



Ecole d'ingénieurs et d'architectes de Fribourg  
Hochschule für Technik und Architektur Freiburg

---

# RMI TP03

---

## TAPR2 - Rapport

Filière informatique

Classe I3

*Auteurs :*

Jocelyn Duc  
jocelyn.duc@edu.hefr.ch

Andrea Marcacci  
andrea.marcacci@edu.hefr.ch

*Enseignant :*

Dr Rudolf Scheurer  
rudolf.scheurer@hefr.ch

Fribourg, 10 décembre 2012

## Table des matières

P1	Involved machines . . . . .	2
P2	Server object . . . . .	2
P3	RMI registry directory . . . . .	3
P4	RMI registry in Java . . . . .	4
P5	Captured frames . . . . .	4
P6	RMI registry response . . . . .	6
P7	logCalls directive . . . . .	6
P8	Phase 2 capture . . . . .	7
P9	Successful authentication . . . . .	8
P10	Wrong username . . . . .	8
P11	getDate2() . . . . .	9
P12	Security manager . . . . .	9
P13	Policy file . . . . .	10
P14	Measure . . . . .	10
P15	Location . . . . .	11

## P1 Involved machines

*Note all relevant information about the two involved machines used for your measurements (operating system, JVM versions, IP addresses, etc.).*

### Serveur

```
Mac OSX 10.7.5
java version "1.7.0_09"
Java(TM) SE Runtime Environment (build 1.7.0_09-b05)
Java HotSpot(TM) 64-Bit Server VM (build 23.5-b02, mixed mode)
IP: 160.98.113.62
```

### Client

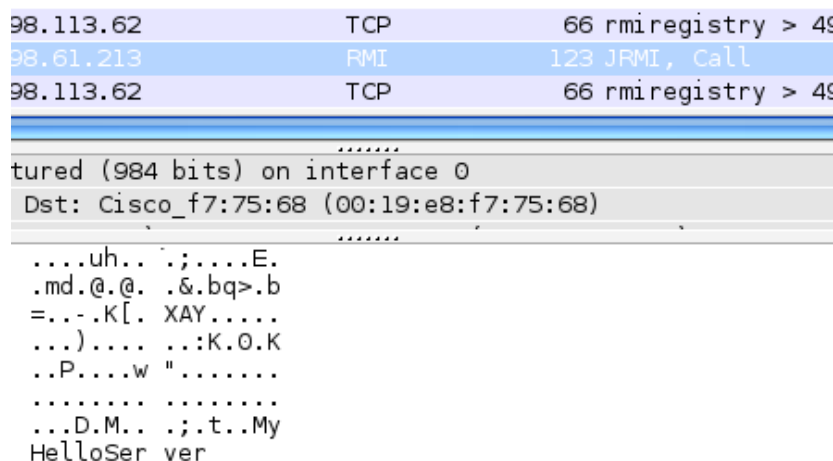
```
Mac OSX 10.7.5
java version "1.7.0_09"
Java(TM) SE Runtime Environment (build 1.7.0_09-b05)
Java HotSpot(TM) 64-Bit Server VM (build 23.5-b02, mixed mode)
IP: 160.98.61.213
```

## P2 Server object

*The server object is registered/published under which name ?*

L'object serveur est enregistré sous le nom de `MyHelloServer`. On peut le voir au moment du binding grâce à la méthode `rebind()` côté serveur mais aussi en paramètre de la méthode `lookup()` du côté du client.

Le nom de `MyHelloServer` est également visible dans la trame d'appel RMI. La capture complète est disponible dans le fichier `p2.cap`.



```

160.98.113.62      TCP      66 rmiregistry > 48
160.98.61.213     RMI      123 JRMID, Call
160.98.113.62      TCP      66 rmiregistry > 48
.....
Captured (984 bits) on interface 0
  Dst: Cisco_f7:75:68 (00:19:e8:f7:75:68)
.....
....uh.. .;.E.
.md.@.@. .&.bq>.b
=...K[. XAY.....
...).... .:K.O.K
..P....w ".....
.....
...D.M.. .;.t..My
HelloSer ver

```

FIGURE 1 – Trame RMI

## HelloServer.java

```
1 HelloServer obj = new HelloServer();
2 System.out.println("\nContacting registry on localhost ...");
3 Naming.rebind("//localhost/MyHelloServer", obj);
```

## HelloClient.java

```
1 System.out.println("Performing lookup ...");
2 Hello obj = (Hello) Naming.lookup("//"+args[0]+"/MyHelloServer");
```

## P3 RMI registry directory

*Why do we have to start the rmiregistry from a very specific directory to make it work? Why is it this one? Explain in detail the exception that occurs if the rmiregistry is started from the wrong directory.*

Le RMI registry est un registre qui liste les objets distants. L'idée est de mapper des objets Java avec des URL qui permettent d'y accéder. Pour le faire, nous utiliserons la commande `rebind()`. Le RMI registry utilise le port 1099 et de ce fait, une deuxième instance ne peut être lancée sur ce même port.

La commande `rmiregistry` doit être lancée depuis le dossier `bin`.

## Lancement de rmiregistry

```
1 cd /Users/Jocelyn/EIAFR/workspaces/eclipse/TAPR_TP03/bin/
2 rmiregistry &
3 [1] 9023
```

Le `rmiregistry` peut être fermé et son port libéré en utilisant la commande `kill` pour tuer le processus.

## Fermeture de rmiregistry

```
1 kill 9023
```

Si nous lançons la commande `rmiregistry` depuis un mauvais dossier, nous obtenons une exception au moment de l'exécution de la méthode `rebind()`. L'exception vient du fait que `rmiregistry` ne peut pas mapper des objets qu'il n'est pas capable de trouver sur le serveur, d'où l'importance de le lancer de la base du projet.

## Lancement de rmiregistry dans le mauvais répertoire

```
1 cd /Users/Jocelyn/EIAFR/workspaces/eclipse/TAPR_TP03/
2 rmiregistry &
3 [1] 9136
```

L'exception suivante est levée. La totalité de cette dernière se trouve dans le fichier joint `p3-exception.txt`.

## Sortie console

```
1 Contacting registry on localhost ...
2 HelloServer exception: RemoteException occurred in server thread;
  nested exception is:
3   java.rmi.UnmarshalException: error unmarshalling arguments;
    nested exception is:
4   java.lang.ClassNotFoundException: rmi.hello.Hello
5   ...
```

## P4 RMI registry in Java

*It is possible to start an RMI registry directly in the Java code, inside the main method on server side. Find out how to do this and then modify your code accordingly.*

Oui, il est possible de le faire directement en Java. Nous avons utilisé le code suivant pour le faire, du côté serveur.

## HelloServer.java

```
1 public static void main(String args[]) {
2     try {
3         Registry r = LocateRegistry.createRegistry(1099);
4         HelloServer obj = new HelloServer();
5         System.out.println("\nContacting registry on localhost ...");
6         Naming.rebind("//localhost/MyHelloServer", obj);
7         System.out.println("HelloServer bound in registry. Stop server
          using Ctrl-C.");
8     } catch (Exception e) {
9         System.out.println("HelloServer exception: " + e.getMessage());
10        e.printStackTrace();
11    }
12 }
```

## P5 Captured frames

*Which are the captured frames related to phases 3, 4, 6 and 9 (see related RMI slide 36) ? Document and explain each phase individually.*

La capture complète est disponible dans le fichier p5.cap.

**Phase 3 : The client object performs a lookup on the registry.**

Le client fait son lookup à la trame 83. Il s'agit une trame de type JRMI Call.

**Phase 4 : The registry returns the serialized stub instance annotated with some information how to contact the skeleton.**

Le client fait reçoit le stub à la trame 85. Il s'agit d'une trame de type JRMI ReturnData.

**Phase 6 : The stub marshalls the parameters and communicates with the skeleton**

Le stub contacte le skeleton à la trame 99. Il s'agit une trame de type JRMI Call.

**Phase 9 : The skeleton marshals the return value and sends it back to the stub.**

Le stub reçoit la réponse du skeleton à la trame 113. Il s'agit d'une trame de type JRMI ReturnData.

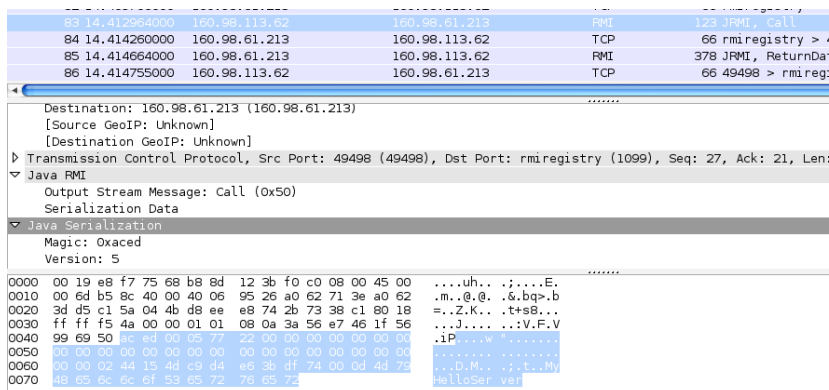


FIGURE 2 – Phase 3 : Lookup

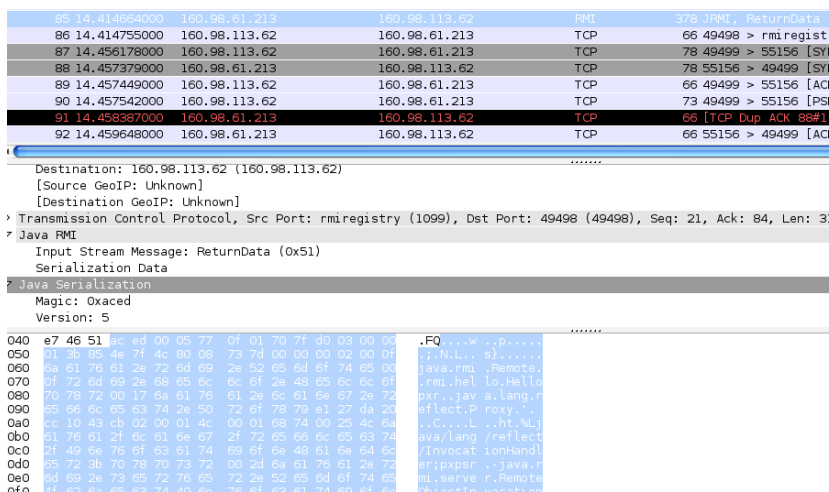


FIGURE 3 – Phase 4 : Retour du stub

99	14.466470000	160.98.61.213	160.98.113.62	TCP	553 55156 > 49499 [PSH, ACK]
100	14.466500000	160.98.113.62	160.98.61.213	TCP	66 49499 > 55156 [ACK] Seq=
101	14.469888000	160.98.113.62	160.98.61.213	RMI	67 JRMIPing
102	14.474122000	160.98.61.213	160.98.113.62	TCP	66 rmiregistry > 49498 [ACK]
103	14.474161000	160.98.61.213	160.98.113.62	RMI	67 JRMIPingAck
104	14.474203000	160.98.113.62	160.98.61.213	TCP	66 49498 > rmiregistry [ACK]

```

> Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 62
  Protocol: TCP (6)
  > Header checksum: 0xedb8 [correct]
    Source: 160.98.61.213 (160.98.61.213)
    Destination: 160.98.113.62 (160.98.113.62)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  > Transmission Control Protocol, Src Port: 55156 (55156), Dst Port: 49499 (49499), Seq: 21, Ack: 478, Len: 287
    000  b8 8d 12 3b f0 c0 00 19 e8 f7 75 68 08 00 45 00  ...:....uh..E.
    010  01 53 5e 14 40 00 3e 06 ed b8 a0 62 3d d5 a0 62  .S^@>. .b=.b
    020  71 3e d7 74 c1 5b 28 f0 15 dd 68 86 9c f0 80 18  q>.t.[(. .h....
    030  ff ff d6 c2 00 00 01 01 08 0a 1f 56 99 9f 3a 56  .....V.:V
    040  e7 76 51 ac ed 00 05 77 0f 01 70 7f d0 03 00 00  .VQ...w..p....
    050  01 3b 85 4e 7f 4c 80 09 73 72 00 12 6a 61 76 61  .:N.L...sr..java
    060  2e 72 6d 69 2e 64 67 63 2e 4c 65 61 73 65 b0 b5  .rmi.dgc .Lease..
    070  e2 66 0c 4a dc 34 02 00 02 4a 00 05 76 61 6c 75  .f.J.4...J..valu
    080  65 4c 00 04 76 6d 69 64 74 00 13 4c 6a 61 76 61  eL..vmid t..L.java
    090  2f 72 6d 69 2f 64 67 63 2f 56 4d 49 44 3b 70 78  /rmi/dgc /VMID;px
    0a0  70 00 00 00 00 09 27 c0 73 72 00 11 6a 61 76  p.....'.sr..jav

```

FIGURE 4 – Phase 6 : Communication du stub vers le skeleton

113	14.479169000	160.98.61.213	160.98.113.62	TCP	101 55156 > 49499 [PSH]
114	14.479217000	160.98.113.62	160.98.61.213	TCP	66 49499 > 55156 [ACK]
115	14.485518000	160.98.113.62	160.98.61.213	TCP	66 49499 > 55156 [FIN]
116	14.485585000	160.98.113.62	160.98.61.213	TCP	66 49498 > rmiregistr

```

> Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 62
  Protocol: TCP (6)
  > Header checksum: 0x2b56 [correct]
    Source: 160.98.61.213 (160.98.61.213)
    Destination: 160.98.113.62 (160.98.113.62)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  > Transmission Control Protocol, Src Port: 55156 (55156), Dst Port: 49499 (49499), Seq: 309, Ack: 520, Len: 35
    00  b8 8d 12 3b f0 c0 00 19 e8 f7 75 68 08 00 45 00  ...:....uh..E.
    01  00 57 21 73 40 00 3e 06 2b 56 a0 62 3d d5 a0 62  .Wis@>. +V.b=.b
    02  71 3e d7 74 c1 5b 28 f0 16 fd 68 86 9d 1a 80 18  q>.t.[(. .h....
    03  ff ff 85 94 00 00 01 01 08 0a 1f 56 99 aa 3a 56  .....V.:V
    04  e7 81 51 ac ed 00 05 77 0f 01 70 7f d0 03 00 00  .:Q...w..p....
    05  01 3b 85 4e 7f 4c 80 0a 74 00 0a 48 69 20 46 6f  .:N.L...t..Hi Fo
    06  6c 6b 73 21 21  lks!!

```

FIGURE 5 – Phase 9 : Communication du skeleton vers le stub

## P6 RMI registry response

*Especially examine the response of the rmiregistry on the client's lookup request (Phase 4). Try to identify the different elements inside this response, e.g. how the client's stub knows on which IP and port address the skeleton may be contacted.*

Conformément à la capture p5.cap, nous pouvons constater que dans la réponse, nous retrouvons des informations telles que l'adresse IP et le port sur lesquelles on peut contacter le skeleton.

## P7 logCalls directive

*Redo the whole process (without Wireshark measurement), this time starting the server object with the JVM directive -Djava.rmi.server.logCalls=true. Document and comment the console output on server side.*

L'utilité de logCalls est de faciliter le debugging de l'application. Grâce à cet argument, les informations entrantes sont loguées dans System.err.

Association d'un nom avec un objet dans le registre (**rebind**)

#### Sortie console

```
1 10-dic-2012 16.38.55 sun.rmi.server.UnicastServerRef logCall
2 MOLTO FINE: RMI TCP Connection(1)-127.0.0.1: [127.0.0.1: sun.rmi.
  registry.RegistryImpl[0:0:0, 0]: void rebind(java.lang.String,
  java.rmi.Remote)]
```

Annnonce de la disponibilité temporelle de l'objet distant dans le registre.

#### Sortie console

```
1 10-dic-2012 16.38.55 sun.rmi.server.UnicastServerRef logCall
2 MOLTO FINE: RMI TCP Connection(2)-160.98.61.213: [160.98.61.213:
  sun.rmi.transport.DGCImpl[0:0:0, 2]: java.rmi.dgc.Lease dirty(
  java.rmi.server.ObjID[], long, java.rmi.dgc.Lease)]
3 HelloServer bound in registry. Stop server using Ctrl-C.
```

Retourne l'objet distant vers le client (**lookup**).

#### Sortie console

```
1 10-dic-2012 16.38.58 sun.rmi.server.UnicastServerRef logCall
2 MOLTO FINE: RMI TCP Connection(3)-160.98.113.62: [160.98.113.62:
  sun.rmi.registry.RegistryImpl[0:0:0, 0]: java.rmi.Remote lookup(
  java.lang.String)]
```

Annnonce de la disponibilité temporelle de l'objet distant dans le registre.

#### Sortie console

```
1 10-dic-2012 16.38.58 sun.rmi.server.UnicastServerRef logCall
2 MOLTO FINE: RMI TCP Connection(4)-160.98.113.62: [160.98.113.62:
  sun.rmi.transport.DGCImpl[0:0:0, 2]: java.rmi.dgc.Lease dirty(
  java.rmi.server.ObjID[], long, java.rmi.dgc.Lease)]
3 - saying hello to the world ...
```

Appel vers la méthode de l'objet distant (**sayHello()**).

#### Sortie console

```
1 10-dic-2012 16.38.58 sun.rmi.server.UnicastServerRef logCall
2 MOLTO FINE: RMI TCP Connection(4)-160.98.113.62: [160.98.113.62:
  rmi.hello.HelloServer[72309918:13b85780a09:-7fff,
  -2837820585922389599]: public abstract java.lang.String rmi.
  hello.Hello.sayHello() throws java.rmi.RemoteException]
```

## P8 Phase 2 capture

*[optional] Try to capture the communication between server object and the RMI registry (phase 2). Document and explain your measurement.*



## P9 Successful authentication

*Document a successful authentication process by means of a Wireshark measurement (screenshot of relevant frames) of the whole communication.*

Premièrement, nous avons demandé des accès sur le serveur 160.98.31.52. Le mot de passe password est utilisé pour notre utilisateur DucMarcacci.

### Génération du mot de passe en MD5

```
1 md5 -s password
2 MD5 ("password") = 5f4dcc3b5aa765d61d8327deb882cf99
```

Il est demandé de créer une classe `RMIDigestClient` permettant de s'authentifier avec succès auprès du serveur distant. Voici une implémentation basique qui permet de le faire.

### RMIDigestClient.java

```
1 public static void main(String[] args) {
2     try {
3         RMIDigestValidator obj = (RMIDigestValidator) Naming.lookup("//
4             + SERVER + "/" + CLASS);
5         String challenge = obj.getChallenge(USERNAME);
6         MD5Digest md5 = new MD5Digest();
7         boolean result = obj.challengeResponse(USERNAME,
8             md5.doHash(md5.doHash(PASSWORD), challenge.getBytes()));
9         System.out.println("Challenge: " + (result ? "Success" : "Fail"
10             ));
11     } catch (Exception e) {
12         e.printStackTrace();
13     }
14 }
```

Selon notre implémentation, à ce stade, à sortie est la suivante.

### Sortie console

```
1 Challenge: Success
```

## P10 Wrong username

*What happens if you use a wrong username? Document and explain the exception that occurs.*

Si nous spécifions un nom d'utilisateur incorrect, une exception est propagée. Il s'agit d'une exception `RemoteException`. Elle a été lancée sur le serveur puis propagée chez le client. L'exception contient un message qui nous indique clairement que le nom d'utilisateur spécifié n'existe pas.

**Sortie console**

```
1 Caused by: java.rmi.RemoteException: getChallenge failed: unknown
  username 'BadUsername'!
2     ...
```

La totalité de cette exception se trouve dans le fichier joint `p10-exception.txt`.

Par contre, si l'on spécifie un nom d'utilisateur existant avec un mauvais mot de passe, aucune exception n'est lancée mais le challenge est refusé.

**Sortie console**

```
1 Challenge: Fail
```

## P11 getDate2()

*Document and explain what happens when executing getDate2().*

L'exception suivante est lancée. On voit qu'elle est de type `ClassNotFoundException`.

**Sortie console**

```
1 Challenge: Success
2 date1:Tue Dec 04 10:03:44 CET 2012
3 java.rmi.UnmarshalException: error unmarshalling return; nested
  exception is:
4   java.lang.ClassNotFoundException: rmi.digest.MyDate (no security
    manager: RMI class loader disabled)
```

Cette exception est lancée pour la simple est bonne raison que le serveur retourne un objet de type `MyDate` qui étend la classe `Date` et que le client ne possède pas cette classe `MyDate`.

## P12 Security manager

*Use the RMI security manager in your client code and try again. Again, document and explain what happens.*

Nous avons activé le `RMISecurityManager` à l'aide des lignes suivantes :

**RMIDigestClient.java**

```
1 if (System.getSecurityManager() == null) {
2     System.setSecurityManager(new RMISecurityManager());
3 }
```

A l'exécution, nous obtenons tout de même une exception liée à la sécurité. C'est parce que le `RMISecurityManager` n'a pas reçu de fichier de permission (`policy.txt`) en argument.

**Sortie console**

```
1 java.security.AccessControlException: access denied (java.net.  
   SocketPermission 160.98.31.52:1099 connect,resolve)  
2   ...
```

La totalité de cette exception se trouve dans le fichier joint `p12-exception.txt`.

## P13 Policy file

*Use the policy file as provided in the zipped sources. Explain why it works now.*

Le fichier `policy.txt` contient des polices de sécurité. Son contenu est le suivant :

**policy.txt**

```
1 grant {  
2   // WARNING: for testing purposes only: allow everything for now  
3   permission java.security.AllPermission;  
4 };
```

**Directive de sécurité**

```
1 -Djava.security.policy=/Users/Jocelyn/EIAFR/workspaces/eclipse/  
   TAPR_TP03/src/rmi/digest/policy.txt
```

La récupération de `MyDate` se fait sans problème dès que le fichier de police est passé en argument de la JVM. La sortie est donc la suivante :

**Sortie console**

```
1 Challenge: Success  
2 date1:Mon Dec 10 18:26:45 CET 2012  
3 date2:10. decembre 2012, 18:26:45 [on DigestValidator]
```

## P14 Measure

*Use Wireshark to measure the traffic while using your client. Document and explain the HTTP communication within this measurement.*

La capture se trouve dans le fichier joint `p14.cap`.

Concernant le trafic enregistré, en résumant, nous constatons ceci :

- Le client fait un lookup de la classe `RMIDigestServer` auprès de `160.98.31.52`
- Le client fait un appel sur l'objet distant et remarque qu'il lui manque une classe
- Le serveur l'informe que la classe se trouve sur `http://rudolf.scheurer.home.hefr.ch/java/rmi/digest/M`
- Le client télécharge la classe au moyen d'un GET sur HTTP
- Le client peut enfin utiliser la classe téléchargée et continuer son exécution

## P15 Location

*From where does your client know the exact URL (IP/port address) he has to contact ? Try using the JVM directive `-Dsun.rmi.loader.logLevel=VERBOSE` to get more information.*

Grâce au mode verbeux cité ci-dessus, en plus de l'analyse de trames faites à l'exercice précédent, nous pouvons dire que la classe manquante est téléchargée par le client sur `http://rudolf.scheurer.home.hefr.ch/java/rmi/digest/MyDate.class`.

Le mode verbeux indique que `http://rudolf.scheurer.home.hefr.ch/java/` constitue un `codebase` et concatène à cela, le reste de l'URL, appelée `name`.