

## Java Web サービス: Axis2 WS-Security の基本

### Rampart を Axis2 にインストールして UsernameToken の処理を実装する方法

Dennis Sosnoski

Consultant

Sosnoski Software Associates Ltd

2009年 5月 26日

Rampart セキュリティー・モジュールを Apache Axis2 にインストールして、Web サービスで WS-Security の機能を使用できるようにする方法を学んでください。今回、Axis2 での WS-Security および WS-SecurityPolicy の使用を焦点に再開した Dennis Sosnoski の連載「[Java Web サービス](#)」では、単純な最初のステップとして UsernameToken を取り上げます。連載の以降の記事では、Axis2 と Rampart によって実装する WS-Security、WS-SecurityPolicy について深く掘り下げていきます。

[このシリーズの他の記事を見る](#)

セキュリティは、多くのエンタープライズ・サービスにとって重要な要件であるとともに、自力で実装を試みるには危険な分野でもあります。ちょっとした見落としや目立たないミスが重大な脆弱性につながるからです。このような特性からセキュリティの処理を標準化することが求められており、多くの専門家たちが標準の作成に貢献し、個人での見落としがなくなるようにしています。SOAP ベースの Web サービスでは、セキュリティのニーズに対応するために広く支持されている WS-Security とこれに関連する標準を適用して、それぞれのサービスに適したセキュリティを構成することができます。

Apache Axis2 では、Rampart モジュールという手段で WS-Security と関連標準をサポートします（「[参考文献](#)」を参照）。この記事では、Rampart を Axis2 にインストールして構成し、サービス・リクエストでユーザー名とパスワードを送信する際の基本セキュリティ機能として利用する方法を説明します。この連載の以降の記事では、Rampart をさらに高度な形のセキュリティとして利用する方法を学びます。

#### この連載について

Web サービスは、エンタープライズ・コンピューティングにおいて Java™ 技術が担う大きな役割の一部です。この連載では、XML および Web サービスのコンサルタントである Dennis Sosnoski が、Web サービスを使用する Java 開発者にとって重要になる主要なフレームワークと技術について説明します。この連載から、現場での最新の開発情報を入手して、

それらを皆さんのプログラミング・プロジェクトにどのように利用できるかを知っておいてください。

## WS-Security

WS-Security は、SOAP による Web サービス・メッセージの交換にセキュリティーを適用するための標準です (「[参考文献](#)」を参照)。この標準では、SOAP メッセージ・ヘッダー要素を使用してメッセージにセキュリティー情報を追加します。セキュリティー情報の形態として取られるのは、各種の要求 (要求には名前、ID、キー、グループ、特権、機能などを含めることができます) を暗号化およびデジタル署名に関する情報と併せて伝えるトークンです。WS-Security がサポートするトークンのフォーマット、トラスト・ドメイン、署名フォーマット、暗号化技術はいずれも複数あります。そのためほとんどの場合は、ヘッダー情報にそれぞれのコンポーネントに特定のフォーマットおよびアルゴリズム ID を含めなければなりません。セキュリティー情報を追加した結果、ヘッダー情報は複雑な構造になります。この複雑な構造を [リスト 1](#) (これでも大幅に編集してあります) の署名および暗号化が設定されたサンプル・メッセージに示します。

### リスト 1. 署名および暗号化が設定されたサンプル・メッセージ

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" ...>
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1">
      <wsu:Timestamp wsu:Id="Timestamp-d2e3c4aa-da82-4138-973d-66b596d66b2f">
        <wsu:Created>2006-07-11T21:59:32Z</wsu:Created>
        <wsu:Expires>2006-07-12T06:19:32Z</wsu:Expires>
      </wsu:Timestamp>
      <wsse:BinarySecurityToken ValueType="...-x509-token-profile-1.0#X509v3"
        EncodingType="...-wss-soap-message-security-1.0#Base64Binary"
        xmlns:wsu="...oasis-200401-wss-wssecurity-utility-1.0.xsd"
        wsu:Id="SecurityToken-faa295...">MIIEC56MQswCQY...</wsse:BinarySecurityToken>
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmenc#">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5" />
        <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier ValueType="
              ...#X509SubjectKeyIdentifier">LlYsHyhNn0VA9Aj7...</wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>g+A2WJhsoGBKUydZ9Za...</xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#EncryptedContent-ba0556c3-d443-4f34-bcd1-14cbc32cd689" />
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        <SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
          <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <Reference URI="#Id-c80f735c-62e9-4001-8094-702a4605e429">
            <Transforms>
              <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>lKjc5nyLQDZAIu/hZb4B6mLquow=</DigestValue>
          </Reference>
          ...
        </SignedInfo>
        <SignatureValue>TiLmWvlz3mswinLVQn58BgYS0368...</SignatureValue>
      </KeyInfo>
```

```
<wsse:SecurityTokenReference>
  <wsse:Reference URI="#SecurityToken-faa295..."
    ValueType="...-x509-token-profile-1.0#X509v3" />
</wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
</wsse:Security>
</soap:Header>
<soap:Body wsu:Id="Id-8db9ff44-7bef-4737-8091-cdac51a34db8">
  <xenc:EncryptedData Id="EncryptedContent-ba05..."
    Type="http://www.w3.org/2001/04/xmlenc#Content"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
    <xenc:CipherData>
      <xenc:CipherValue>mirmi0KuFEEI56eu2U3cICz...</xenc:CipherValue>
    </xenc:CipherData>
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>
```

この記事に記載する WS-Security ヘッダーの例は、トークンが 1 つしかない単純なものです。リスト 1 に示したような複雑な構造については、連載の次の記事で詳しく説明します。

WS-Security は実際の SOAP メッセージの交換に適用されます。Web サービスの実装では、WS-Security が受信メッセージに正しく適用されているかどうかを確認することができますが、クライアントは Web サービスを使用するためには何を実装しなければならないのかを事前に知っておく必要があります。WS-Security は複雑であり、サポートされるオプションも多いので、WS-Security を適用するためにテキスト記述を使用するだけでも難しいことがあります。また、手動での WS-Security 処理の構成はエラーの原因になりかねないため、Web サービスの拡張要件を指定するための汎用構造である WS-Policy と、WS-Security のサポート専用 WS-Policy を拡張した WS-SecurityPolicy が用意されています。この 2 つの標準が合わさって、コンピューターが読み取れる形式での WS-Security 要件の記述をサポートします。WS-Policy と WS-SecurityPolicy の情報は単独で使用することも、直接 WSDL (Web Services Description Language) 文書に組み込んでサービスの要件に合わせて Web サービス・フレームワークが自動的に構成されるようにすることも可能です。

## Rampart の紹介

Rampart は、WS-Security、WS-SecurityPolicy、WS-SecureConversation、および WS-Trust をサポートする Axis2 用のセキュリティー・モジュールです。今回の記事では Rampart に備わった WS-Security と WS-SecurityPolicy の機能だけを取り上げますが、今後の記事ではこの他の機能についても取り上げる予定です。

Rampart は 1 つのモジュールとして実装されるため (実際には rampart.mar と rahas.mar というモジュールで構成されています)、Axis2 処理フレームワークに組み込むと、インバウンド処理とアウトバウンド処理の特定の時点でメッセージをインターセプトします。そしてメッセージをチェックするか、または必要に合わせてメッセージを変更することによって、その役割を果たします。

## Rampart のインストール方法

Rampart には複数の .jar ファイル (ディストリビューションの lib ディレクトリー内) と 2 つの .mar モジュール・ファイル (dist ディレクトリー内) が付属しています。Rampart を Axis2 で使用するに

は、.jar ファイルをクラスパスに追加し、.mar ファイルをクラスパスまたは Axis2 リポジトリ構造に追加する必要があります。

Rampart の .jar ファイルと .mar ファイルを処理するのに最も簡単な方法は、これらのファイルを Axis2 システムのなかに追加することです。Rampart の lib ディレクトリーにある .jar ファイルは Axis2 の lib ディレクトリーに、Rampart の dist ディレクトリーにある .mar ファイルは Axis2 の repository/modules ディレクトリーに直接コピーするだけで済みます (Rampart の samples ディレクトリーにある Ant の build.xml を使ってファイルを Axis2 システムにコピーすることもできます。その場合、`AXIS2_HOME` 環境変数を Axis2 インストール・ディレクトリーに設定し、Rampart の samples ディレクトリーを指定して開いたコンソールから `ant` を実行するだけです)。

また多くの WS-Security 機能では、Bouncy Castle セキュリティー・プロバイダーを JVM セキュリティー構成に追加し、Bouncy Castle の .jar ファイルを Axis2 システムに追加する必要があります。このステップは今回の記事で学ぶ UsernameToken には不要ですが、連載の以降の記事で他のセキュリティ機能について説明するために必要です。一部のセキュリティ・アルゴリズムの特許に関する問題から、Bouncy Castle の .jar ファイルは Rampart とは別のダウンロードとなっています (「[参考文献](#)」を参照)。ご使用の Java ランタイムに対応したバージョンの .jar ファイルをダウンロードして、その .jar ファイルを Axis2 の lib ディレクトリーに追加してください。次に、Java システムのセキュリティ・ポリシーが Bouncy Castle コードを使用するように変更する必要があります。そのための作業は、Java ランタイムの lib/security ディレクトリーにある `java.security` ファイルに 1 行追加することです。このファイルで、複数の異なる `security.provider` 行があるセクションを見つけて以下の行を追加します。

```
security.provider.99=org.bouncycastle.jce.provider.BouncyCastleProvider
```

ファイル内で `security.provider` 行がどのような順序で記述されているかは問題になりませんが、上記の行はあらかじめ定義されているセキュリティ・プロバイダー実装の後に追加することをお勧めします。

Axis2 サーバー・システムで Rampart のコードを使用するには、新しい `axis2.war` ファイルを作成し、そのファイルに Rampart から追加した .jar および .mar ファイルを含める必要があります。webapp ディレクトリーに提供されている Ant の build.xml を使用すると、`axis2.war` を作成することができます。その場合には、このファイルの終わり近くにある `<exclude name="axis2-codegen*.jar"/>` という行を削除した上で、Axis2 の webapp ディレクトリーを指定して開いたコンソールから `ant` を実行します。build.xml の実行が完了すると、Axis2 システムの dist ディレクトリーに `axis2.war` Web アプリケーションが作成されているはずです。

## サンプル・アプリケーション

サンプル・コード (「[ダウンロード](#)」を参照) に提供されているアプリケーションは、「[Axis2 でのデータ・バインディング](#)」で Axis2 に異なるデータ・バインディングを使用する方法を説明するために用いたアプリケーションをベースにしたものです。今回の記事と Axis2 での WS-Security のサポートに関する以降の記事のために、アプリケーションの操作を `getBook`、`addBook`、`getBooksByType` の 3 つだけに減らしました。簡単のため、ADB (Axis Data Binding) バージョンのコードしか提供していませんが、ADB が Axis2 で WS-Security を操作する上での要件になっているわけではありません。Rampart は、コードが使用するデータ・バインディ

ングの手法に依存しないレベルで WS-Security を実装するため、Axis2がサポートするデータ・バインディングの形であれば、どのデータ・バインディングでも有効に機能します。

サンプル・コードのルート・ディレクトリーは `jws04code` です。このディレクトリーのなかには、Ant の `build.xml` および `build.properties` ファイル、サンプル・アプリケーション用のサービス定義を指定する `library.wsdl` ファイル、クライアント・サイドのロギングを構成するために使用する `log4j.properties` ファイル、そしてプロパティーを定義する複数の XML ファイル (いずれも、`XXX-policy-client.xml` または `XXX-policy-server.xml` というファイル名) があります。サンプル・アプリケーションの操作を構成するのは `build.properties` ファイルです。リスト 2 に、この記事に付属のプロパティー・ファイルを記載します。

## リスト 2. 付属の build.properties ファイル

```
# set axis-home to your Axis2 installation directory
axis-home=PATH_TO_AXIS2_INSTALLATION
# set the connection protocol to be used to access services (http or https)
protocol=http
# set the name of the service host
host-name=localhost
# set the port for accessing the services (change this for monitoring)
host-port=8080
# set the base path for accessing all services on the host
base-path=/axis2/services/
# set the name of the policy file to be used by the client
client-policy=plain-policy-client.xml
# set the name of the policy file to be used by the server
server-policy=plain-policy-server.xml
```

サンプルを試してみる前に、`build.properties` ファイルを編集して (前のセクションで Rampart を追加した) Axis2 システムの実際のパスを設定してください。サーバーに別のホストまたはポート番号を使用している場合は、`host-name` と `host-port` の値も変更する必要があります。その他の値については、この記事で後ほど説明します。

## WS-Security を試してみる

WS-Security で定義しているセキュリティー・トークンには、コア仕様に含まれるトークン、仕様のプラグイン拡張としてプロファイルで定義されているトークンを含め、さまざまなタイプがあります。しかもこれらのトークンには、トークンの構成方法、使用方法についても多数のオプションがあります。この記事では Axis2 で Rampart を構成および使用する方法を焦点としているので、ここでは最も単純で最も便利なトークンだけを例に用いることにします。それは、`UsernameToken` プロファイルによって定義される `UsernameToken` です。

### UsernameToken の WS-SecurityPolicy

`UsernameToken` の目的は、WS-Security ヘッダーの一部としてユーザー名とパスワード情報を伝えることだけです。最も基本的な形の `UsernameToken` は、ユーザー名とパスワードの両方を平文で送信します。セキュリティーの点からすると最適な方法ではありませんが (ただし、この方法をセキュアな接続で使用するのであれば何も問題はありません)、送信される内容を簡単に確認できるこのトークンは、出発点としては便利です。

テキストとして送信される `UsernameToken` の WS-SecurityPolicy 構成は、リスト 3 のとおりの単純なものにすることができます。このポリシー (ここではページの幅に合わせて 1 行で記述すると



ころを 2 行に分けて記載してありますが、実際には 2 行にわたって記述することはできません) は、WS-SecurityPolicy の UsernameToken に関する設定内容を囲む標準 WS-Policy ラッパー (wsp プレフィックスを使用する要素) で構成されています。

### リスト 3. 平文の UsernameToken の WS-SecurityPolicy

```
<wsp:Policy wsu:Id="UsernameToken" xmlns:wsu="
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SupportingTokens
        xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
            ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"/>
        </wsp:Policy>
      </sp:SupportingTokens>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

リスト 3 の UsernameToken は、IncludeToken 属性を使用して、トークンを組み込むメッセージ・フローのタイプを指定しています。この例の場合、リクエストの開始側 (つまり、クライアント) からリクエストの受信側 (サーバー) へのすべてのメッセージ・フローがこの指定したタイプになっています。IncludeToken 属性に定義された他の値を使って、トークンの他の使用方法を指定することもできますが、一般的に UsernameToken に妥当な値は、この例で使用している値だけです。

### ポリシーの適用

WS-Policy と WS-SecurityPolicy は、WSDL サービス定義のなかに組み込めるようになっていいます。その場合には、参照を使用してポリシーを <wsdl:binding>、<wsdl:binding>/<wsdl:operation>、または <wsdl:message> 定義の 1 つまたは複数に関連付けます。Axis2 1.4.X は WSDL に組み込まれたポリシーの前処理を実装しますが、Axis2 1.4.1 の時点ではまだ堅牢な実装にはなっていません。そのためこの記事ではポリシーを直接クライアントとサーバーに追加し、1.4.1 コードと互換するようにしています。

### サーバー・サイドのポリシー処理

サーバー・サイドにポリシーを適用するには、Axis2 のそれぞれの .aar サービス・アーカイブに含まれる services.xml 構成ファイルにポリシーを追加します。ポリシーを <service> 要素の直接の子として追加することで、該当するサービスで定義されたすべての操作に適用することができます。また、services.xml には <module> 要素を追加する必要もあります。これは、Axis2 に Rampart モジュールをサービスの構成に組み込むよう指示するためです。リスト 4 に、サンプル・アプリケーションで使用するように編集した services.xml のバージョンを記載します。追加されたモジュール参照とポリシー情報は太字で示してあります。

### リスト 4. ポリシーを組み込んだ services.xml

```
<serviceGroup>
  <service name="library-username">
    <messageReceivers>
      <messageReceiver
```

```

        class="com.sosnoski.ws.library.adb.LibraryUsernameMessageReceiverInOut"
        mep="http://www.w3.org/ns/wsdl/in-out"/>
</messageReceivers>
<parameter
    name="ServiceClass">com.sosnoski.ws.library.adb.LibraryUsernameImpl</parameter>
<parameter name="useOriginalwsdl">true</parameter>
<parameter name="modifyUserWSDLPortAddress">true</parameter>
<operation mep="http://www.w3.org/ns/wsdl/in-out" name="getBook"
    namespace="http://ws.sosnoski.com/library/wsdl">
    <actionMapping>urn:getBook</actionMapping>
    <outputActionMapping>http://.../getBookResponse</outputActionMapping>
</operation>
...

<module ref="rampart"/>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsu="http://.../oasis-200401-wss-wssecurity-utility-1.0.xsd"
    wsu:Id="UsernameToken">
    <wsp:ExactlyOne>
    <wsp:All>
    <sp:SupportingTokens
        xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
        <sp:UsernameToken
            sp:IncludeToken="http://.../IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
            <sp:HashPassword/>
            </wsp:Policy>
            </sp:UsernameToken>
        </wsp:Policy>
        </sp:SupportingTokens>

        <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
            <ramp:passwordCallbackClass>...PWCBHandler</ramp:passwordCallbackClass>
        </ramp:RampartConfig>

    </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
</service>
</serviceGroup>

```

## services.xml を容易に再生成する方法

Axis2 の Wsd12Java ツールを使用してサーバー・サイドのデプロイメント・ファイルを生成すると、生成される成果物の 1 つとして、(生成対象のディレクトリー配下の resources ディレクトリー内に) services.xml ファイルが作成されます。このファイルは WSDL サービス定義を変更した場合には再生成する必要があり、そのたびにこのファイルにモジュール参照とポリシー情報を組み込まなければならないので手間がかかります。サンプル・コードには、生成されたファイルに対する変更を自動化するツールとして、com.sosnoski.ws.MergeServerPolicy クラスが組み込まれています(ソースおよびバイナリーは mergetool ディレクトリー内にあります)。build.xml ファイルの generate-server ターゲットは、サーバー・コードが生成されるたびにこのツールを実行することで、生成済みの services.xml ファイルにモジュール参照と適切なポリシー情報を挿入します。お望みであれば、独自のプロジェクトにこのツールを使用してください。このツールはコマンドラインの最初のパラメーターとして services.xml ファイルへのパス、2 番目のパラメーターとしてポリシー・ファイルへのパス、そして残りのパラメーターとして、追加される任意の数のモジュールの名前を取ります。

リスト 4 に組み込まれたポリシーをリスト 3 の基本ポリシーと比較すると、リスト 4 には `<ramp:RampartConfig>` 要素が追加されていることがわかります。この要素はポリシー情報に Rampart 特有の拡張を加える要素で、この例ではパスワード・コールバックを処理するために使用するクラスの名前を指定しています。サーバー・コードはこのコールバックという手段で、ク

クライアントがリクエストに指定したユーザー名とパスワードの組み合わせを検証することができます。

[リスト 5](#) に、平文のパスワードに使用する場合のコールバック・クラスの実際の実装を記載します。この例では、ユーザー名とパスワードの両方がコールバックに提供されます。コールバックに必要なのは、その組み合わせを検証することだけです。ユーザー名とパスワードが期待される値と一致した場合はユーザー名とパスワードをそのまま返し、そうでない場合はエラーを示す例外をスローします。

## リスト 5. パスワード・コールバックのコード

```
import org.apache.ws.security.WSPasswordCallback;

public class PWCBHandler implements CallbackHandler
{
    public void handle(Callback[] callbacks)
        throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];
            String id = pwcb.getIdentifer();
            if (pwcb.getUsage() == WSPasswordCallback.USERNAME_TOKEN_UNKNOWN) {

                // used when plain-text password in message
                if (!"libuser".equals(id) || !"books".equals(pwcb.getPassword())) {
                    throw new UnsupportedCallbackException(callbacks[i], "check failed");
                }
            }
        }
    }
}
```

実際のアプリケーションでは当然、これとは別のメカニズム (データベースや外部セキュリティ・メカニズムなど) でユーザー名とパスワードの組み合わせを検証する必要がでてくると思います。コールバックの手法では、Rampart セキュリティ処理の拡張として必要な任意の検証手法を使用することができます。

## クライアント・サイドの構成

クライアント・コードに Rampart を使用するには、まずはこのモジュールを Axis2 で使用できるようにしなければなりません。それには、クライアントの Axis2 リポジトリ構造を構成するという方法もとれますが、通常は rampart.mar モジュール・ファイル (および、使用が必要があるその他すべてのモジュール) をクラスパスに組み込むだけのほうが簡単です。付属のサンプルでは、このクラスパスの方法を使用しています。

次に、クライアントのセキュリティ・ポリシーと関連パラメーターを構成します。この構成を処理する最も簡単な方法は、直接サービス・スタブで値を設定することです。[リスト 6](#) に、サンプル・コードでの構成方法を示します。

## リスト 6. クライアントの構成

```
/**
 * Load policy file from classpath.
 */
private static Policy loadPolicy(String name) throws XMLStreamException {
    ClassLoader loader = WebServiceClient.class.getClassLoader();
```



```

        InputStream resource = loader.getResourceAsStream(name);
        StAXOMBuilder builder = new StAXOMBuilder(resource);
        return PolicyEngine.getPolicy(builder.getDocumentElement());
    }

    public static void main(String[] args) throws IOException, XMLStreamException {

        // check for required command line parameters
        if (args.length < 4) {
            System.out.println("Usage:\n java " +
                "com.sosnoski.ws.library.adb.WebServiceClient protocol host port path");
            System.exit(1);
        }

        // create the client stub
        String target = args[0] + "://" + args[1] + ":" + args[2] + args[3];
        System.out.println("Connecting to " + target);
        LibraryUsernameStub stub = new LibraryUsernameStub(target);

        // configure and engage Rampart
        ServiceClient client = stub._getServiceClient();
        Options options = client.getOptions();
        options.setProperty(RampartMessageData.KEY_RAMPART_POLICY,
            loadPolicy("policy.xml"));
        options.setUserName("libuser");
        options.setPassword("books");
        client.engageModule("rampart");
    }

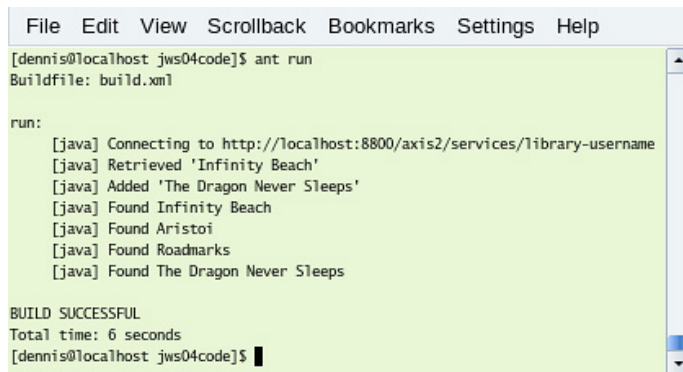
```

構成が行われている部分は、リスト 6 の最後のコード・ブロックです。この部分では、まず作成されたスタブから `org.apache.axis2.client.ServiceClient` インスタンスを取得し、クライアント・オプションに (クラスパスからロードした) ポリシー情報と、ユーザー名/パスワードを設定します。続いて Axis2 構成の中でクライアントが使用する Rampart モジュールを関連付けます。この作業が完了すれば、WS-Security を使用していない場合とまったく同じように、スタブを使用してサービスにアクセスできるようになり、Rampart は各リクエストに `UsernameToken` を自動的に追加するようになります。

## 結果の確認

Ant がインストールされていれば、サンプル・コードのディレクトリーを指定して開いたコンソールから `ant` を実行するだけで、クライアント・コードとサーバー・コードの両方をビルドすることができます。これによって作成された `library-username.aar` ファイルを Axis2 サーバー・システム (もちろん、Rampart の `.jar` および `.mar` ファイルが組み込まれたシステム) にデプロイすると、コンソールに `ant run` と入力することで、クライアントを試してみることができます。すべて正しくセットアップされていれば、[図 1](#) に示す出力が表示されるはずです。

## 図 1. アプリケーション実行時のコンソール出力



```
File Edit View Scrollback Bookmarks Settings Help
[dennis@localhost jws04code]$ ant run
Buildfile: build.xml

run:
[java] Connecting to http://localhost:8800/axis2/services/library-username
[java] Retrieved 'Infinity Beach'
[java] Added 'The Dragon Never Sleeps'
[java] Found Infinity Beach
[java] Found Aristoi
[java] Found Roadmarks
[java] Found The Dragon Never Sleeps

BUILD SUCCESSFUL
Total time: 6 seconds
[dennis@localhost jws04code]$
```

サーバーとクライアントを一緒に実行しただけでは、もちろん何が起きているのかはわかりません。WS-Security の UsernameToken の動作を確認するには、クライアントとサーバーの仲介役として機能する TCPMon などのツールを使用して交換されるメッセージをキャプチャーするという方法があります (「[参考文献](#)」を参照)。この方法を行うには、まず TCPMon をセットアップし、あるポートでクライアントからの接続を受け入れられるようにし、その接続を今度は別のポート (または別のホスト) で実行中のサーバーに転送します。その後、build.properties ファイルを編集し、host-port の値を TCPMon のリスニング・ポートに変更します。この状態で、コンソールに ant run と入力すると、交換されているメッセージが表示されるはずです。[リスト 7](#) に、クライアント・メッセージをキャプチャーした例を記載します。

## リスト 7. UsernameToken を使用したクライアント・メッセージ

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security xmlns:wsse="...wss-wssecurity-secext-1.0.xsd"
      soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="...wss-wssecurity-utility-1.0.xsd"
        wsu:Id="UsernameToken-1815911473">
        <wsse:Username>libuser</wsse:Username>
        <wsse:Password Type="...wss-username-token-profile-1.0#PasswordText"
          >books</wsse:Password>
        </wsse:UsernameToken>
      </wsse:Security>
    </soapenv:Header>
    <soapenv:Body>
      <ns2:getBooksByType xmlns:ns2="http://ws.sosnoski.com/library/wsd1">
        <ns2:type>scifi</ns2:type>
      </ns2:getBooksByType>
    </soapenv:Body>
  </soapenv:Envelope>
```

## UsernameToken をセキュアにする

基本的な平文の UsernameToken を使用するだけでは十分なセキュリティにはなりません。メッセージをモニターできる誰に対しても、ユーザー名とそれに対応するパスワードの両方が可視になってしまうからです。暗号化した通信チャネルを使用しているとしたら問題にはありませんが、それは、チャネルの暗号化が確実で部外者がメッセージをモニターできない場合に限りです。好都合なことに、WS-SecurityPolicy では[リスト 8](#)に記載するように暗号化チャネルの使用を要件とする方法を定義しています (このリストもページ幅に合わせるために行を分割しています)。

実際のポリシーは、サンプル・コード・パッケージの `secure-policy-server.xml` ファイルで確認してください。

## リスト 8. HTTPS 接続を要件とするポリシー

```
<wsp:Policy wsu:Id="UsernameToken" xmlns:wsu=
  "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:TransportBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <wsp:Policy>
          <sp:TransportToken>
            <wsp:Policy>
              <sp:HttpsToken RequireClientCertificate="false"/>
            </wsp:Policy>
          </sp:TransportToken>
        </wsp:Policy>
      </sp:TransportBinding>
      <sp:SupportingTokens
        xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
            ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"/>
        </wsp:Policy>
      </sp:SupportingTokens>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

リスト 8 の太字で示している部分が追加された内容で、このように `<sp:TransportBinding>` 要素に `<sp:HttpsToken>` 要素がネストされています。この `<sp:HttpsToken>` 要素が指定している内容は、サービスとの通信にはセキュア HTTPS 接続を使用するという要件です。このポリシーを使って .aar サービスをビルドし (build.properties の `server-policy` 値を `secure-policy-server.xml` に変更してから `ant build-server` を実行し)、デプロイしようすると、Rampart がこのポリシー要件を適用し、通常の HTTP 接続を拒否することがわかります。

サービスに対して HTTPS 接続を試してみることもできますが、それにはまず、HTTPS 接続をサポートするように Web サーバーを構成しなければなりません (その方法については、Tomcat が `/tomcat-docs/ssl-howto.html` でわかりやすく説明しています)。さらに、build.properties の `protocol` 値を `https` に変更し、Web サーバーの自己署名証明書を使用している場合には Ant の `test` ターゲットを実行するときにトラスト・ストアをクライアントに渡す必要もあります。付属の build.xml にはそのためのコメント・アウトされた行があるので、その行のコメントを外し、システム上の適切なトラスト・ストア・ファイルの場所を設定するだけで済みます。

UsernameToken をよりセキュアにするためのもう 1 つの方法は、暗号化されていないリンクにも効果が及びます。この方法では、パスワードに他の 2 つのテキスト値を組み合わせたストリングで計算されたダイジェスト値を使用します。テキスト値の一方はノンス、つまりリクエストごとに送信者が生成する乱数値です。もう一方のテキスト値は作成時のタイムスタンプで、これは送信者が UsernameToken を作成したときの時刻にすぎません。この 2 つの値が UsernameToken に平文で組み込まれます。クライアントとサーバーの両方が正しく使用すれば、ダイジェストに含まれるこの 2 つの値とパスワードの組み合わせによって、サーバーはダイジェストの生成時に正しいパスワードが使用されたことを確認できると同時に、部外者が有効なパスワードを偽造することが難しくなります。リスト 9 に、ダイジェスト・パスワードを使用する場合のポリシー例と、

それに続き、ダイジェスト・パスワードを使用したメッセージからの実際にキャプチャーしたものを記載します (両方ともページ幅に合わせるためにフォーマットを変えてあります。実際のポリシーは、hash-policy-client.xml ファイルを確認してください)。このリストでも、元のポリシーとの違いは太字で示しています。

## リスト 9. パスワード・ダイジェストを使用したポリシーとサンプル・メッセージ

```
<wsp:Policy wsu:Id="UsernameToken" xmlns:wsu=
"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:SupportingTokens
        xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
              <sp:HashPassword/>
            </wsp:Policy>
          </sp:UsernameToken>
        </wsp:Policy>
      </sp:SupportingTokens>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>

<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <wsse:Security xmlns:wsse=".../oasis-200401-wss-wssecurity-secext-1.0.xsd"
      soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="...wss-wssecurity-utility-1.0.xsd"
        wsu:Id="UsernameToken-1421876889">
        <wsse:Username>libuser</wsse:Username>
        <wsse:Password Type="...wss-username-token-profile-1.0#PasswordDigest"
          >/Wt/2yDdZwa8a5qd7U70hrp29/w=</wsse:Password>
        <wsse:Nonce>4ZQz5ytME/RXfChuKJ03iA==</wsse:Nonce>
        <wsu:Created>2009-03-17T11:20:57.467Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <ns2:getBooksByType xmlns:ns2="http://ws.sosnoski.com/library/wsd1">
      <ns2:type>scifi</ns2:type>
    </ns2:getBooksByType>
  </soapenv:Body>
</soapenv:Envelope>
```

## まとめ

この記事では、ポリシーをベースとした基本的な形の WS-Security 処理に Axis2 と Rampart を使用する方法を説明しました。次回の「[Java Web サービス](#)」では、WS-Security に備わった 2 つの強力な機能、XML 暗号化および署名について学びます。XML 暗号化を使用すると、どのようなタイプの接続で操作するとしても、さらには信用できない仲介が処理に関連するとしても、メッセージの内容に関する機密を保つことができます。一方、XML 署名によって、メッセージが本当にリクエストの送信元からのものであること、そしてメッセージの内容が送信中に改ざんされていないことが保証されます。暗号化と署名は、ほとんどのエンタープライズ・セキュリティ実装にとって基本的な機能なので、これらの機能を独自の Web サービスに適用する方法をもう一度確認してください。





## ダウンロード

内容	ファイル名	サイズ
Source code for this article	<a href="#">j-jws4.zip</a>	10KB

## 著者について

Dennis Sosnoski



Dennis Sosnoski は Java ベースの [XML および Web サービス](#) を専門とするコンサルタント兼トレーナーです。専門家としてのソフトウェア開発経験は 30 年以上に渡り、この 10 年間はサーバー・サイドの XML 技術や Java 技術に注力しています。オープンソースの [JiBX XML Data Binding](#) フレームワークや、それに関連した [JiBX/WS](#) Web サービス・フレームワークの開発リーダーを務め、さらに [Apache Axis2](#) Web サービス・フレームワークのコミッターでもあります。彼は JAX-WS 2.0 および JAXB 2.0 仕様のエキスパート・グループの一員でもありました。

© Copyright IBM Corporation 2009

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

商標

([www.ibm.com/developerworks/jp/ibm/trademarks/](http://www.ibm.com/developerworks/jp/ibm/trademarks/))