Ingénieurs 2000

Projet Corba

Projet

Java Mobile Data Security

Compte rendu

Fiche d'approbation

Version	Date	Rédacteurs			
1.0	17/03/05	J. GUERS	S. GUINCHARD	M. GEFFROY	D. BARDIN

Fiche de révision

Version	Date	Objet	Auteur
0.1	08/03/05	Création du document	M.GEFFROY
0.2	15/03/05	Revue du document	D. BARDIN
1.0	17/03/05	Création de la version finale	D. BARDIN

SOMMAIRE

1. Introduction

Ce document présente le fonctionnement du projet JMDS assurant la sécurité des échanges entre un client et un serveur utilisant la technologie CORBA. Ce système à l'avantage de pouvoir s'interfacer sur n'importe quelle application du moment que l'ORB utilisé pour le client est paramétré pour initialiser les objets à l'aide du ClientORBInitializer du projet et celui utilisé pour le serveur avec le ServerORBInitializer.

Nous allons tout d'abord expliquer les concepts utilisés avec un pointage des écarts fonctionnels avec le cahier des charges. Puis nous verrons la façon de paramétrer et d'utiliser cet outil de gestion de accès sécurisés.

2. Les concepts

2.1. Authentification

Au départ nous devions générer un service contexte contenant un message crypté contenant l'ID de l'utilisateur ainsi que l'heure d'envoi du message et une clé de session symétrique différente à chaque échange de message. Ce message était crypté à l'aide de la clé publique du serveur initialisée à la création de la carte. Pour la réponse, le message devait être crypté à l'aide de la clé de session.

L'utilisation d'une clé symétrique nous permettait de gagner en rapidité de cryptage et décryptage tout en gardant une sécurité importante puisque la clé était différente à chaque échange.

Malheureusement, il semble compliquer de réaliser des opérations de cryptage en dehors des cartes. Comme nous ne disposions pas de suffisamment de carte, ce concept a été remplacé par une solution plus simple.

Actuellement le service contexte contient un tableau de 5 bits non initialisés (pour la clé de session qui n'a pas été encore implémentée), 10 bits qui contient l'ID de l'utilisateur ainsi que le request ID. L'ensemble est crypté à l'aide d'un OU EXCLUSIF avec une clé symétrique identique pour tout le monde initialisée à la création de la carte.

On ne retrouve donc plus qu'une seule clé dans les fichiers de paramétrages et non plus une par client comme cela était défini dans le cahier des charges.

Lorsque le client ou le serveur reçoit un message, il récupère le service contexte pour le décoder. Ainsi il peut vérifier que le request ID récupérer dans le service contexte décodé et bien celui du message. Si ce n'est pas le cas alors le message sera bloqué avant d'être reçu par le destinataire.

2.2. Gestion des permissions

Une fois l'authentification effectuée reste encore un étape de gestion des permissions du coté serveur. La seule différence avec le cahier des charges vient du fait que la gestion des permissions ne se fait qu'au niveau des objets. Par manque de temps les permissions sur les fonctions des objets n'ont pas était gérées.

La gestion des permissions s'appuie sur deux fichiers de configuration, Security.xml et Rights.xml. Voici ci-dessous un exemple pour chacun d'eux.

- Security.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<jmds>
      <server key="0123456789"/>
   <clients>
      <client id="jmds1" group="grp_1"/>
      <client id="jmds2" group="grp_2"/>
      <client id="jmds3" group="grp_1"/>
   </clients >
</jmds>
```

- Rights.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<jmds>
      <ior id="IOR:0000……20001">
           <group id="grp1"/>
           <group id="grp2"/>
      </ior>
      <ior id="IOR:0000……20002">
           <group id="grp2"/>
      </ior>
      <ior id="IOR:0000……20003">
           <group id="grp1"/>
           <group id="grp3"/>
      </ior>
</jmds>
```

Dans ce fichier de configuration, nous retrouvons la liste exhaustive de toutes les références (IOR) des objets de l'ORB serveur connus au moment du lancement de FindObject. L'accès aux objets s'appuie sur une notion de groupes. Ainsi, pour chaque objet un ou plusieurs groupes sont notés comme ayant accès. Notons que la liaison entre les clients et les groupes se fait à travers le fichier Security.xml.

2.3. Génération de cartes

Chacune des cartes s'appuie sur la même applet facilitant ainsi leurs générations. Le code contenu par ces cartes est donc toujours identique. En revanche, la phase d'initialisation permet d'obtenir une carte nominative. Durant cette phase, l'identifiant du client et la clé commune seront renseignés.

Notons que ces deux informations ne sont accessibles qu'en écriture afin d'assurer leur sécurisation.

3. L'utilisation

3.1. Configuration des accès serveur

La configuration des accès serveur utilise une notion de groupes afin de réduire les redondances. Ainsi, dans le fichier Security.xml, nous allons avoir une liste de client (avec identifiant unique) chacun associé à un groupes (plusieurs client peuvent appartenir au même groupe). Ensuite dans le fichier Rigths.xml, nous allons, pour chacun des objets listés, définir les groupes ayant accès en indiquant l'identifiant du groupe.

Afin de faciliter la création du fichier Rigths.xml, un objet est mis à disposition pour lister l'ensemble des objets disponible sur le server. Cet objet se base sur le service de nommage sur lequel il va se connecte pour parcourir l'arbre de façon récursive. Chacun des objets trouvés sera sauvegardé dans le fichier Rigths.xml.

Par contre l'utilisation d'un tel concept implique l'utilisation d'un serveur persistant. En effet, il convient que les références des objets créés restent identiques d'un lancement du serveur à l'autre pour que le fichier de configuration Rigths.xml reste valide.

Au moment où l'intercepteur serveur reçoit un message il le décrypte à l'aide de sa clé et récupère le client concerné et l'objet auquel il veut accéder. En récupérant la configuration de Security.xml, on peut déterminer le groupe associé au client (ex : groupe1). Une fois le groupe récupérer et grâce au fichier Rigths.xml, il est alors possible de valider ou non l'accès à l'objet.

3.2. Initialisation de la Java Card

Le chargement de l'applet src.de.berlios.jmds.applet.SCApplet.java sur la Java Card doit suivre quelques règles :

- Package AID : 112233445566
- Applet AID : 11223344556677
- Taille de chargement : 1800

Notons qu'une fois l'objet chargé et l'instance activée, il est possible de tester les fonctions de la carte via l'APDU Manager. Afin de faciliter son utilisation, un fichier de définition des trames est disponible à la racine du projet (SCApplet.fcn).

Avant toute utilisation de la carte, il est nécessaire de l'initialiser afin de renseigner les paramètres de cryptage (client_ID et key). Cette initialisation passe par le lancement du programme suivant : src.de.berlios.jmds.applet.InitApplet.java ou par le lancement de trame APDU SetClientID et SetKey.

Une deuxième applet src.de.berlios.jmds.applet.SCAppletWithKey.java utilise les clés et un cryptage natif à la carte. Cette solution n'est pas utilisable du fait d'un décryptage trop complexe en dehors de la carte (sur le serveur). Cette applet est néanmoins fournit puisque fonctionnelle est représentant un nombre important de recherches.

3.3. Lancement du serveur

Avant le lancement du serveur, il est nécessaire de lancer l'orbd afin de pouvoir accéder au service de nommage :

```
orbd –ORBInitialPort 1234
```

La gestion de la sécurité sur le serveur nécessite d'ajouter aux propriétés de l'ORB la prise en compte de l'initialiseur de serveur JMDS.

```
props.put
("org.omg.PortableInterceptor.ORBInitializerClass.de.berlios.jmds.server.ServerO
RBInitializer" , ""); // Prise en compte de l'initialiseur
props.put ("com.sun.CORBA.POA.ORBPersistentServerPort" , "12345");
props.put ("com.sun.CORBA.POA.ORBServerId" , "1");
props.put ("org.omg.CORBA.ORBInitialPort" , "1234");
ORB orb = ORB.init(args, props);
```

Afin de facilité la prise en main du système, un serveur à référence persistante est mis à disposition : src.fr.umlv.ir3.corba.forum.PersitantForumServer.java

3.4. Lancement du client

Comme pour le serveur, le client doit prendre en compte l'initialiseur dans les propriétés de l'ORB afin que la sécurisation soit effective.

```
props.put("org.omg.PortableInterceptor.ORBInitializerClass.de.berlios.jmds.client.Client
ORBInitializer", "");
props.put("org.omg.CORBA.ORBInitialPort" , "1234");
ORB orb = ORB.init(args, props);
```

Là encore, un client d'exemple est mis à disposition : src.fr.imlv.ir3.corba.forum.ForumClient.java

4. Conclusion

Ce travail effectué sur la sécurisation des échanges entre un client et un serveur s'appuyant sur la technologie CORBA a pu nous apprendre beaucoup de choses.

Nous regrettons de ne pas avoir pu pousser le développement aussi loin que ce que nous avions définis dans le cahier des charges. Les problèmes de connexion à la carte mais surtout le manque de documentation ont perturbé l'avancement du projet.

En effet, il est apparu que les outils de développement Java Card ne sont pas encore matures. Ainsi, les multiples réinstallations de drivers nous ont fait perdre beaucoup de temps.

Le manque de documentation sur les intercepteurs a aussi été pénalisant. Excepté quelques exemples simples rapidement dépassés, l'information est presque inexistante.

Mais il s'avère que ce concept d'interception offre des grandes possibilités quand à la gestion des communications. Une fois les concepts assimilés, il devient intéressant d'en exploiter toutes les possibilités.

De la même manière, les applet fournissent de grandes possibilités même si le développement lié est rendu difficile par le manque d'intérêt des communautés sur le sujet et le manque de maturité des produits.