```
            // this handler will save ALL log messages in the file
            trustFh = new FileHandler("log.client.trust.txt");
            integrityFh = new FileHandler("log.client.integrity.txt");
            policyFh = new FileHandler("log.client.policy.txt");
            // the format is simple rather than XML
            trustFh.setFormatter(new SimpleFormatter());
            integrityFh.setFormatter(new SimpleFormatter());
            policyFh.setFormatter(new SimpleFormatter());
            trustLogger.addHandler(trustFh);
            integrityLogger.addHandler(integrityFh);
            policyLogger.addHandler(policyFh);
            trustLogger.setLevel(java.util.logging.Level.ALL);
            integrityLogger.setLevel(java.util.logging.Level.ALL);
            policyLogger.setLevel(java.util.logging.Level.ALL);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
    public void discovered(DiscoveryEvent evt) {
        ServiceRegistrar[] registrars = evt.getRegistrars();
        Class [] classes = new Class[] {FileClassifier.class};
        FileClassifier classifier = null;
        ServiceTemplate template = new ServiceTemplate(null, classes,
                                                       null);

        for (int n = 0; n < registrars.length; n++) {
            System.out.println("Lookup service found");
            ServiceRegistrar registrar = registrars[n];
            try {
                classifier = (FileClassifier) registrar.lookup(template);
            } catch(java.rmi.RemoteException e) {
                e.printStackTrace();
                System.exit(4);
                continue;
            }
            if (classifier == null) {
                System.out.println("Classifier null");
                continue;
            }
            System.out.println("Getting the proxy");
            // Get the proxy preparer
            ProxyPreparer preparer = null;
            try {
                preparer =
                (ProxyPreparer) config.getEntry(
                                                "client.TestFileClassifierProxyPre-
parer",
```

The runtime needs to be told about these classes, which you can do by using the runtime define:

```
-Djava.security.properties=security.properties
```

where `security.properties` is a file containing the single line saying which Jini class to use for dynamic policies.

```
policy.provider=net.jini.security.policy.DynamicPolicyProvider
```

For the client, an array of permissions specifies the permissions the client will grant to a proxy. This array is set in the `BasicProxyPreparer`.

The server can set a permission in the `BasicILFactory`. This permission is used to perform server-side access control on incoming remote calls.

Summary

Ensuring security on the network is a complex task, and the Jini possibilities of mobile code increase the security risks. This chapter presented an end-programmer's view of the new Jini 2.0 security. The architecture behind the Jini security model is highly configurable, and we've looked at one set of "plug-ins" to make it (relatively) easy for you as a programmer. However, if you want more control over any part of this process, be aware that you can dig further into this architecture and roll your own for almost all parts of it.