Web Services Security

Presentation by Gunnar Peterson
www.arctecgroup.net

Brief History of Software
Mission Accomplished!

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/PERL</td>
<td>Network firewall &amp; SSL</td>
</tr>
</tbody>
</table>
Mission Accomplished!

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/PERL</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1997</td>
<td>JSP, ASP</td>
<td>Network firewall &amp; SSL</td>
</tr>
</tbody>
</table>
Mission Accomplished!

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/Perl</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1997</td>
<td>JSP, ASP</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1998</td>
<td>EJB, DCOM</td>
<td>Network firewall &amp; SSL</td>
</tr>
</tbody>
</table>
Mission Accomplished?

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/PERL</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1997</td>
<td>JSP, ASP</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1998</td>
<td>EJB, DCOM</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1999</td>
<td>SOAP, XML</td>
<td>Network firewalls &amp; SSL</td>
</tr>
</tbody>
</table>
Mission Accomplished?

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/PERL</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1997</td>
<td>JSP, ASP</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1998</td>
<td>EJB, DCOM</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1999</td>
<td>SOAP, XML</td>
<td>Network firewalls &amp; SSL</td>
</tr>
<tr>
<td>2001</td>
<td>SOA, REST</td>
<td>Network firewalls &amp; SSL</td>
</tr>
</tbody>
</table>
## Mission Accomplished?

<table>
<thead>
<tr>
<th>Year</th>
<th>Software</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>CGI/PERL</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1997</td>
<td>JSP, ASP</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1998</td>
<td>EJB, DCOM</td>
<td>Network firewall &amp; SSL</td>
</tr>
<tr>
<td>1999</td>
<td>SOAP, XML</td>
<td>Network firewalls &amp; SSL</td>
</tr>
<tr>
<td>2001</td>
<td>SOA, REST</td>
<td>Network firewalls &amp; SSL</td>
</tr>
<tr>
<td>2003</td>
<td>Web 2.0</td>
<td>Network firewalls &amp; SSL</td>
</tr>
<tr>
<td>2007</td>
<td>Cloud Computing</td>
<td>Network firewalls &amp; SSL</td>
</tr>
</tbody>
</table>
Why Services?

• Service Oriented Architecture goals
  – Virtualization - connect Bangalore, Beijing, and Bloomington
  – Interoperability - get Java, .Net working together
  – Reusability - how many claims/pricing/order mgmt systems does one company need?

High level view of services

• Service interface
  • Method signature, types, values
• Mapping layer
  • Mapping message to runtime implementation types and values
• Implementation
  • Application logic
• Message
  • Data payload
Lions, Tigers, and Port 80, Oh My!

- First came SOAP - invented as a firewall friendly protocol
- Bruce Schneier: “calling SOAP firewall friendly is like skull friendly bullet”

Information Security: A new oxymoron

Source: Robert Garigue http://1raindrop.typepad.com/1_raindrop/2007/02/thinking_about_.html
Security Goals

• Security as a Service
  – Virtualization
  – Interoperability
  – Reusability

Security Mechanisms

- Service Consumer
- Security Services
- Service Provider

- Authentication
- Authorization
- Audit

Assurance
Virtualization

Deploy and deliver authentication, authorization, and audit services in decentralized and distributed systems - Bangalore, Beijing, and Breckenridge

Interoperability

Standards based, consistent authorization policy enforcement (XACML, SAML)
Service Oriented Security in Einstein’s Universe

- Mainframes are Newton’s world
  - The computer
  - The price
  - The record
- Distributed computing is Einstein’s world
  - Pat Helland: Computers don’t make decisions, computers *try* to make decisions.
  - Its all about Memories, Guesses and Apologies
  - Security mechanisms don’t make policy-based decisions, security mechanisms *try* to make policy-based decisions
Memories, Guesses and Apologies in Security

• Memory
  – Security Policies - for example Triple A policy
• Guess
  – Security Policy Enforcement Decision
• Apology
  – Giant Global Bank is sorry your account was compromised!

Memories, Guesses and Apologies in Security

• Memories
  – Triple A Security Policies
  – Audit logs
  – User account information
  – Authorization Logic - concrete mapping Subject, Resource, Condition, Action
• Guesses
  – Security Policy Enforcement Decision Points
  – Authentication Logic
  – Monitoring, detection, fraud response
• Apologies
  – Identity Management tools - provisioning, deprovisioning
  – Reimburse customer for fraud losses
  – Compensating Transaction - Giant Global Bank is still sorry your account was compromised!
Trends

• Virtualization
  – Finding Vulns in a Virtualized World
    • Problem - Applications are more configured than coded. Runtime behavior and structure not apparent due to weak typing and inversion of control.
    • Result - finding bugs becomes harder.
    • Action - use screens to target finding time and resources
  – Fixing Vulns in a Virtualized World
    • Problem - how do I locate the controls when interfaces run in Beijing, Bangalore and Boston?
    • Result - synchronization and/or replication of security policy is problematic
    • Action - decentralized policy enforcement points and policy decision points.

• Interoperability
  – Finding interoperable vulns
    • XSS - Javascript is an equal opportunity offender
  – Fixing interoperable vulns
    • App servers, ESBs, and services are the attacker’s royal road. Interoperable access control can be leveraged across the enterprise
<SOAP:Envelope>
  <SOAP:Header>
    <WSSE:Security>
      <ds:Signature>
        <ds:Reference URI="#body"/>
      </ds:Signature>
    </WSSE:Security>
  </SOAP:Header>
  <SOAP:Body wsu:Id='body'>
    ...
  </SOAP:Body>
</SOAP:Envelope>

- Add signature token in header to sign message body

<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <ns1:echo xmlns:ns1="http://sample01.samples.rampart.apache.org">
      <param0>
        My Credit Card Number
      </param0>
    </ns1:echo>
  </soapenv:Body>
</soapenv:Envelope>

Encrypt sensitive data at the message level

<wss:Security xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
  <wss:EncryptedKey Id="EncKey0Id-5020592">
    <wss:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  </wss:EncryptedKey>
  <wss:CipherValue>
    XNQ0a4legiie5mWFx06CQQk2hIldYNmKroObue/LXs/VYtvaTgMbCujhGExDkqCvikU/Qc2/
    T6mso0WVTmBM138roga8jD
    +nS9yZ2f3c3CwoTh72h8wlL3D0DEe91iwJT9JBY1GXv7v9lyuxK0ooD0YECblhH74CpmTv3dBc
    +GQ=
  </wss:CipherValue>
</wss:Security>
## SOA Threat Model

<table>
<thead>
<tr>
<th>Threat</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Assume identity of client, server or request/response</td>
<td>Phishing attack to fool user into sending credentials to fake site</td>
</tr>
<tr>
<td>Tampering</td>
<td>Alter contents of request of response</td>
<td>Message integrity compromised to change parameters or values</td>
</tr>
<tr>
<td>Dispute</td>
<td>Dispute legitimate transaction</td>
<td>Illegitimately claiming a transaction was not completed</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Unauthorized release of data</td>
<td>Unencrypted message sniffed off the network</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Service not available to authorized users</td>
<td>System flooded by requests until web server fails</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Bypass authorization system</td>
<td>Attacker changes group membership</td>
</tr>
</tbody>
</table>

## STRIDE Threat Model

<table>
<thead>
<tr>
<th>Threat</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Assume identity of client, server or request/response</td>
<td>Phishing attack to fool user into sending credentials to fake site</td>
</tr>
<tr>
<td>Tampering</td>
<td>Alter contents of request of response</td>
<td>Message integrity compromised to change parameters or values</td>
</tr>
<tr>
<td>Dispute</td>
<td>Dispute legitimate transaction</td>
<td>Illegitimately claiming a transaction was not completed</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Unauthorized release of data</td>
<td>Unencrypted message sniffed off the network</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Service not available to authorized users</td>
<td>System flooded by requests until web server fails</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Bypass authorization system</td>
<td>Attacker changes group membership</td>
</tr>
</tbody>
</table>
## SOA Threat Model

<table>
<thead>
<tr>
<th>Threat</th>
<th>Security Service</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
<td>WS-Security, SAML</td>
</tr>
<tr>
<td>Tampering</td>
<td>Digital Signature</td>
<td>XML Signature, SSL/TLS</td>
</tr>
<tr>
<td>Dispute</td>
<td>Audit Logging</td>
<td>None</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Encryption</td>
<td>XML Encryption, SSL</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
<td>None</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Authorization, Input validation</td>
<td>None</td>
</tr>
</tbody>
</table>

## SOA Threat Model

<table>
<thead>
<tr>
<th>Threat</th>
<th>Security Service</th>
<th>Data</th>
<th>Method</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
<td>WS-Security</td>
<td>WS-Security</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Tampering</td>
<td>Digital Signature</td>
<td>XML Signature</td>
<td>None</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Dispute</td>
<td>Audit Logging</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Encryption</td>
<td>XML Encryption</td>
<td>None</td>
<td>SSL</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Authorization, Input validation</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
Trends

• Reusability
  – Reusable Findings & Fixes
  • Consider two bug findings
    – Session management bug: session state is passed around to every component, service and user. Makes for many high priority findings in audit report, also the fix is required on virtually every program
    – Data validation bug: Data access object (DAO) has a SQL injection hole. One major high priority finding in report. DAO used by many business logic classes, one fix location serves many classes
SOA Security Scorecard

<table>
<thead>
<tr>
<th>Description</th>
<th>Interaction 1</th>
<th>Interaction 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Confidentiality</td>
<td>Confidential channel</td>
<td></td>
</tr>
<tr>
<td>Transport Authentication</td>
<td>Authenticate channel usage</td>
<td></td>
</tr>
<tr>
<td>Transport Encoding</td>
<td>Encode for channel</td>
<td></td>
</tr>
<tr>
<td>Message authentication</td>
<td>Message authentication tokens &amp; verification</td>
<td></td>
</tr>
<tr>
<td>Message integrity</td>
<td>Integrity &amp; verification</td>
<td></td>
</tr>
<tr>
<td>Message confidentiality</td>
<td>Encrypt &amp; decrypt message</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>Authorize based on entitlement, permissions and roles</td>
<td></td>
</tr>
<tr>
<td>Schema validation</td>
<td>What schemas are used for validation</td>
<td></td>
</tr>
<tr>
<td>Content Validation</td>
<td>Black/white/graylist validation</td>
<td></td>
</tr>
</tbody>
</table>

SOA Security Scorecard

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Encoding</td>
<td>Encode message and document</td>
</tr>
<tr>
<td>Virus protection</td>
<td>Check for virus</td>
</tr>
<tr>
<td>Message size</td>
<td>Allowable size</td>
</tr>
<tr>
<td>Message throughput</td>
<td>Amount of message and throughput time</td>
</tr>
<tr>
<td>Identity, key, cert provision</td>
<td>Provisioning processes</td>
</tr>
<tr>
<td>Endpoint security profile</td>
<td>Security posture of endpoint</td>
</tr>
<tr>
<td>Audit logging</td>
<td>Audit log for services</td>
</tr>
<tr>
<td>Software engineering assurance</td>
<td>Assurance activities</td>
</tr>
<tr>
<td>XML Denial of Service protection</td>
<td>Availability services</td>
</tr>
<tr>
<td>Testing</td>
<td>Independent verification</td>
</tr>
</tbody>
</table>
Example Scale

• Token type
  – 0: no token
  – 1: hashed token
  – 2: hashed and signed token
  – 3: hashed and signed token from authoritative source

Example Scale

• Validation type
  – 0: no validation
  – 1: schema validation
  – 2: schema validation against hardened schema
  – 3: schema validation against standard, hardened schema
Putting it all together

- Use value assessment to focus time and effort
- Use scoring index to improve quality
REST Goals

<table>
<thead>
<tr>
<th>SOAP</th>
<th>REST</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML In, XML Out</td>
<td>HTTP-Get In, XML Out</td>
</tr>
<tr>
<td>Service or process centric</td>
<td>URI or resource centric</td>
</tr>
<tr>
<td>Transport neutral</td>
<td>Use HTTP</td>
</tr>
<tr>
<td>Many standards</td>
<td>Leverage existing infrastructure</td>
</tr>
</tbody>
</table>

RESTful Web Services - This is Web 2.0?

“The actual XML message is contained in the HTTP request and security is provided by HTTPS, which is the secure version of HTTP. This, in a nutshell, is virtually everything that a Web service user or creator needs to know about REST.”

Dion Hinchcliffe

http://webservices.sys-con.com/read/79282.htm
REST Server -- look Ma, no WSDL

```java
@WebServiceProvider()
@ServiceMode(value = Service.Mode.PAYLOAD)
public class RestSourcePayloadProvider implements Provider<DOMSource> {

    public DOMSource invoke(DOMSource request) {
        MessageContext mc = wsContext.getMessageContext();
        String path = (String)mc.get(Message.PATH_INFO);
        String query = (String)mc.get(Message.QUERY_STRING);
        String httpMethod = (String)mc.get(Message.HTTP_REQUEST_METHOD);

        if (httpMethod.equalsIgnoreCase("POST")) {
            return updateCustomer(request);
        } else if (httpMethod.equalsIgnoreCase("GET")) {
            if (path.equals("/customerservice/customer") && query == null) {
                return getAllCustomers();
            }
        }
    }
```
Rest Client

```java
public static void main(String[] args) throws Exception {
    QName serviceName = new QName
        ("http://apache.org/hello_world_xml_http/wrapped", "cutomerservice");
    QName portName = new QName
        ("http://apache.org/hello_world_xml_http/wrapped", "RestProviderPort");
    String endpointAddress = "http://localhost:9000/customerservice/"
        + "customer";
    URL url = new URL(endpointAddress + "?id=1234");
    InputStream in = url.openStream();
    Source source = new StreamSource(in);
    printSource(source);
    Service service = Service.create(serviceName);
    service.addPort(portName, HTTPBinding.HTTP_BINDING, endpointAddress);
    Dispatch<DOMSource> dispatcher = service.createDispatch(portName, DOMSource.class, Service.Mode.PAYLOAD);
    Map<String, Object> requestContext = dispatcher.getRequestContext();
    requestContext.put(MessageContext.HTTP_REQUEST_METHOD, new String("GET"));
    requestContext.put(MessageContext.QUERY_STRING, "id=1234");
    requestContext.put(MessageContext.PATH_INFO, path);
    DOMSource returnSource = dispatcher.invoke(null);
    printSource(returnSource);
}
```

Service oriented: Every service has its own interface and operations, but does not represent an entity

Resource oriented: entities or collections represented by a URI

Source (http://www.innoq.com/blog/st/2006/06/30/rest_vs_soap_oh_no_not_again.html)
REST Request Authentication

Summary of HMAC-SHA1 Request Authentication

1. You construct a request to AWS.
2. You use your Secret Access Key to calculate the request signature, a Keyed-Hashing for Message Authentication code (HMAC) with an SHA1 hash function, as defined in the next section of this topic.
3. You send the request data, the signature, and your Access Key ID to AWS.
4. AWS uses the Access Key ID to look up the Secret Access Key.
5. AWS generates a signature from the request data and the Secret Access Key using the same algorithm you used to calculate the signature in the request.
6. If the signature generated by AWS matches the one you sent in the request, the request is considered to be authentic. If the comparison fails, the request is discarded, and AWS returns an error response.

(note: append timestamp to request to limit replays to 15 minute window)


Rest Request Authentication

"Authorization: AWS " + AWSAccessKeyId + ":" + base64(hmac-sha1(VERB + "\n" + CONTENT-MD5 + "\n" + CONTENT-TYPE + "\n" + DATE + "\n" + CanonicalizedAmzHeaders + "\n" + CanonicalizedResource))

Example:

PUT /quotes/nelson HTTP/1.0
Authorization: AWS 44CF95900D6BF252F707:jZNOcbfWmD/A/f3h8VvXzZjM2HU=
Content-Md5: c8fdb181845a4ca69b6fc737b3581d7b
Content-Type: text/html
Date: Thu, 17 Nov 2005 18:49:58 GMT
X-Amz-Meta-Author: foo@bar.com
X-Amz-Magic: abracadabra
Rest Threat Model

<table>
<thead>
<tr>
<th>Threat</th>
<th>Security Service</th>
<th>Data</th>
<th>Method</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authentication</td>
<td>XML Signature</td>
<td>None</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Tampering</td>
<td>Digital Signature</td>
<td>XML Signature</td>
<td>None</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>Dispute</td>
<td>Audit Logging</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Information Disclosure</td>
<td>Encryption</td>
<td>XML Encryption</td>
<td>None</td>
<td>SSL</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Authorization, Input validation</td>
<td>Oauth</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

Security Architecture Elements

- **Enablement Services**: services managing business enabling such as capabilities provisioning, federation, identity, and secure integration
- **AAA Services**: Authentication, Authorization, and Auditing services
- **Defensive Services**: conservative services that deal with threats and vulnerabilities
AAA Services: SAML

SAML Assertion

Headers & Control Information
SAML Issuer
Timers
XML Encryption spec supports:
- Block Encryption: TRIPLE
  DES, AES-128, AES-256
- Key Transport: RSA-v1.5,
  RSA-OAEP
Digital Signature spec supports:
- Digest: SHA1
- MAC: HMAC-SHA1
XML Canonicalization:
- CanonicalXML (Without comments)
- Transform: Enveloped
Signature
Signature: RSAwithSHA1
(recommended in XML
Signature but needed for
interoperability)
SAML Assertion

Authentication Statement
How was the user authenticated

Attribute Statement
Is there any additional identity information about the user

Authorization Decision Statement
Have any authorization decisions been made for this user

SAML 2.0

Define SP and IDP Metadata
Attributes and Authorization Policy Decision Point
Supported identifier profiles, protocols, and attributes

Support general purpose use cases, such as Browser SSO and attribute exchange

Request/Response to query for assertion or SAML binding action

Support for various AuthN protocols and standards, including Kerberos and smart cards

Assertion
Protocol
Binding
Message
AuthN Context
  <saml:Issuer>http://authority.example.com</saml:Issuer>
  <!-- signature by the issuer over the assertion -->
  <ds:Signature>...</ds:Signature>
  <saml:Subject>
    <saml:NameID format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent">jygH5F901</saml:NameID>
    </saml:Subject>
  <saml:AuthnStatement AuthnInstant="2005-04-01T16:57:30.000Z" SessionIndex="6345789">
    <saml:AuthnContext>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>

Source Paul Madsen http://www.xml.com/pub/a/2005/01/12/saml2.html

SAML Producer Consumer Model

  ©2005-7 Arctec Group
Defensive Architecture: Security Gateway

- Vulnerability
  - In SOAP and Rest style Web services there is no default authentication, messages are typically sent in XML over HTTP and contain nothing that can be used to perform authentication.
  - Simply applying general purpose security standards like WS-Security is not adequate, the WS-Security Username token may pass the user's password in plaintext form. For example:

```xml
<SOAP>
  <SOAPHeader>
    <wsse:Username>Joe</wsse:Username>
    <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">MyPassword</wsse:Password>
  </SOAPHeader>
</SOAP>
```
**Vulnerability**

- The next step beyond Username Token with Password in cleartext is to look at hashing the password

```xml
<wsse:Username>Joe</wsse:Username>
<wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordDigest">E9rKWg/JSBzmaQufwyf0BRjc3w=</wsse:Password>
```

- This token is marginally stronger, but also lacks a timestamp and nonce so may be vulnerable to message replay and other attacks. Further, if the password is hashed, it's likely there is a cleartext password sitting somewhere in the system that generated it. WS-Security provides a general purpose framework for transmitting claims, but the standard is treated differently in practice in implementation.
XML Security Gateway Pattern

- Context: The primary goal of Web services is to solve interoperability and integration problems. Web services traverse multiple technologies and runtimes.
- Problem: Web service requesters and providers do not agree upon binary runtimes like J2EE, instead they agree upon service contracts, message exchange patterns, and schema. Service and message level authentication, authorization, and auditing services for Web services are not delivered by a single container, rather these services must span technical and organizational boundaries
• Solution: Use a XML Security Gateway to provide decentralized security services for Web services

```xml
    <saml:AuthorizationDecisionStatement Decision="Permit" Resource="http://host/service">
      <saml:Subject>
        <saml:NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName">Test</saml:NameIdentifier>
        <saml:Action>getCustomerDetails</saml:Action>
      </saml:Subject>
      <dsig:SignatureValue>V6pRh05Snrv58X7+WXIdNv1rOHvkAUMV14Y27KfG/ jDLm5bStrBDE3tAM4rI6mAL U+gt20yrS8rb+ALlpxNKouxZMvdlcJzrrQg1jt339DvYL6QR323KvDy7n6W16Pm w8Yo7actG8IFW5pV5ocPs50Vv+OGHafYTG6lbQv</dsig:SignatureValue>
      <dsig:KeyInfo Id="Id-000001129354af1d-0000000000000002"/>
    </saml:AuthorizationDecisionStatement>
  </saml:Assertion>
</wsse:Security>
</soap:Header>
<soap:Body>
  <ns0:getCustomerDetails xmlns:ns0="http://servicehost/">
    <customernumber>1234</customernumber>
  </ns0:getCustomerDetails>
</soap:Body>
</soap:Envelope>
```
Enablement

Policy

“Security should depend on policy not topology.”
-Bill Gates Feb. 6, 2007
WS-Policy

• WS-Policy Framework
    • Operations - all, exactlyone, oneormore
    • Usage - required, rejected, optional
  – WS-PolicyAttachment - standard for attaching policy assertions to resources, for example WSDL

WS-Security Policy

• Part of WS-PolicyFramework; provides declarative security requirements for service
• Can be requested standalone or through WS-Mex
• Sample usages
  – Define allowed security token types, issues
  – Defines message integrity policy through allowed XML Digital Signature algorithms & specifying what message elements are to be signed
  – Defines allowed message processing schemes & lifetimes
Transport Binding Assertions

```xml
  <wsp:ExactlyOne>
    <wsp:All>
        <wsp:Policy>
          <sp:TransportToken>
            <wsp:Policy>
              <sp:HttpsToken RequireClientCertificate="false"/>
            </wsp:Policy>
            </sp:TransportToken>
          </wsp:Policy>
        </sp:TransportBinding>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
</wsp:Policy>
```

...
Asymmetric Binding Assertion

```xml
  <sp:InitiatorToken>
    <wsp:Policy>
      <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
        <wsp:Policy>
          <sp:WssX509V3Token10/>
        </wsp:Policy>
      </sp:X509Token>
      ...
    </wsp:Policy>
  </sp:InitiatorToken>
  ...
</sp:AsymmetricBinding>
```

Asymmetric Binding Assertion (cont.)

```xml
...<sp:RecipientToken>
  <wsp:Policy>
    <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">
      <wsp:Policy>
        <sp:WssX509V3Token10/>
      </wsp:Policy>
    </sp:X509Token>
    ...
  </wsp:Policy>

  <sp:AlgorithmSuite>
    <wsp:Policy>
      <sp:TripleDesRsa15/>
    </wsp:Policy>
    ...
  </sp:AlgorithmSuite>

  ...
</sp:RecipientToken>
```
Summary

• WS-SecurityPolicy provides granular control over security policy at the transport (non-message level), message level security, and allowable crypto and token types

• WS-SecurityPolicy may be used to enforce policy decisions and as such these files and assertions become part of the access control architecture and require a high level of protection - through digital signature and verification