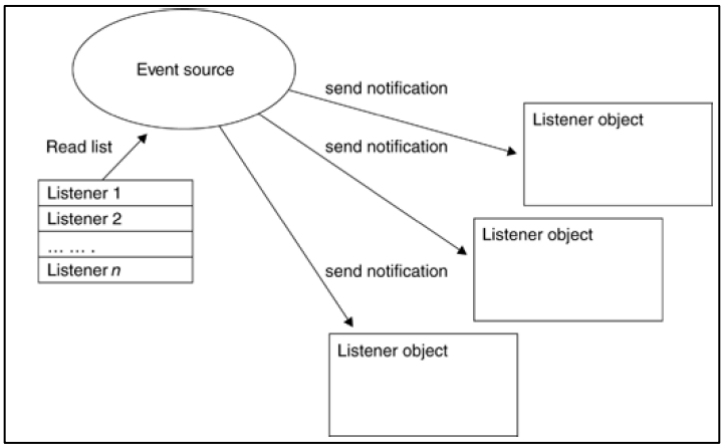
Κατανεμημένα Συστήματα
8ο Φυλλάδιο εργαστηρίου (11/05/2017)

RMI CALLBACKS & ΠΟΛΙΤΙΚΕΣ ΑΣΦΑΛΕΙΑΣ

CALLBACKS

Σε πολλές εφαρμογές είναι χρήσιμο οι συνδεδεμένοι πελάτες να ενημερώνονται για τις αλλαγές που έχουν πραγματοποιηθεί στην κατάσταση του εξυπηρετητή. Μία λύση σε αυτή την περίπτωση θα ήταν, οι πελάτες σε τακτά χρονικά διαστήματα να ρωτούν τον εξυπηρετητή για την κατάσταση του (polling). Αυτή η λύση προφανώς δεν είναι αποδοτική διότι δημιουργεί συνεχώς κίνηση στον εξυπηρετητή χωρίς να είναι σίγουρο τελικά αν υπάρχει αλλαγή ή όχι.

Μια γνωστή λύση σε τέτοιες εφαρμογές είναι η χρήση **κλήσεων επιστροφής** (**callback**) από τον εξυπηρετητή στον πελάτη. Αυτή είναι μια πιο αποδοτική τεχνική διότι ο πελάτης ενημερώνεται μόνο όταν συμβαίνει ένα γεγονός στον εξυπηρετητή. Όπως φαίνεται και από το ακόλουθο σχήμα στο μοντέλο αυτό οι πελάτες (Listeners) εγγράφονται στον εξυπηρετητή, ο οποίος αναλαμβάνει να τους ενημερώνει κάθε φορά που συμβαίνει ένα γεγονός (event).



Η υλοποίηση των callbacks σε JAVA RMI αρχιτεκτονική δεν είναι δύσκολη. Θα πρέπει και ο πελάτης να υλοποιεί ένα RMI Interface δηλαδή να λειτουργεί εξίσου και ως RMI Server. Το Interface που θα υλοποιεί ο πελάτης θα πρέπει να περιέχει μεθόδους που θα καλούνται από τον εξυπηρετητή κάθε φορά που πρέπει να τον ενημερώνει. Επίσης ο εξυπηρετητής θα πρέπει να διαθέτει πρόσθετες μεθόδους στο RMI Interface του με σκοπό την εγγραφή και διαγραφή του πελάτη ως Listener.

Παράδειγμα - Callback

Το ακόλουθο χαρακτηριστικό παράδειγμα υλοποιεί το μοντέλο callback στην εφαρμογή ChatServer που είδαμε σε προηγούμενη ενότητα.

Διαβάστε τα σχόλια και προσθέστε τον κατάλληλο κώδικα όπου σας ζητείται:

```
---------------------- ChatListener Interface ----------------------
// Η RMI διεπαφή που υλοποιεί ο πελάτης
import java.rmi.*;

public interface ChatListener extends Remote {

// Ορίστε το πρωτότυπο της συνάρτησης που θα δέχεται ως παράμετρο
// ένα μήνυμα και θα το τυπώνει στον client (όνομα συνάρτησης:update)
public void update (ChatMessage msg) throws RemoteException;
}


---------------------- ChatInterface Interface ----------------------
// Η RMI διεπαφή που υλοποιεί ο server
import java.rmi.*;

public interface ChatInterface extends Remote{
    public void sendMessage(ChatMessage msg) throws RemoteException;

// Ορίστε το πρωτότυπο των συναρτήσεων joinChat και leaveChat που θα δέχεται
// ως παράμετρο ένα πελάτη και θα τον προσθέτει στην λίστα – register στον
// εξυπηρετητή ή θα τον διαγράφει από την λίστα αντίστοιχα
public void joinChat(ChatListener client) throws RemoteException;
public void leaveChat (ChatListener client) throws RemoteException;
}


---------------------- ChatMessage Class ----------------------
import java.io.Serializable;
import java.util.Date;

public class ChatMessage implements Serializable{
    private String name;
    private String msg;
    private Date date;

 public ChatMessage (String name, String msg){
     this.name=name;
```

```
     this.msg=msg;
     this.date = new Date();
 }

 public String getName(){
  return this.name; }

public String getMessage(){
  return this.msg; }

public Date getDate(){
  return this.date; }

public String toString() {
    return name + " " + date + " " + msg; }
}
```

---------- ChatRMIServer *Class* ----------

```
import java.io.*;
import java.net.MalformedURLException;
import java.rmi.*;
import java.rmi.registry.Registry;
import java.rmi.server.*;
import java.util.ArrayList;

public class ChatRMIServer extends UnicastRemoteObject implements
ChatInterface{
    // Ορίστε την λίστα με τους συνδεδεμένους πελάτες
    private ArrayList<ChatListener> clients;

    public ChatRMIServer() throws RemoteException{
    // Υλοποιήστε τον constructor
    super();
    clients = new ArrayList<ChatListener>();
    }

    public void sendMessage(ChatMessage msg) throws RemoteException {
        try {
            BufferedWriter file = new BufferedWriter(new
FileWriter("ChatMessages.txt",true));
            file.write(msg.getName()+" : "+msg.getDate()+" : "+
msg.getMessage()+"\n");
            file.close();

             } catch (IOException ex) {
```

```
                ex.printStackTrace();
            }

        // Ενημέρωση όλων των συνδεδεμένων πελατών
        // Κλήση της μεθόδου που υλοποιείται στον πελάτη
        for (ChatListener client:clients){

            try{
                client.update(msg);
             } catch (RemoteException ex) {
                 clients.remove(client);
             }
        }


    // Εγγραφή πελάτη στην λίστα του εξυπηρετητή
    public void joinChat(ChatListener client) throws RemoteException {
        System.out.println("Client Connected....");
        clients.add(client);
    }


    // Αποσύνδεση πελάτη από την λίστα του εξυπηρετητή
    public void leaveChat(ChatListener client) throws RemoteException {
        System.out.println("Client Disconnected....");
        clients.remove(client);
    }


    public static void main(String[] args) {

      // RMISecurityManager security = new RMISecurityManager();
      // System.setSecurityManager(security);

        ChatRMIServer server;
        try {
            server = new ChatRMIServer ();
            Naming.rebind("//localhost/ChatRMI", server);
            System.out.println("Waiting new Messages");
        } catch (RemoteException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
           ex.printStackTrace();
        }
    }
}
```

```
---------------------- ChatRMIClient Class ----------------------
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Scanner;

public class ChatRMIClient extends UnicastRemoteObject implements
ChatListener{

    public ChatRMIClient () throws RemoteException {
        super();
    }

    // υλοποίηση της RMI μεθόδου που θα κληθεί από τον εξυπηρετητή
    public void update(ChatMessage msg) throws RemoteException {
        System.out.println(msg);
    }

    public static void main(String[] args) {

     //RMISecurityManager security = new RMISecurityManager();
     //System.setSecurityManager(security);

     Scanner scan=new Scanner(System.in);
     String answer, message;
     try {
           ChatInterface look_up=(ChatInterface)
           Naming.lookup("//localhost/ChatRMI");

           ChatRMIClient client = new ChatRMIClient();

           // εγγραφή του πελάτη στον εξυπηρετητή
           look_up.joinChat(client);

            do {
            System.out.println("Enter your message ?");
            message=scan.nextLine();

            look_up.sendMessage(new ChatMessage("nikos", message));

            System.out.println("Continue y/n ?");
            answer=scan.nextLine();}
```

```
            while (answer.equals("y"));

        } catch (NotBoundException ex) {
            ex.printStackTrace();
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
        } catch (RemoteException ex) {
            ex.printStackTrace();
        }
    }
}
```

Η τεχνική αυτή μπορεί να υλοποιηθεί και με έτοιμες κλάσεις-διεπαφές που παρέχει το JAVA API. Αυτές είναι η κλάση **Observable** και η διεπαφή **Observer**.

ΠΟΛΙΤΙΚΕΣ ΑΣΦΑΛΕΙΑΣ

Θα ασχοληθούμε με την έννοια της πολιτικής ασφάλειας και τον καθορισμό του αρχείου πολιτικής ασφάλειας στα κατανεμημένα συστήματα RMI και θα εξετάσουμε τη δομή αυτών των αρχείων. Η ασφάλεια, για την περίπτωση μας, ουσιαστικά βασίζεται σε τρεις κύριες αρχές:

Εμπιστευτικότητα: Πρόσβαση μόνο σε εξουσιοδοτημένους χρήστες.

Ακεραιότητα: Τροποποίηση των δεδομένων μόνο από εξουσιοδοτημένους χρήστες.

Αυθεντικότητα: Πιστοποίηση της πρόσβασης συγκεκριμένων χρηστών.

Το μοντέλο ασφαλείας της Java 2 (JDK 1.2 και νεότερο) επιτρέπει λεπτομερή έλεγχο πρόσβασης στους διάφορους πόρους του συστήματος, προκειμένου να περιορίσει την πρόσβαση σε ανεπιθύμητους χρήστες. Αυτό σημαίνει ότι μπορούμε, να επιτρέψουμε στους αιτούντες να έχουν πρόσβαση μόνο στους πόρους που είναι επιθυμητό. Στη Java μπορεί να οριστεί μια πολιτική ασφάλειας μέσω του security manager. Ένας security manager ενεργοποιείται μέσω της εντολής :

System.setSecurityManager(new RMISecurityManager());

Οι πιο συχνά χρησιμοποιούμενοι τύποι άδειας είναι οι ακόλουθοι:

Η οθόνη διεπαφής χρήστη: Περιορισμένη πρόσβαση στη διεπαφή του χρήστη - για παράδειγμα, είναι δυνατόν να επιτρέψουν στους αιτούντες να διαβάζουν την τιμή των pixel οθόνης ή άλλων γεγονότων που συμβαίνουν στο περιβάλλον εργασίας χρήστη. Με αυτό τον τρόπο αποκλείουμε κακόβουλες εφαρμογές να υποκλέψουν στιγμιότυπα της οθόνης, να εμφανίσουν παράθυρα με σκοπό την υποκλοπή προσωπικών δεδομένων ή ακόμα και να προσομοιώσουν τις ενέργειες ενός χρήστη. Αυτός ο τύπος άδειας περιλαμβάνει άδεια για πρόσβαση στο «πρόχειρο» (accessClipBoard), παρακολούθηση των γεγονότων που συσχετίζονται με το γραφικό περιβάλλον (listenToAllAwtEvents), άδεια για πρόσβαση σε pixel της οθόνης (readDisplayPixels), κλπ.

Αρχεία. Η άδεια μπορεί να χορηγηθεί ώστε να διαβάζουν (read), να γράφουν (write), να εκτελούν (execute) ή να διαγράφουν (delete) συγκεκριμένα αρχεία, ή τμήματα του συστήματος αρχείων.

Sockets. Η άδεια μπορεί να χορηγηθεί για να περιμένει συνδέσεις, να δέχεται συνδέσεις, ή να εκτελέσει DNS ερωτήματα.

Ιδιότητες. Η άδεια μπορεί να χορηγηθεί για πρόσβαση σε διάφορα συστήματα και σε ιδιότητες που σχετίζονται με την JVM (για παράδειγμα, ονόματα χρηστών ή καταλόγων).

Από αυτούς τους τέσσερις τύπους, θα δούμε πώς δίνουμε δικαιώματα σε αρχεία και σε Sockets.

Δικαιώματα Αρχείων:

Τα δικαιώματα πρόσβασης ελέγχονται από μπλοκ με την ακόλουθη σύνταξη:

```
grant {

    permission java.io.FilePermission "filePath", "permissionList";

};
```

όπου filePath είναι η διαδρομή του αρχείου και όπου permissionList είναι μια λίστα με δικαιώματα (**read, write, delete, execute**) διαχωρισμένη με κόμματα.

Σημειώστε στο filePath μπορούν να δοθούν δικαιώματα για παραπάνω από ένα αρχεία και φακέλους κάνοντας χρήση wildcard χαρακτήρων. Με την χρήση του χαρακτήρα «*» στο τέλος του filePath αναφερόμαστε σε όλα τα αρχεία στον τρέχοντα κατάλογο, ενώ με το χαρακτήρα «-» αναφερόμαστε σε όλα τα αρχεία στον τρέχοντα κατάλογο και όλους τους υποκαταλόγους.

```
grant {

    permission java.io.FilePermission "/home/bob/-", "read";

    permission java.io.FilePermission "/home/bob/temp/*", "read, write, delete";

};
```

Παράδειγμα σε Windows Λειτουργικό Σύστημα:

```
grant {

    permission java.io.FilePermission "C:\\home\\download\\-", "read";

    permission java.io.FilePermission "C:\\home\\download\\temp\\*", "read, write,
delete";

};
```

Δικαιώματα Socket:

Τα δικαίωμα που αφορούν συνδέσεις αναφέρονται στις διάφορες λειτουργίες που μπορούν να γίνουν σε επίπεδο των socket. Τέτοιες είναι :

- Ανάλυση ενός ονόματος domain (**resolve**), δηλαδή η μετάφραση μιας ip διεύθυνσης σε όνομα
- Σύνδεση (**connect**), δημιουργία μίας σύνδεσης (socket connection) προς μία απομακρυσμένη εφαρμογή
- Αναμονή σύνδεσης (**listen**), δημιουργία ενός server socket για την αναμονή συνδέσεων από απομακρυσμένους χρήστες. Με αυτή την άδεια καθορίζουμε τις επιτρεπόμενες θύρες οι οποίες αναμένουν συνδέσεις.
- Αποδοχή σύνδεσης (**accept**). Με αυτή την άδεια καθορίζουμε ποιες απομακρυσμένες εφαρμογές έχουν δικαίωμα να συνδεθούν στην εφαρμογή μας.

Μπορούμε να ορίσουμε δικαιώματα στις συνδέσεις που δέχεται ένα σύστημα, προσθέτοντας την παρακάτω εντολή στην πολιτική μας:

```
permission java.io.SocketPermission "host:ports", "permissions";
```

όπου host:ports η IP διεύθυνση και η θύρα που επιτρέπεται η πρόσβαση.

Σαν διεύθυνση θα μπορούσε να δηλωθεί κάποιο domain. Επίσης θα μπορούσαν να χρησιμοποιηθούν επεκτάσεις **μόνο** στα domain σε περίπτωση που χρησιμοποιηθεί ο χαρακτήρας «*****», πχ .«*.com» όπου δικαίωμα πρόσβασης έχουν όλα τα domain που τελειώνουν σε .com. Επιπλέον υπάρχει η δυνατότητα να δηλωθεί παραπάνω από μια θύρα, κάνοντας χρήση του χαρακτήρα «-» πχ: «1024-1053» όπου δικαίωμα πρόσβασης έχουν όλες οι θύρες από 1024 μέχρι 1053. Σε περίπτωση που ορίζεται μόνο ένας αριθμός μαζί με τον χαρακτήρα «-», εάν το σύμβολο αυτό είναι πριν από τον αριθμό τότε δικαίωμα πρόσβασης έχουμε από όλες τις μικρότερες θύρες ενώ, εάν είναι μετά έχουμε από όλες τις μεγαλύτερες θύρες.

```
grant {

      //Socket permissions

    permission java.net.SocketPermission "localhost:1024-", "listen";

    permission java.net.SocketPermission "*.example.com:1024-", "connect, accept";

    permission java.net.SocketPermission "*. example.com:80", "connect";

};
```

Code Base

Η γενική ιδέα της πολιτικής ασφάλειας είναι να έχουμε ένα αρχείο με κανόνες οι οποίοι ισχύουν για παραπάνω από μια εφαρμογές που έχουμε δημιουργήσει. Υπάρχουν όμως περιπτώσεις που θέλουμε να δώσουμε επιπλέον δικαιώματα. Κάνοντας χρήση του codeBase μπορούμε να δώσουμε δικαιώματα σε διαφορετικές εφαρμογές.

```
grant codeBase "file://home/bob/trustedAps" {
    permission java.security.AllPermission;
};

grant {
    //File permissions
    permission java.io.FilePermission "/home/bob/-", "read";
    permission java.io.FilePermission "/home/bob/temp/*", "read, write, delete";
    //Socket permissions
    permission java.net.SocketPermission "*. example.com:1024-","connect,accept";
    permission java.net.SocketPermission "*. example.com:80", "connect";
};
```

Η παραπάνω πολιτική ασφάλειας επιτρέπει να γίνεται οποιαδήποτε ενέργεια στις εφαρμογές του φακέλου home/bob/trustedAps ενώ για τις υπόλοιπες εφαρμογές θα ισχύει η πολιτική που ακολουθεί.

Τοποθέτηση των αρχείων με τις πολιτικές ασφάλειας

Τη πολιτική ασφάλειας την γράφουμε σε ένα αρχείο με κατάληξη **.policy**, πχ: secure.policy. Υπάρχει η δυνατότητα να προσδιοριστεί μια γενική πολιτική ασφάλειας η οποία θα βρίσκεται στον κεντρικό κατάλογο της java **jdk/jre/lib/security/java.policy** ή μια πολιτική ασφάλειας για κάθε εφαρμογή η οποία θα είναι τοποθετημένη στο φάκελο της εφαρμογής.

Φόρτωση πολιτικής ασφάλειας

Το αρχείο της πολιτικής ασφάλειας το φορτώνουμε την στιγμή που θα τρέχουμε τον κώδικά μας πχ:

```
java -Djava.security.policy=MyServer.policy RMIServer
```

Εργαστηριακή Άσκηση

Εκτελέστε την εφαρμογή Chat που υλοποιούμε με την τεχνική callback ορίζοντας τις απαραίτητες πολιτικές ασφαλείας για τον πελάτη αλλά και για τον εξυπηρετητή.

Απάντηση

Στον κώδικα που δίνεται στο παράδειγμα του Chat θα πρέπει να προσθέσετε τις εντολές :

```
RMISecurityManager security = new RMISecurityManager();
System.setSecurityManager(security);
```

για να ενεργοποιηθούν οι πολιτικές ασφαλείας.

Οι άδειες που πρέπει να εκχωρηθούν σε επίπεδο εξυπηρετητή και πελάτη φαίνονται ακολούθως.

Server Security policy

```
grant {
    //File permissions
    permission java.io.FilePermission "D:\\programming\\DS\\ChatRMIServer\\src\\*", "read,
write";
// Το D:\\programming\\DS\\ChatRMIServer είναι το μονοπάτι στο οποίο εμείς έχουμε
// δημιουργήσει το project του εξυπηρετητή

    //Socket permissions
    permission java.net.SocketPermission "localhost:1024-","accept";
    permission java.net.SocketPermission "localhost:1099","connect";
    permission java.net.SocketPermission "localhost:1024-","connect";
};
```

Client Security policy

```
grant {

    //Socket permissions
   permission java.net.SocketPermission "localhost:1024-","accept";
   permission java.net.SocketPermission "localhost:1024-","connect";
};
```