JINI SECURITY AN OVERVIEW

SORCER LAB
sooriana@cs.ttu.edu

Presentation Agenda

- Server and Interface
- JASS Configuration
- Trusted and un-trusted proxy
- Verifiers and TrustEquivalence
- SSL configuration
- Server policy
- Server configuration
- httpmd and Anotation provider
- Client
- Proxy trust verification
- Dynamic permission grants

Java™ Technologies Used

- Java™ 2 Platform, Standard Edition (J2SE™)
- Java™ Remote Method Invocation (Java RMI)
- Java™ Authentication and Authorization Service (JAAS) API
- Java™ Secure Socket Extension (JSSE)
- Java™ Archive (JAR) files

Hello interface and Server


```
public interface Hello extends Remote {
        String sayHello() throws RemoteException;
}

Public class Server {
        private static Server serverImpl;

        protected Server(String[] configOptions) throws
        ConfigurationException {
                config = ConfigurationProvider.getInstance(configOptions);
        }

        Public static void main(String[] args) {
                serverImpl = new Server(args);
                serverImpl.init();
        }

        public TrustVerifier getProxyVerifier() {
                return new SmartProxy.Verifier(serverProxy);
        }
}
```

JAAS login of server


```
protected void init() throws Exception {
        /* JAAS login */
        loginContext = newLoginContext("com.sun.jini.example.hello.Server");

        if (loginContext == null) {
            initAsSubject();
        } else {
            loginContext.login();
            Subject.doAsPrivileged(
                        loginContext.getSubject(),
                        new PrivilegedExceptionAction() {
                            public Object run() throws Exception {
                                initAsSubject();
                                return null;
                            }
                        },
                        null);
        }
  }
```

JAAS login configuration

The information for the configuration of LoginContext to be used is given in the command line as following :

```
java    -Djava.security.auth.login.config=config/ssl-server.login \
        ……
```

ssl-server.login

```
com.sun.jini.example.hello.Server {
   com.sun.security.auth.module.KeyStoreLoginModule required
   keyStoreAlias="server"
   keyStoreURL="file:lib/server.keystore"
   keyStorePasswordURL="file:lib/server.password";
};
```

ServerProxy – Directly Trusted by clients


```
protected void initAsSubject() throws Exception {
        /* Get exporter which supports jini security framework */
        Exporter exporter = getExporter();
        serverProxy = (Hello) exporter.export(this);

        /* Create the smart proxy */
        SmartProxy smartProxy = new SmartProxy(serverProxy);

        ……..

        JoinManager joinManager =
           new JoinManager(smartProxy, null /* attrSets */,
                           getServiceID(),
                           discoveryManager,
                           null /* leaseMgr */,
                           config);
}
```

Smart Proxy-Not trusted by clients


```
Public class SmartProxy implements Serializable, Hello,
                                                    RemoteMethodControl {
        //The exported server proxy
        Hello serverProxy;

        public SmartProxy(Hello serverProxy) {
                this.serverProxy = serverProxy;
        }

        //delegate all hello calls to server proxy
        //delegate all RemoteMethodControlMethod to serverProxy

        //Provide access to the underlying server proxy to permit the
        //ProxyTrustVerifier class to verify the proxy
        private ProxyTrustIterator getProxyTrustIterator() {
           return new SingletonProxyTrustIterator(serverProxy);
        }
        //return verifier object for clients. See later….
        final static class  Verifier { …. }
}
```

Verifier for the client


```
final static class Verifier implements TrustVerifier, Serializable {
    private final RemoteMethodControl serverProxy;

    Verifier(Hello serverProxy) {
        this.serverProxy = (RemoteMethodControl) serverProxy;
    }

    public boolean isTrustedObject(Object obj, TrustVerifier.Context ctx)
                                                    throws RemoteException  {

        if (obj == null || ctx == null)  throw new NullPointerException();
        else if (!(obj instanceof  SmartProxy))  return false;

        RemoteMethodControl otherServerProxy =
            (RemoteMethodControl) ((SmartProxy) obj).serverProxy;

        MethodConstraints mc = otherServerProxy.getConstraints();
        TrustEquivalence trusted =
                (TrustEquivalence) serverProxy.setConstraints(mc);
          return trusted.checkTrustEquivalence(otherServerProxy);
    }
}
```

What is TrustEquivalence ?

•TrustEquivalence: Defines an interface for checking that an object (that is not yet known to be trusted) is equivalent in trust, content, and function to a known trusted object. This is intended to be used by TrustVerifiers.
•Proxies returned by exported object using BasicILFactory directly implements this semantics.
•TrustEquivalence.checkTrustEquivalence() is a recursive operation.

“In general, a sub object either needs to implement TrustEquivalence, in which case the recursion is to call checkTrustEquivalence, or it's an object whose equals method is known to be "strong" enough to imply trust equivalence.

For example, BasicInvocationHandler recursively uses checkTrustEquivalence on the ObjectEndpoints, but uses equals on the constraints because the MethodConstraints spec requires the equals method to be a sufficient substitute for checkTrustEquivalence.” –Bob Schiffler.

Starting your server :


```
java -Djava.security.manager= \
    -Djava.security.policy=config/ssl-server.policy \
    -Djava.security.auth.login.config=config/ssl-server.login \  //JAAS
    -Djava.security.properties=config/dynamic-policy.security-properties \
    -Djavax.net.ssl.trustStore=lib/truststore  \          //For your SSLEndpoint
    -Djava.protocol.handler.pkgs=net.jini.url \
    -Djava.rmi.server.RMIClassLoaderSpi= \
          com.sun.jini.example.hello.MdClassAnnotationProvider   \
    -Dexport.codebase.source=lib \
    -Dexport.codebase=httpmd://$host:8080/server-dl.jar\;md5=0  \
    -jar lib/server.jar \
    config/ssl-server/ssl-server.config
```

Setting up for your SslServerEndpoint

Generate Server Keys :

```
//Generate X509 keys for server and store in server.keystore
keytool -keypass serverpw -storepass serverpw \
        -keystore lib/server.keystore \
        -genkey -validity 1800 -alias server -dname CN=Server

//Extract  the server certificate
keytool  -keypass serverpw -storepass serverpw \
         -keystore lib/server.keystore \
         -export -alias server -file lib/server.cert

//Add the certificate to your truststore
keytool  -keypass trustpw -storepass trustpw \
        -keystore lib/truststore \
        -import -noprompt -alias server -file lib/server.cert

//Inform your JVM via the following system property
-Djavax.net.ssl.trustStore=lib/truststore

//For added fun, try -Djavax.net.debug=ssl
```

Security policy for server (ssl-server.policy)

-Djava.security.policy=config/ssl-server.policy

```
/* Security policy for SSL server */
/* Keystore containing trusted certificates to use for authentication */
keystore "../..${/}lib${/}truststore";

/* Grant all permissions to local JAR files */
grant codeBase "file:lib${/}*" {
    permission java.security.AllPermission;
};

/* Grant permissions to client principal */
grant principal "client" {
    /* Call sayHello method */
    permission com.sun.jini.example.hello.ServerPermission "sayHello";
};

/* Grant permissions to all principals */
grant {
    /* Call getProxyVerifier method */
   permission com.sun.jini.example.hello.ServerPermission "getProxyVerifier";
};
```

ServerPermission & AccessPermission


```
import net.jini.security.AccessPermission;
 //Represents permissions used to express the access control policy for the
 //Server class. The name specifies the names of the method which you have
 //permission to call using the matching rules provided by AccessPermission.
public class ServerPermission extends AccessPermission {
    //Creates an instance with the specified target name.
   public ServerPermission(String name) {
            super(name);
   }
}
```

- Represents permission to call a method. An instance of this class contains a target-name but no actions list; you either have the named permission or you don't.
- This class can be used with BasicInvocationDispatcher.
- Recommended that each remote object exported by an Exporter, has different subclass of AccessPermission, to allow separation of grants in policy files.

ssl-configuration for server exporter

ssl-server.config

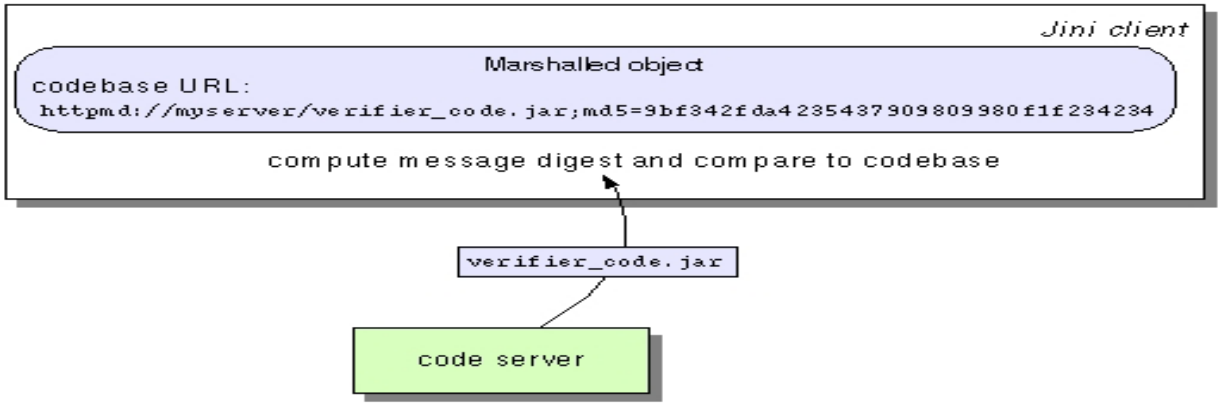
```
exporter =
        /* Use secure exporter */
        new BasicJeriExporter(
          SslServerEndpoint.getInstance(0), /* Use SSL transport */
          /* Support ProxyTrust */
          new ProxyTrustILFactory(
                /* Require integrity for all methods */
                new BasicMethodConstraints(
                   new InvocationConstraints(Integrity.YES, null)),
                /* Require ServerPermission */
                ServerPermission.class));
```

- BasicMethodConstains are used to enforce minimum constraints for remote calls
- Integrity.YES : Constraint on the integrity of message contents. (Both in-band and Out-of-band).
- The ServerPermission class, is used to perform server-side access control on incoming remote calls. (How ?)

What the configuration means ?

- ProxyTrustILFactory : To be used if the exported proxy needs to be a ProxyTrust.
- ProxyTrust.getVerifier() and ServerProxyTrust.getVerifier() are mapped by BasicInvocationHandler and BasicInvocationDispatcher used by ProxyTrustILFactory.
- Returned exported object implements all the interfaces of the object being exported and also RemoteMethodControl, TrustEquivalence and ProxyTrust
- BasicJeriExporter+ProxyTrustILFactory used when the (inner) proxy will be trusted by clients as the bootstrap proxy, and you put that (inner) proxy inside a smart proxy that won't directly be trusted by clients.
- If custom InvocationHandler/custom Endpoint is used, the exported proxy is not trusted directly by client. In such cases, you have to use ProxyTrustExporter instead of the above combination thus giving clear seperation between serverProxy (not trusted) and BootStrap proxy direcly trusted by client.

httpmd Protocol to protect out-of-band data



- Client gets object anotated with httpmd code-base. Object integrity ensured by the transport layer
- Httpd protocol involves the codebase anotated with a hash of the anotated jar file at the end Eg: http://cs.ttu.edu/xxx.jar;md5=123456789
- Now client, to verify code integrity downloads xxx.jar(unsecurely) ,computes hash(xxx.jar) and compares with anotated hash.
- URL protocol handlers in J2SE can be configured with additional protocol handler as in => -Djava.protocol.handler.pkgs=net.jini.url
- -Djava.rmi.server.RMIClassLoaderSpi= com.sun.jini.example.hello.MdClassAnnotationProvider

RMIClassLoaderSpi => service provider interface for RMIClassLoader. MdClassAnnotationProvider over-rides getCoudeAnotation() of RMIClassLoaderSpi

MdClassAnotationProvider


```
public class MdClassAnnotationProvider extends PreferredClassProvider {
    private final String codebase = HttpmdUtil.computeDigestCodebase(
                                    System.getProperty("export.codebase.source"),
                                    System.getProperty("export.codebase"));

    …..
    //Over-ride method of RMIClassLoaderSpi
    protected String getClassAnnotation(ClassLoader loader)  {
            return codebase;
    }
}
```

- **-Dexport.codebase.source=lib**
- **-Dexport.codebase=httpmd://$host:8080/server-dl.jar\;md5=0**
- **HttpmdUtil knows the hash algorithm to use from the last parameter of the export.codebase specified here.**

Client using secure ssl-server


```
public class Client {

public static void main(String[] args) throws Exception {
   //get the configuration and login context and login
   Subject.doAsPrivileged( loginContext.getSubject(),
        new PrivilegedExceptionAction() { public Object run() throws Exception {
                                          mainAsSubject(config); return null; } }, null);
}

static void mainAsSubject(Configuration config) throws Exception {
  // discover Hello service proxy
  ProxyPreparer preparer = (ProxyPreparer) config.getEntry(
     "com.sun.jini.example.hello.Client",
     "preparer", ProxyPreparer.class, new BasicProxyPreparer());

  //That's all a client has to do to make himself secure!!
  server = (Hello) preparer.prepareProxy(server);
  System.out.println("Server says: " + server.sayHello());
  System.exit(0);
 }
}
```

Starting your client


```
java -Djava.security.manager=  \
    -Djava.security.policy=config/ssl-client.policy  \
    -Djava.security.properties=config/dynamic-policy.security-properties \
    -Djava.security.auth.login.config=config/ssl-client.login \
    -Djavax.net.ssl.trustStore=lib/truststore  \
    -Djava.protocol.handler.pkgs=net.jini.url  \
    -Djava.rmi.server.RMIClassLoaderSpi=   \
                    com.sun.jini.example.hello.MdClassAnnotationProvider  \
    -Dexport.codebase.source=lib   \
    -Dexport.codebase=httpmd://$host:8080/sdm-dl.jar\;sha=0   \
    -Djavax.net.debug=ssl \
    -jar lib/client.jar   config/ssl-client.config
```

Client configuration


```
import com.sun.jini.config.KeyStores;
/* Keystore for getting principals */
private static users=  KeyStores.getKeyStore("file:lib/truststore", null);
private static clientUser = Collections.singleton(
                                        KeyStores.getX500Principal("client", users));
private static serverUser = Collections.singleton(
                                        KeyStores.getX500Principal("server", users));
/* Preparer for server proxy */
static preparer = new BasicProxyPreparer( /* Verify the proxy. */   true,
        // Require integrity, client authentication, and server
        //  authenticate with the correct principal for all methods.
        new BasicMethodConstraints(
              new InvocationConstraints(new InvocationConstraint[] {
                Integrity.YES, ClientAuthentication.YES, ServerAuthentication.YES,
                new ServerMinPrincipal(serverUser) },null)),
        /* Authenticate as client when connecting to server */
        new Permission[]   {
            new AuthenticationPermission(clientUser, serverUser, "connect") }
        );
```

Client configuration details

•Verify = true in BasicProxyPreparer implies the proxy prepared with this prepared must be verified. This is typically done by calling Security.verifyObjectTrust() .

•Once object is verified, then dynamic permissions are granted by calling Security.grant

•It then sets constrains on the proxy by calling RemoteMethodControl.setConstraints. Setting constraints to it returns a new Proxy. This is because Proxies are normally immutable.The new proxy with all appropriate constrains set is returned back.

•If any of the action fails, then RemoteException or SecurityException is thrown in accordance to communication related exception or security related exception

Client configuration details

BasicMethodConstraints apply to all remote calls made.

Integrity.YES : Data and Code integrity to be maintained for all method calls

ClientAuthentication.YES : Constraint on authentication of the client to the server.

ServerAuthentication.YES : Constraint on authentication of the server to the client.

ServerMinPrincipal.YES : Server must authenticate itself as at least all of the specified principals.

AuthenticationPermission(clientUser, serverUser, "connect") : When making outgoing remote calls (for action “connect”), give permission to authenticate as atmost clientUser (set of principals):maximum bound and atleast server User: maximum bound.

Dynamic Grants

NEED:

- Need to grant proxy permission to authenticate using client's Subject
  - Provide (indirect) access to private credentials

- Codebase and signers not known in advance
  - Implementation details of service

- Need to delay until after proxy trust verification
  - Otherwise untrusted code can act as client

SCOPE:

- Grant to all protection domains matching:
  - Class loader of proxy's top-level class
  - Principals of thread's current Subject
- Requires java.security.Policy provider support
  - DynamicPolicyProvider wrapper supplied

```
public final class Security {
        ...
        public static void grant(Class c, Permission[] perms);
}
```

Q&A

Demo