Programmation Réseau

Sécurité Java

Jean-Baptiste.Yunes@univ-paris-diderot.fr

UFR Informatique

2012-2013

Sécurité ?

- différentes sécurités disponibles et contrôlables
  - intégrité
  - contrôle d'accès
  - signature/authentification/cryptographie

Le contrôle d'accès

- `SecurityManager`

  - une classe avec différentes méthodes `check*(...)` qui sont utilisées par la JVM lorsque l'accès correspondant est demandé sur un objet décrit par le(s) paramètre(s)

  - dans le cas des applications « standards » pas de `SecurityManager` par défaut

  - dans le cas des Applets et autres, des `SecurityManager` adaptés sont automatiquement installés

```
import java.io.*;

public class Test {
  public static void main(String []args) {
    try {
      System.out.println("SM="+System.getSecurityManager());
      System.out.println("java.home="+
                                    System.getProperty("java.home"));
      FileInputStream fis = new FileInputStream("../in.txt");
      System.out.println("in.txt read granted");
      FileOutputStream fos = new FileOutputStream("out.txt");
      System.out.println("out.txt write granted");
      int b;
      while ((b = fis.read()) != -1) fos.write(b);
      fis.close();
      fos.close();
    } catch (Exception e) {
      System.err.println("Something's wrong...");
      e.printStackTrace();
    }
  }
}
```

SecurityManager par défaut

- le gestionnaire de sécurité par défaut interdit tout
  - sauf les accès en lecture pour les fichiers contenus dans le répertoire ou les sous-répertoires du CodeBase!!!!
  - Ces accès sont TOUJOURS permis...

Politique de sécurité

- Les gestionnaires standards acceptent la définition d'une politique de sécurité via une spécification

  - fichier (en général d'extension `.policy`) pouvant être généré via l'outil `policytool`

Policy files

- On y trouve des directives comme

```
grant {
  permission something,value;
};
```

- par exemple

```
grant {
  permission java.net.SocketPermission "*:1024-","accept";
};
```

- rechercher Permissions in the Java Development Kit

```
grant {
   permission java.io.FilePermission "../*","read";
   permission java.io.FilePermission "out.txt","write";
   permission java.util.PropertyPermission "java.home","read";
};
```

Contrôle de la sécurité au lancement...

- `java classe`
  - exécution sans sécurité
- `java -Djava.security.manager classe`
  - exécution avec sécurité par défaut
- `java -Djava.security.manager -Djava.security.policy=policyfile classe`
  - exécution avec politique de sécurité

Contrôle de la sécurité dans le code...

- Construire sa propre sous-classe de `SecurityManager`

  - en choisissant les bonnes méthodes à redéfinir

- rechercher Deciding what securitymanager methods to override

Paranoid

```
import java.util.*;
import java.io.*;

public class ParanoidManager
             extends SecurityManager {
  private String password;
  private Scanner scan;
  public ParanoidManager() {
    password = "paranoid";
    scan = new Scanner(System.in);
  }
  private boolean verify() {
    System.out.print("Give me the word ");
    return scan.nextLine().equals(password);
  }
…
```

Paranoid

```
…
  public void checkRead(FileDescriptor filedescriptor) {
    System.out.println("read "+filedescriptor);
    if (!verify())
      throw new SecurityException("Raté");
  }
  public void checkRead(String filename) {
    System.out.println("read "+filename);
    if (filename.endsWith(".txt")) {
        if (!verify()) {
          super.checkRead(filename);
          throw new SecurityException("Raté");
        }
      }
  }
  public void checkRead(String filename, Object executionContext) {
    System.out.println("read "+filename+" "+executionContext);
    if (!verify())
      throw new SecurityException("Raté");
  }
…
```

Paranoid

```
…
  public void checkWrite(FileDescriptor filedescriptor) {
    System.out.println("write "+filedescriptor);
    if (!verify())
      throw new SecurityException("Raté");
  }
  public void checkWrite(String filename) {
    System.out.println("write "+filename);
    if (!verify())
      throw new SecurityException("Raté");
    }
}
```

OneTime

```
public class OneTimeManager extends SecurityManager {
  private String password;
  private Scanner scan;
  private boolean checked;
  public OneTimeManager() {
    password = "onetime";
    scan = new Scanner(System.in);
    checked = false;
  }
  private boolean verify() {
    if (!checked) {
      System.out.print("Give me the word ");
      checked = scan.nextLine().equals(password);
    }
    return checked;
  }
…
```