Security APIs

Marc Hadley
Sun Microsystems, Inc.

Goals

- Protocol neutrality
- Security mechanism neutrality
  - Enable support for WSS/WS-I BSP in the SOAP/HTTP binding
  - Flexible enough to accommodate other security mechanisms in other protocol bindings.
- Compatibility
  - Interoperability and application portability.
  - Use of existing technologies where possible and appropriate.

Goals cont'd

- Ease of use
  - Number of new concepts to the minimum
  - Avoid asking for information from inappropriate parties
    - E.g. deployment info from developers
  - Meaningful defaults and a graceful rise in exposure to detail as defaults are overridden.

Approach

- Separate abstract security requirements from concrete expression of those requirements
- Developers work at the abstract level
- Deployers configure concrete expression of abstract requirements
- Symbolic name links abstract to concrete
  - Abstract: Require integrity using config
  - Concrete: Integrity using config over SOAP means sign SOAP body using WSS
- Symbolic name used as key in concrete configuration

Capabilities

- Confidentiality
  - Protecting data from being read by anyone except the intended recipient
- Integrity
  - Providing assurance that the data received by a recipient is the same as the data sent by the originator
- Authentication
  - Ability to establish or constrain the identity of the source and/or recipient of a message.

Annotations

```
@Target({TYPE, METHOD})
public @interface MessageSecurity {
    public enum SecurityFeatures {
    CONFIDENTIALITY, INTEGRITY, AUTHENTICATION };

    SecurityFeatures[] security()
    default {INTEGRITY, CONFIDENTIALITY};
    SecurityFeatures[] outgoingSecurity()
    default {INTEGRITY, CONFIDENTIALITY};
    SecurityFeatures[] incomingSecurity()
    default {INTEGRITY, CONFIDENTIALITY};

    String securityConfiguration()
    default “javax.xml.rpc.security.Default”;
    String incomingSecurityConfiguration()
    default “javax.xml.rpc.security.Default”;
    String outgoingSecurityConfiguration()
    default “javax.xml.rpc.security.Default”;
}
```

Annotation Usage

```
@WebService
public class MyService {
   @WebMethod
   @MessageSecurity(
      incomingSecurity = AUTHENTICATION,
      incomingSecurityConfiguration =
        ”com.foo.corp.Default”)
   public void doIt() {
      ...
   }
}
```

APIs

```
public interface MessageSecurityRequirements {
    public void setSecurity(String configName,
        SecurityFeatures... features);
    public void setIncomingSecurity(
        String configName,
        SecurityFeatures... features);
    public void setOutgoingSecurity(
        String configName,
        SecurityFeatures... features);

    public String getOutgoingConfigName();
    public String getIncomingConfigName();
    public SecurityFeatures[] getOutgoingSecurity();
    public SecurityFeatures[] getIncomingSecurity();
}

MessageSecurityRequirements
    Service.getMessageSecurityRequirements();
MessageSecurityRequirements
    Binding.getMessageSecurityRequirements();
```

API Usage

```
ServiceFactory factory = ServiceFactory.newInstance();
Service service =
   factory.createService(SOME_SERVICE_QNAME);
SEI sei = (SEI)service.getPort(SEI.class);
Binding binding = sei.getBinding()
MessageSecurityRequirements msr =
   binding.getMessageSecurityRequirements()
msr.setOutgoingSecurity(
   “com.foo.corp.Default”, AUTHENTICATION);
```

Application Information

- Add new context property
  - Name:
    - javax.xml.rpc.security.callbackhandler
  - Type:
    - javax.security.auth.callback.CallbackHandler
- Applications
  - Implement CallbackHandler
  - Set this property on a binding provider.
- JAX-RPC runtime and handlers
  - Use this property to request information from the application during security processing.