Improving performance for security enabled web services
- Dr. Colm Ó hÉigeartaigh
Agenda

- Introduction to Apache CXF
- WS-Security in CXF 3.0.0
- Securing Attachments in CXF 3.0.0
- RS-Security in CXF 3.0.0
- Some empirical data
- Using Single Sign-On (SSO)
Speaker Introduction

Apache CXF

Apache Syncope

Apache Santuario

Apache Webservices
Apache CXF

- One of the leading web service frameworks.
- Supports JAX-WS and JAX-RS frontend APIs
- Protocols: SOAP, XML/HTTP, RESTful HTTP, CORBA, etc.
- Transports: HTTP, JMS, JBI, etc.
- Comprehensive WS standards support.
- Security Services: STS, XKMS, Fediz.
- Strong OSGi support.
Apache CXF Stats

- Founded in 2006 as a merger of Celtix + XFire.
- Apache TLP releases go from 2.0.6 to the current 2.7.10 / 3.0.0-milestone2.
- 33 committers, 22 of whom are PMC members.
- Embedded in other Apache projects such as Apache Syncope, Apache Camel, Apache TomEE+.
- Used in industry products such as JBoss Web Services, Jboss Fuse, Talend ESB, etc.
JAX-WS

- A service is typically defined by a WSDL document
- Java code generated by “WSDL2Java” functionality
- Alternatively, can start with code + use annotations
- Typically a SOAP binding is used over HTTP
- SOAP Body contains service payload
- SOAP Header contains service metadata
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://www.example...</Action>
  </soap:Header>
  <soap:Body>
    <ns2:DoubleIt xmlns:ns2="http://www.example.org/schema/DoubleIt">
      <numberToDouble>25</numberToDouble>
    </ns2:DoubleIt>
  </soap:Body>
</soap:Envelope>
JAX-RS

- Web Services using Representational State Transfer (REST) paradigm.
- Can use WADL to define the service, but typically code + annotations are used
- Messages can be marshalled/unmarshalled to/from Java Objects using JAXB
- Messages in XML/JSON format.
Annotations Example

```java
@Path("/customerservice/")
@Produces("application/xml")
public class CustomerService {

    public CustomerService() {
    }

    @GET
    public Customers getCustomers() {
        .......
    }

    @GET
    @Path("/customers/{id}")
    @Produces("application/json")
    public Customer getCustomer(@PathParam("id") String id) {
        .......
    }
}
```
WS-Security

- A set of OASIS specifications to secure SOAP Messages
- Message Confidentiality (XML Encryption)
- Message Integrity (XML Signature)
- Client Authentication via tokens (Username Tokens, Kerberos Tokens, SAML Tokens, Asymmetric Signature Certificates/Public Keys).
Secured SOAP Envelope

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1">
      <xenc:EncryptedKey Id="EK-...">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-opac-mgf1p"/>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier>u4QVnV7jQhG8h2GiTSVJyB2g9c=</wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </xenc:EncryptedKey>
      <xenc:CipherData>
        <xenc:CipherValue>dBWhc...</xenc:CipherValue>
      </xenc:CipherData>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#ED-..."/>
      </xenc:ReferenceList>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <xenc:EncryptedData Id="ED-..." Type="http://www.w3.org/2001/04/xmlenc#Content">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference><wsse:Reference URI="#EK-1..."/></wsse:SecurityTokenReference>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>t2UI7i3M...</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```
WS-SecurityPolicy

- WS-SecurityPolicy can be used to configure WS-Security via a WS-Policy expression.
- By embedding the policy in a WSDL, a service can publish security requirements to a client.
- Client/Service only need to configure usernames, passwords, keys, etc.
- Requests are validated against the set of applicable policies.
<sp:TransportBinding>
    <wsp:Policy>
        <sp:TransportToken>
            <wsp:Policy>
                <sp:HttpsToken><wsp:Policy/></sp:HttpsToken>
            </wsp:Policy>
        </sp:TransportToken>
        <sp:IncludeTimestamp/>
        <sp:AlgorithmSuite>
            <wsp:Policy><sp:Basic128/></wsp:Policy>
        </sp:AlgorithmSuite>
    </wsp:Policy>
</sp:TransportBinding>
<sp:SupportingTokens>
    <wsp:Policy>
        <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/...|AlwaysToRecipient">
            <wsp:Policy><sp:WssUsernameToken10/></wsp:Policy>
        </sp:UsernameToken>
    </wsp:Policy>
</sp:SupportingTokens>
WS-Security @ Apache

- Apache Santuario: XML Signature + XML Encryption
- Apache WSS4J: WS-Security layer built on top of Santuario
- Apache CXF / Apache Axis/Rampart: Web Services stacks that include WSS4J – WS-SecurityPolicy support, WS-Trust, WS-SecureConversation, etc.
A familiar problem...

(Time per 1000 requests (s))

- Small Msg
- Large Msg

Plaintext
Signed + Encrypted
Security is Expensive

- There is a large performance penalty associated with using WS-Security.
- This is partly due to the work involved in signing and encrypting (in particular using XML).
- However, a large reason is due to the fact that up to now, WS-Security processing requires DOM.
- This requires a lot of memory for large requests
- Also, a StAX-enabled stack such as CXF needs to convert the request into DOM
Streaming WS-Security

- A WS-Security implementation based on StAX would solve the problem of large memory requirements and having to convert to DOM.
- However, there are huge difficulties with porting things like XML canonicalization to use a streaming approach.
- 2011: Problem solved by Marc Giger donating his SWSSF project to Apache, a streaming WS-Security prototype based on WSS4J.
SWSSF @ Apache

• Rather than create a new project, SWSSF has been integrated into the existing projects.
• The XML Signature + Encryption parts have been added to Apache Santuario 2.0.0.
• The WS-Security parts have been added to Apache WSS4J 2.0.0.
• WSS4J now has two WS-Security stacks, one based on DOM and one on StAX.
CXF Integration

• The new StAX code is fully integrated into CXF
• It uses the exact same configuration as for the DOM code
• New interceptors: WSS4JStax(Out|In)Interceptor
• Works with WS-SecurityPolicy - StAX functionality enabled by a boolean configuration property (“ws-security.enable.streaming”)
• DOM functionality is enabled by default for WS-SecurityPolicy
Real-time validation

- Apache CXF parses the set of WSS4J results + evaluates the set of applicable WS-SecurityPolicy policies against them.
- The new StAX implementation does real-time validation of the policies while it is evaluating a request.
- SecurityEvents are generated during processing.
- This has performance gains and is more resistant to Denial of Service (DoS) style attacks.
Performance

• The StAX WS-Security stack uses far less memory for large requests (see Empirical Data section)
• It should be more efficient for a service handling many simultaneous requests as a result
• It performs better in some scenarios than the DOM stack, and worse in others
• Profiling and future optimisations will hopefully improve performance to a point where we can switch the default stack in CXF
What's not supported?

- XPath evaluation
- “Strict” Layout validation
- Policy combinations that require two separate Encryption actions (EncryptBeforeSigning + EncryptSignature)
- Policy combinations that require two separate Signature actions (e.g. Endorsing tokens with (a)symmetric bindings – with some exceptions).
WSS4J 2.0.0

- Lots of new features apart from StAX implementation
- New consolidated WS-SecurityPolicy model
- Support for securing message attachments
- Support for caching based on EhCache
- Support for encrypting passwords in Crypto properties files using Jasypt
Securing attachments in CXF
3.0.0
Securing attachments

- Signing/encrypting message attachments not supported prior to CXF 3.0.0.
- WSS4J 2.0.0 supports the WS-Security SOAP Messages with Attachments Profile.
- If a "<sp:Attachments />" policy is used as a (Signed|Encrypted)Parts in CXF 3.0.0, all attachments are automatically secured.
- There are also policies to only sign the content or to include the attachment headers.
Example

Payload: --uuid:e49a5dc5-689d-4879-b51e-d1e192a5276d^M
Content-Type: text/xml; charset=UTF-8; type="text/xml"^M
Content-Transfer-Encoding: binary^M
Content-ID: <root.message@cxo.apache.org>

<soap:Envelope ....>
  <xenc:EncryptedData...>
    <xenc:CipherData>
      <xenc:CipherReference
        URI="cid:attachment=e02d4dce-bcd4-45ab-99c6-824a848697b8@apache.org">
        <xenc:Transforms>
          <ds:Transform Algorithm="...-SwAPProfile-1.1#Attachment-Ciphertext-Transform="/>
          </xenc:Transforms>
        </xenc:CipherReference>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Envelope>
--uuid:e49a5dc5-689d-4879-b51e-d1e192a5276d^M
Content-Type: application/octet-stream^M
Content-Transfer-Encoding: binary^M
Content-ID: <attachment=e02d4dce-bcd4-45ab-99c6-824a848697b8@apache.org>^M
^M
{%}мёдо^Drёл<םי^E....
--uuid:e49a5dc5-689d-4879-b51e-d1e192a5276d--
Using MTOM

- If MTOM is enabled with WS-Security, attachments are inlined before the SOAP Body is secured.
- Signing/encrypting using MTOM is targeted for CXF 3.0.1.
- However, the cost associated with BASE-64 encoding the attachment + inlining it for signature digest calculation may make the SwA approach more efficient.
- CXF 3.0.0 has a minor efficiency gain not to inline the attachments with MTOM for most TransportBinding use-cases.
RS-Security in CXF 3.0.0
RS-Security

- CXF supports XML Signature + Encryption for JAX-RS clients and endpoints as well.
- XML Signature options: Enveloped, Enveloping, Detached.
- Separate interceptors for Signature + Encryption, that can be chained.
- Using XML Signature with PKI allows an alternative to the standard HTTP/BA over TLS or TLS with client auth.
Sample signed request
Streaming RS-Security

- It's possible to use the new StAX functionality for JAX-RS as well in CXF 3.0.0.
- New interceptors: XmlSec(Out|In)Interceptor
- XML Signature (enveloped only) + Encryption supported.
- Testcase: https://github.com/coheigea/testcases/tree/master/apache/cxf/cxf-jaxrs-xmlsec
Some empirical data...
Benchmarks I

HEAP memory consumption during encryption

Memory [MB]

0 50000 100000 150000 200000 250000 300000 350000 400000 450000
Number of XML start elements

StAX  DOM
Benchmarks II

HEAP memory consumption during decryption

Memory [MB]

0 50 100 150 200 250 300 350

Number of XML start elements
0 50000 100000 150000 200000 250000 300000 350000 400000 450000

StAX  ————
DOM  ————

StAX
DOM
Benchmarks IV

HEAP memory consumption during signature verification

Memory [MB]

Number of XML start elements

StAX  DOM

0  50000  100000  150000  200000  250000  300000  350000  400000  450000
Using Single Sign-On (SSO)
Single Sign-On (SSO)

- Thus far we have focused on securing messages
- However, client authentication can also be expensive...
- This is where Single Sign-On (SSO) comes in
- The client “signs-on” to a centralized authentication service of some kind, and retains a resulting token for any subsequent authentication (until the user signs out).
SSO using WS-SecConv

- A really simple way supported in CXF for SSO is to use WS-SecureConversation.
- A rudimentary STS is embedded with a CXF endpoint
- The client authenticates and receives a token + negotiated secret.
- The client signs the request using the secret + references the token in any subsequent request.
- Testcase (SSOTest):
  https://github.com/coheigea/testcases/tree/master/apache/cxf/cxf-shiro
SSO using an STS

- CXF ships with an advanced SecurityTokenService (STS)
- The client authenticates to the STS + receives a SAML Token.
- The client caches the token + re-uses it until expiry.
- Roles/claims are embedded in the token for authorization
- Testcase (SSOTest):
  https://github.com/coheigea/testcases/tree/master/apache/cxf/cxf-sts
SSO using SAML SSO

- CXF supports SSO via the SAML SSO Web Profile
- A JAX-RS filter can redirect a service request to an IdP
- The IdP authenticates the client and redirects to the service
- Authenticated state saved as a cookie
- The SAML Assertion is also saved to allow for role retrieval
- Testcase
Questions