Université Paris Diderot
L3 informatique
Sécurité 2017-2018

Sécurité en Java

Les exercices 1 et 2 illustrent la différence entre la politique de sécurité standard de Java entre application locale et distante (les applications locales ne sont pas soumises à la vérification du gestionnaire de sécurité).

Exercice 1.—
En utilisant les méthodes `getProperty()` et `getProperties()`de la classe `java.lang.System` écrire une application `MaSec` permettant d'afficher:

1. La version de la machine virtuelle java utilisée
2. Le répertoire où est installée la machine virtuelle java
3. Chemin (liste de répertoires racines) utilisé pour trouver les répertoires et archives JAR contenant les fichiers class (bytecode)
4. Le nom de l'utilisateur
5. Le répertoire de connexion (home directory)
6. Le répertoire de travail
7. Les séparateurs de fichiers (dans une référence), les séparateurs de répertoires (dans un chemin), les séparateur de ligne
8. La première ligne d'un fichier dont la référence est donnée en argument

Créer un fichier `texte` dans le même repertoire que votre fichier ByteCode et un fichier `texte2` dans le répertoire père.
Executer votre application avec ces deux fichiers.

Exercice 2.— Transformer votre application en applet. Pouvez vous afficher les mêmes informations? (Des mécanismes de sécurité peuvent aussi avoir été rajoutés par l'OS, le navigateur...)

Les exercices 3 et 4 permettent de forcer la politique de sécurité de Java.

Exercice 3.— Exécuter votre application avec l'option `java -Djava.security.manager MaSec texte`

Exercice 4.— Vous allez créer deux politiques extrêmes. La première où rien n'est autorisée la deuxième où tout est autorisée. Créer un fichier vide de nom `mapolvide.policy` et un autre fichier de nom `mapol.policy` contenant

```
grant
{
    permission java.security.AllPermission;
};
```

Executer
`java -Djava.security.manager -Djava.security.policy=`*politique* `MaSec` *fichier*
pour les politiques `mapolvide.policy` et `mapol.policy` et les fichiers `texte` et `../text2`

Exercice 5.— L'outil `policytool` permet de créer un fichier de permission. Créer un fichier de permission `mapolplus.policy` donnant pour java.util.PropertyPermission les permissions "read, write". Vous devez obtenir par exemple si `ref` est le catalogue où se trouve votre fichier class.

```
grant codeBase "file:ref" {
  permission java.util.PropertyPermission "*", "read,write";
};
```

Exécuter
`java -Djava.security.manager -Djava.security.policy=mapolplus.policy MaSec` *fichier*
pour les fichiers `texte` et `../text2` .
Modifier le fichier `mapolplus.policy` pour que l'application puisse accéder au fichier `../text2`

Exercice 6.— Le programme suivant permet de connaitre la politique de sécurité de java pour accéder aux propriétés:

```
import java.io.*;
import java.security.*;

public class testPerm {
    public static void main( String[] vv){

        //get currentsecuritymanager
    SecurityManager sm = System.getSecurityManager();

    if ( sm==null)
        System.out.println("PAS DE SECURITY MANAGER");
    else{

     try{ sm.checkPropertiesAccess();
 System.out.println("on peut lire ou modifier  les proprietes!");}
     catch (SecurityException exception)
        {
        System.out.println ("pas acces : " + exception.getMessage());
        }
     }
    }
}
```

Modifier ce programme pour connaitre les accès de l'application par rapport à un fichier donnée en argument. Tester votre application avec un SecurityManager et les différentes politiques vues dans les exercices précédents.

Exercice 7.— Créer votre propre SecurityManager `monSecurity` qui demandera un mot de passe à toute application demandant à connaitre les permissions d'accès sur les fichiers et les propriétés.
Vous pourrez le tester par
`java -Djava.security.manager=monSecurity -Djava.security.policy=`*politique appli param*