THE JINI SECURITY MODEL REVISITED (REDUX)

Brian Murphy
Sun Microsystems

Basic Security Concepts

- Authentication
  - Tell me 'who' you are, and prove it
- Authorization and access control
  - Based on who you are, this is what I'll allow you to do
- Integrity
  - Guarantee no one has has corrupted your communication
    - Cryptographic checksums
- Confidentiality
  - Guarantee no one is 'listening' – communication has not been intercepted
    - Encryption

Java^{TM} Authentication – JAAS

Java Authentication And Authorization Service

- Securely determine who is executing Java code
- Pluggable infrastructure
  - > Implements PAM - Pluggable Authentication Module
- To determine how authentication should be done
  - > **javax.security.auth.login. LoginContext**
  - > **javax.security.auth.login. Configuration**
- Provider: javax.security.auth.spi.LoginModule
  - > **com.sun.security.auth.module.KeyStoreLoginModule** (JSSE)
  - > **com.sun.security.auth.module.Krb5LoginModule** (Kerberos)

Java Authorization – JAAS (Again)

- Ensure authenticated user has permission to perform requested actions
- Extends Java access control mechanism
  - Standard access control based on codesource
    - Where the code originated from
    - Who signed the code
  - Extended to also be based on the user or entity running the code
    - Represented by **javax.security.auth.Subject**
    - With **java.security.Principal**'s and credentials

Java Access Control – Policy File

- Keystore entry
  - > Lookup public keys of signers
  - > Map principal aliases to X.509 distinguished names
- Grant entries
  - > CodeBase – location of the code being executed
  - > SignedBy – public key certificate alias to verify signature
  - > Principal – 'who' the code is executing as
- Permission entries
  - > Permission class name with target and action
  - > SignedBy

Secure Message Exchange

Authentication, Integrity, Confidentiality – JSSE & JGSS API

- Java Secure Socket Extension – JSSE
  - > Java version of SSL/TLS protocols (RFC 2246)
  - > Pluggable, provider architecture
  - > Socket based communication
- Java Generic Security Services API (RFC 2853)
  - > Kerberos Version 5 (RFC 1964)
  - > Selective encryption
  - > Token based communication
- Used in conjunction with JAAS for authentication

JAAS LoginContext And Subject

```
public void main(String[] args) throws Exception {
    final LoginContext loginContext =
               new LoginContext(“app-client.jaas.login”);
    try {
        loginContext.login();
        Subject.doAsPrivileged (
             loginContext.getSubject(),
             new PrivilegedExceptionAction() {
                 public Object run() throws Exception {
                     Client thisClass = new Client();
                     thisClass.runClient();
                     return null;
                 }//end run
             },//end PrivilegedExceptionAction
             null
        );//end doAsPrivileged
    } catch(Throwable e) { e.printStackTrace(); }
}//end main
```

JAAS Login Config File – JSSE

-Djava.security.auth.login.config=/home/app/config/jsse.login

```
app-client.jaas.login {
    com.sun.security.auth.module.KeyStoreLoginModule
                                                    required
    keyStoreAlias=″app-client″
    keyStoreURL=″file:/home/app/jsse/app-client.keystore″
    keyStorePasswordURL=″file:/home/app/jsse/
                                          app-client.passwd″;
};



app-admin.jaas.login {
    com.sun.security.auth.module.KeyStoreLoginModule
                                                    required
    keyStoreAlias=″app-admin″
    keyStoreURL=″file:/home/app/jsse/app-admin.keystore″
};
```

JAAS Login Config File – Kerberos

-Djava.security.auth.login.config=/home/app/config/kerb.login

```
app-client.jaas.login {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="/home/app/kerb/servers.keytab"
    storeKey=true
    doNotPrompt=true
    principal="app-client"
};


app-admin.jaas.login {
    com.sun.security.auth.module.Krb5LoginModule required
    storeKey=true
};
```

Java Policy File – JSSE

```
keystore “file:/home/certs/public.truststore”;

....................

grant codebase  “http://cHost.sun.com:8080/client-dl.jar”
      signedBy  “certs.cto.sun.boston.ma.usa”
      principal “app-client”
{
    permission app.service.ServerPermission “getPrice”;
};


grant codebase  “http://cHost.sun.com:8080/admin-dl.jar”
      signedBy  “certs.cto.sun.boston.ma.usa”
      principal “app-admin”
{
    permission java.io.FilePermission “/tmp/log”,
                                              “read, write”;

    permission app.service.ServerPermission “administer”;
    permission app.service.ServerPermission “shutdown”;
};
```

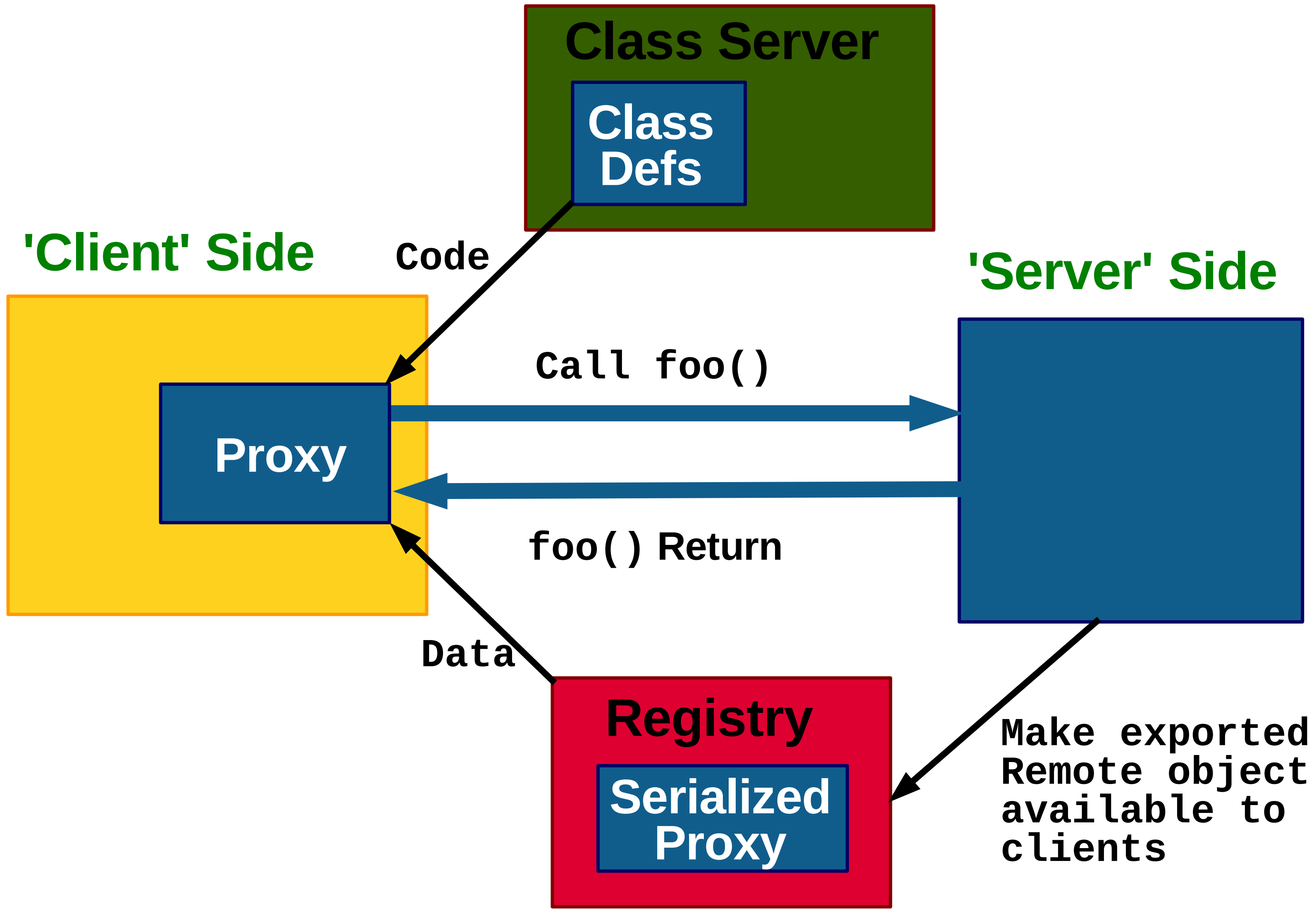
Java Policy File – Kerberos

```
.....................

grant codebase  "http://cHost.sun.com:8080/client-dl.jar"
      signedBy  "certs.cto.sun.boston.ma.usa"
      principal
        javax.security.auth.kerberos.KerberosPrincipal
                                                    "app-client"
{
    permission app.service.ServerPermission "getPrice";
};

grant codebase  "http://cHost.sun.com:8080/admin-dl.jar"
      signedBy  "certs.cto.sun.boston.ma.usa"
      principal
        javax.security.auth.kerberos.KerberosPrincipal
                                                     "app-admin"
{
    permission java.io.FilePermission "/tmp/log",
                                                "read, write";
    permission app.service.ServerPermission "administer";
    permission app.service.ServerPermission "shutdown";
};
```

The Java Remote Call Model

- Remote object is exported on the server side
  - Produces a proxy to the remote object
- Client side obtains the proxy somehow (RMI Registry, Lookup Service, UDDI, etc.)
  - Code may be downloaded in the process
- Execution of the call is initiated on the client side
- Communication between client and server occurs
- Execution of the call occurs on the server side

The Java Remote Call Model In Action



Network Security Model Requirements

- Provide basic network security for remote calls
  - > Mutual authentication
    - > Server authenticates the client and vice-versa
  - > Mutual authorization and access control
  - > Object integrity (code as well as data integrity)
  - > Confidentiality
- Consistent with principles of Jini technology
  - > The network cannot be ignored (Deutsche's 7 fallacies)
  - > Agreement is in the public interface
  - > Code is moved

The Remote Call Model And Security

- Problem: how to operate securely in the face of
  - Remote calls
  - Downloaded code
- Java security gets us only so far
  - No mechanism for verifying 'foreign' code can be trusted
  - Static policy
    - No mechanism for granting permissions dynamically
  - Dealing with the dynamic nature of RMI communication
    - Custom socket factories can be inflexible
- Java provider model (SPI's) gets us only so far
  - Configuration can be tedious and inflexible

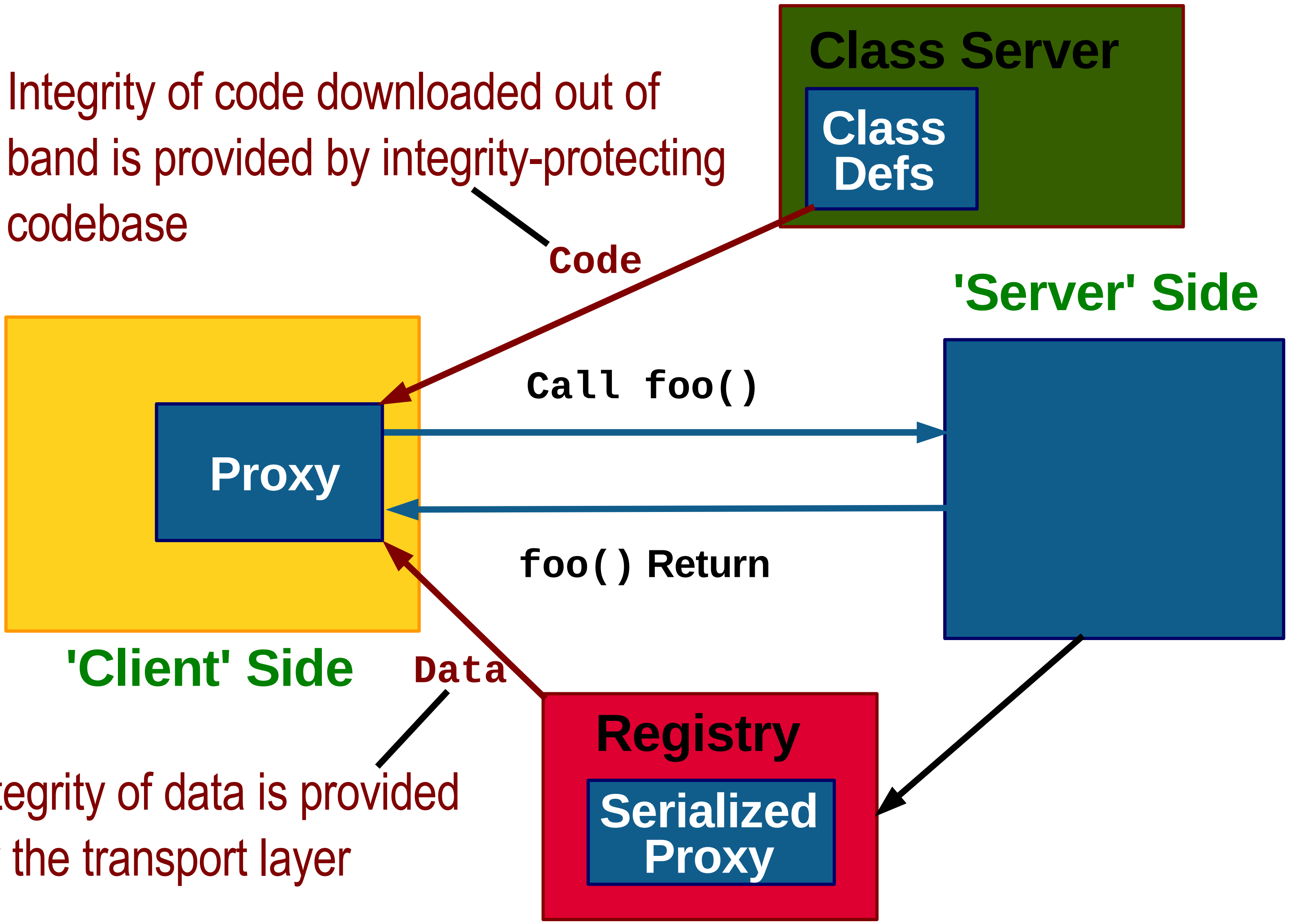
The Constraint Model

Providing Authentication, Integrity, Confidentiality

- Specify what a Subject ***must do, must not do***
  - Server/ClientAuthentication.YES/NO (JAAS Subjects)
  - Integrity.YES/NO
  - Confidentiality.YES/NO
  - Quality of service (max. threads, connection timeout, etc.)
- Enforced on a per-method basis
  - Example: authenticate on write, but anonymous for read
- Proxy implements **RemoteMethodControl** interface
  - Indicates proxy supports network security
  - Allows client to attach constraints to proxy

Object Integrity & The Remote Model

Integrity of code downloaded out of band is provided by integrity-protecting codebase



Integrity of data is provided by the transport layer

Providing Object Integrity

- Verify integrity of code as well as data
  - > Data verification handled by transport layer
  - > Code verification requires integrity-protecting codebases
- Integrity-protecting codebases
  - > HTTPS URLs (can have high overhead)
  - > HTTPMD URLs with JAR files (MD = Message Digest)
    - > Digest integrity covered by in band integrity verification
    - > httpmd://raglan.sun.com:8080/reggie-dl.jar;md=3dc20ac6b7b854b24224c8ade7b30db5
- Other schemes possible – pluggable API

Proxy Trust Verification

Downloaded Code Presents A Unique Problem

- Client receives and executes 'foreign' proxy code
  - Client must verify that the proxy code can be trusted
    - Before granting any permissions to the proxy
    - Before making any remote calls through the proxy
- Solution: proxy trust verifiers
  - Client obtains a verifier from the trusted server
  - Through verifier, asks server if proxy can be trusted
- Trust verifiers minimize client's prior knowledge
  - Client has to know only who the server authenticates as
  - Client needs to know nothing else
    - Not codebase or signers or protocols
  - Allow server impl changes without client reconfiguration

Providing Authorization

Downloaded Code And Dynamic Policy

- Once a Subject is authenticated, determine actions allowed by that Subject
- Standard Java access control mechanism
  - Static permission grants in security policy
  - Specify what a Subject (and/or codesource) ***can do***
- Also requires a mechanism for granting permissions dynamically – dynamic policy provider
  - 'Foreign' code is downloaded into client's VM
  - Codesource not necessarily known in advance

Impact On The Remote Call Model

Server Side vs Client Side: Exporters And ProxyPreparers

- Server side: exporting Remote objects
  - > Specify the communication transport (SSL, Kerberos, etc.)
  - > Specify required permissions for access control
  - > Attach constraints from server side's point of view
- Client side: proxy preparation
  - > Verify that the proxy can be trusted
  - > Attach constraints to the proxy
  - > Grant necessary permissions to the proxy

Supporting Security In The Model

A New RMI Implementation – Jini Extensible Remote Invocation

- Current RMI implementations not aware of the new network security model or constraints
  - RMI/JRMP, RMI/IIOP
- Jini ERI (“RMI 2.0”)
  - Supports constraints and the network security model
  - Provides customizable server-side authorization
    - Method-level permission checks against authenticated clients
  - Provides code integrity in client-side invocation layer
  - Supports pluggable transport providers
    - SSL & HTTPS (JSSE), Kerberos (GSS), TCP, HTTP, JXTA

Deployment-Time Configuration

New Mechanism For Expressing Network Security Requirements

- Need ability to change secure configuration without changing/recompiling code
- Need to configure complex Java objects
  - Exporters and ProxyPreparers
  - Not just Strings and primitive types
  - Name=Value pairs not enough
    - Java property files, XML configuration files, text files
- Extensible and pluggable
  - Configuration provider set via resource

Server Side Configuration

Configuring An Exporter

```
app.service {
   ...............
   private endpt = SSlServerEndpoint.getInstance(0);

   private constraints = new BasicMethodConstraints
     (new InvocationConstraints
         (new InvocationConstraint[]{Integrity.YES},
          null) );

   private ilFactory = new ProxyTrustILFactory
                             (constraints,
                              ServerPermission.class);

   serverExporter = new BasicJeriExporter
                                  (endpt, ilFactory);
   ...............
}//end app.service
```

Impact On Security Policy

Server Side Policy File (JSSE)

```
keystore “file:/home/certs/public.truststore;”


grant codebase “file:/home/app/lib/app-service.jar {
    permission java.security.AllPermission “”, “”;
};


grant principal “app-client” {
    permission app.service.ServerPermission “getPrice”;
};


grant principal “app-admin” {
    permission java.io.FilePermission “/tmp/log”,
                                             “read, write”;
    permission app.service.ServerPermission “administer”;
    permission app.service.ServerPermission “shutdown”;
};
```

Client Side Configuration

Configuring A ProxyPreparer

```
app.client {
   ..............
   private verifyTrust = true;
   private constraints /* next slide for details */
               = new BasicMethodConstraints(...);
   private dynamicPermissions /* slide after next */
               = new Permission[] {...};

   proxyPreparer = new BasicProxyPreparer
                            (verifyTrust,
                             constraints,
                             dynamicPermissions);
   ..............
   loginContext = new LoginContext
                        ("app-client.jaas.login");
}//end app.client
```

Detail: Constraints Configuration

Configuring A ProxyPreparer

```
...............

private reqs = new InvocationConstraint[]
    { Integrity.YES,
      ClientAuthentication.YES,
      ServerAuthentication.YES,
      new ServerMinPrincipal(serverPrincipal) };

private prefs = null;

private reqsAndPrefs =
        new InvocationConstraints(reqs, prefs);

private constraints =
        new BasicMethodConstraints(reqsAndPrefs);

...............
```

Detail: Dynamic Permissions Config

Configuring A ProxyPreparer

```
...............

private dynamicPermissions = new Permission[]
{ new AuthenticationPermission( clientPrincipal,
                                serverPrincipal,
                                ”connect” ) };
...............
```

- When a ”connect” request is made through the proxy to any server that authenticates as the principal referenced by serverPrincipal
- Grant permission – to the proxy – to authenticate (run) as the principal referenced by clientPrincipal

Client Code With Jini Configuration

```
public void main(String[] args) throws Exception {
    try {
        cfg = ConfigurationProvider.getInstance(...);
        try {
            loginContext = (LoginContext)config.getEntry
                      (“app.client”, “loginContext”, ...);
            loginContext.login();
            Subject.doAsPrivileged (
                 loginContext.getSubject(),
                 new PrivilegedExceptionAction() {
                     public Object run() throws Exception {
                         Client thisClass = new Client(cfg);
                         thisClass.runClient();
                         return null;
                     }//end run
                 },//end PrivilegedExceptionAction
                 null
            );//end doAsPrivileged
        } catch(NoSuchEntryException e0) {//no Subject
            Client thisClass = new Client(cfg);
            thisClass.runClient();
    } catch(Throwable e1) { e1.printStackTrace(); }
 }//end main
```

Summary – Extending Java Security

- Provide network security in the face of remote calls and downloaded code
  - Authentication, authorization & access control, object integrity, confidentiality
- New concepts and mechanisms
  - Constraints
  - Proxy trust verification
  - Dynamic policy
  - Proxy preparation
    - Verify trust, attach constraints, set necessary permissions
  - A new RMI implementation – Jini ERI
  - Deployment-time configuration of complex objects

THE JINI SECURITY MODEL REVISITED (REDUX)

Brian Murphy
brian.t.murphy@sun.com