux

Comment rendre l'ensemble du système sûr, alors que des parties de programme peuvent être importées via le réseau, depuis des sites à priori inconnus, et peut-être hostiles ?

La sécurité est implémentée par tous ces niveaux :

1. sources et spécifications publiques
2. langage et compilateur
3. *byte code verifier*
4. *ClassLoader*
5. entrées / sorties disques et réseau

langage et compilateur

- typage fort
- contrôle d'accès (`private, protected, final`)
- pas de conversions implicites
- interdiction de définir des opérateurs de conversion
- pas d'arithmétique sur les pointeurs
- gestion de la mémoire
- pas de surcharge des opérateurs

byte code verifier

On ne peut être sûr que le compilateur a bien respecté les spécifications. Comme le code peut être importé depuis le réseau, il est nécessaire de faire un certain nombre de vérifications avant exécution. En général, cette phase de vérification n'est faite que pour les programmes venant de l'extérieur.

Une fois les vérifications effectuées, le code est reputé sûr et exécuté sans contrôle par l'interpréteur (ou le CPU Java).

Le *byte code verifier* n'est pas lié au langage Java.

- vérification du format du fichier `.class`
  - informations sur la classe (nom, superclasse)
  - informations sur les méthodes et les attributs. Pour les méthode, ces informations incluent le code (`byte code` ou codes pour des CPU particuliers) et les handlers d'exception
  - zones des constantes

- signature (optionnel)

- Il faut s'assurer que le code :
  - ne contrefait pas de pointeurs
  - ne viole pas les restrictions d'accès
  - utilise les objets pour ce qu'ils sont
  - ne fait pas de conversions illégales
  - appelle les méthodes avec le bon nombre d'arguments et le bon type
  - qu'il n'y a pas de débordement de la pile

La machine virtuelle Java

Elle définie 6 types de bases :

integers 32 bits

longs ou long integers 64 bits

single floats 32 bits FP

double floats 64 bits FP

handles pointeurs sur objets et tableaux

return address pointeurs sur code

et des types tableaux (integers, longs, single float, double floats, handle, booleans, bytes, shorts, caractères Unicode).

- c'est une machine CISC
- instructions de longueur fixe (sauf "table lookup"), opcode sur un octet
- les opérations s'effectuent sur la pile
- les éléments de la pile sont typés mais non taggés
- les instructions dépendent du type de données (aload, iload,…)
- les registres sont utilisés pour stocker les variables locales.

Conséquence :

> *Connaissant le type avant l'exécution d'une instruction, on connait le type après l'exécution*

Autre contrainte :

> *quand il existe deux chemins d'exécution pour arriver à un point, les chemins doivent y arriver dans le même "type state"*

Résultat :

- le code ainsi vérifié peut être exécuté par l'interpréteur ou le CPU sans autre contrôle
- Les informations de types ne sont plus utiles dans la phase d'exécution

```
class vector{
  int arr[];
  int sum(){
    int la[] = arr;
    int S = 0;
    for (int i = la.length; --i >= 0;)
      S += la[i];
    return S;
  }
}
```



```
   aload_0          ; load this
   getfield #10     ; load this.arr
   astore_1         ; store in la
   iconst_0
   istore_2         ; store 0 in S
   aload_1          ; load la
   arraylength
   istore_3         ; store in i
A: iinc 3 -1        ; --i
   iload_3          ; load i
   iflt B           ; exit if < 0
   iload_2          ; load S
   aload_1          ; load la
   iload_3          ; load i
   iaload           ; load la[i]
   iadd             ; add in S
   istore_2         ; store to S
   goto A
B: iload_2          ; load S
   ireturn
```

Le ClassLoader

le *ClassLoader* garantit :

- un espace de nom unique pour les classes lues depuis le système de fichier local (classes built-ins)
- un espace de nom unique associé à *chaque* site

Lorsqu'une classe référence une autres classe, l'environnement d'exécution recherche, dans l'ordre :

- dans les built-ins
- dans l'espace de noms de la classe qui référence

$\Downarrow$

- une classe importée ne peut pas se faire passer ("spoof") pour une classe built-in
- une classe built-in ne peut pas accidentellement référencer une classe importée (elle peut le faire explicitement).

restrictions d'accès

- implémentée par le browser par la classe SecurityManager (une seule par browser, non dérivable)
- contrôler l'accès aux ressources (disque, réseau) pour les applets
- éventuellement contrôle d'intégrité et authentification du code reçu (non implémenté)

En général et par défaut, les applets (chargées via le réseaux) ne peuvent pas :

- lire ni écrire de fichier
- ouvrir des connexions de réseau avec une machine autre que celle d'origine de l'applet
- exécuter de programmes sur le client (`Runtime.exec()`), ni provoquer l'arrêt du browser (`System.exit()`)
- charger de librairies, ni définir de méthodes "natives"

exemples de configuration

Ces exemples sont directement issus de la FAQ Applet Security.

Les applets chargées depuis le réseau par NetScape 2.0 ne peuvent lire ni écrire de fichiers.

le programme appletviewer fourni dans le JDK 1.0 de Sun est configurable par des listes d'accès, définies dans le fichier `~/.hotjava/properties` (`F:\.hotjava\properties` si le browser est installé sur le disque `F:` d'un système Win32)

lecture de fichiers
`acl.read=file1:dir1:dir2`

écriture `acl.write=dir:file`

properties NetScape 2.0 et l'appletviewer peuvent lire les properties suivante (ex : `String s = System.getProperty("os.name");`):

key	*meaning*
java.version	Java version number
java.vendor	Java vendor-specific string
java.vendor.url	Java vendor URL
java.class.version	Java class version number
os.name	OS name
os.arch	OS architecture
file.separator	/ ou \
path.separator	/ ou \
line.separator	RET, LF

Les applets ne peuvent pas lire :

key	*meaning*
java.home	Java install directory
java.class.path	Java classpath
user.name	UID
user.home	home directory
user.dir	current working dir

masquer des properties Impossible avec NetScape 2.0. Pour l'appletviewer : `os.name=null`

autoriser des properties interdites Impossible avec NetScape 2.0. Pour l'appletviewer : `user.name.applet=true`

connexions de réseau possibles uniquement vers la machine d'où provient la page `html`, ou vers la machine spécifiée dans le paramètre `codebase` du tag `applet`. Ce dernier est prioritaire.

L'autorisation est basée sur l'adresse IP de la machine ayant fourni le code. Si la machine possède plusieurs adresses IP (vu du DNS), seule l'adresse IP ayant servi au chargement est autorisée.

récapitulatif

NN NetScape 2.0, applet chargée via le réseau

NL NetScape 2.0, applet chargée localement

AN appletviewer 1.0, via le réseau

AL appletviewer 1.0, localement

JS Java Standalone

	NN	NL	AN	AL	JS
read file acl.read=null	Non	Non	Non	Oui	Oui
read file acl.read=file	Non	Non	Oui	Oui	Oui
write file acl.write=null	Non	Non	Non	Oui	Oui
write file acl.write=file	Non	Non	Oui	Oui	Oui

	NN	NL	AN	AL	JS
get file info acl.read=null acl.write=null	Non	Non	Non	Oui	Oui
get file info acl.read=file acl.write=/tmp	Non	Non	Oui	Oui	Oui
File.delete()	Non	Non	Non	Non	Oui
/usr/bin/rm	Non	Non	Non	Oui	Oui
lire user.name	Non	Oui	Non	Oui	Oui
cnx rx sur client	Non	Oui	Non	Oui	Oui
cnx rx autre machine	Non	Oui	Non	Oui	Oui
load library	Non	Oui	Non	Oui	Oui
exit(-1)	Non	Non	Non	Oui	Oui
créer une fenêtre non signalée	Non	Oui	Non	Oui	Oui

attaques connues de Java

L'environnement Java se veut sûr, aux erreurs d'implémentations près. Hélas, les mécanismes de sécurité reposent sur des sous-systèmes (langage, byte-code) non formellement spécifiés.

SecurityManager bug, Avril 96 (Just Released). Le code de l'implémentation de la classe SecurityManager de l'appletviewer ne teste pas que le nom du fichier à ouvrir contient `..`. Voir `http://www.infobiogen.fr/people/claude/java/secbug.html`

URL name resolution attack, Avril 96 Une applet chargée depuis l'extérieur sur un client derrière un firewall peut contacter une machine derrière le firewall. Ne fonctionne que dans des configurations extrèmement particulières où le domaines attaqué n'est pas officiellement déclaré auprès de l'InterNIC alors que l'attaquant possède ce nom régulièrement

enregistré. Voir `http://java.sun.com/sfaq/dns-spoof2.html`.

Verifier implementation bug, Mars 96 `http://www.cs.princeton.edu/~ddean/java`

ClassLoader implementation bug, Mars 96 possibilté de charger une classe en violation des règles de sécurité. Corrigé dans NetScape 2.01 et dans le JDK 1.0.1

DNS attack, Février 96 Le DNS d'un attaquant annonce que sa machine a deux adresses IP, la vraie et celle de sa cible. L'applet (sur le client) tente d'ouvrir une connexion de réseau avec la cible. La requète est acceptée car l'adresse de la cible est une des adresses (fausse) de la source de l'applet. Ce bug est dû en partie à l'absence d'authentification dans le DNS. Corrigé dans NetScape 2.01 et dans le JDK 1.0.1, en restreignant les possibilités d'ouverture de connexions à l'adresse IP de la source (pourquoi ne pas tester que gethostbyaddr() est

égal au hostname de l'URL ?). Voir
`http://www.cs.princeton.edu/~ddean/java`

dénis de service Le code d'une applet prend un verrou sur un objet partagé avec le browser. Le browser est bloqué si l'applet ne libère pas le verrou.

canaux secrets "Covert Channels". Une applet peut utiliser des canaux secrets pour communiquer avec le monde extérieur (SMTP, DNS). La méthode `accept()` est insuffisamment protégée dans le JDK.

pointeurs, crédits

- Frequently Asked Questions - Applet Security
  `http://www.javasoft.com/sfaq`
- Low Level Security in Java, Franck Yellin
  `http://www.javasoft.com/sfaq/verifier.html`
- HotJava: The Security Story

  `http://java.sun.com/1.0alpha3/doc/security/security.html`
- The Java Virtual Machine Specification
- Java Security: From HotJava to Netscape and Beyond, Drew Dean, Ed Felten, Dan Wallach, Princeton,
  `http://www.cs.princeton.edu/sip/pub/secure96.html`