「Javaアプリケーション脆弱性事例調査資料」について

- この資料は、Javaプログラマである皆様に、脆弱性を身近な問題として感じてもらい、セキュアコーディングの重要性を認識していただくことを目指して作成しています。

- 「Javaセキュアコーディングスタンダード CERT/Oracle版」と合わせて、セキュアコーディングに関する理解を深めるためにご利用ください。

JPCERTコーディネーションセンター
セキュアコーディングプロジェクト
secure-coding@jpcert.or.jp

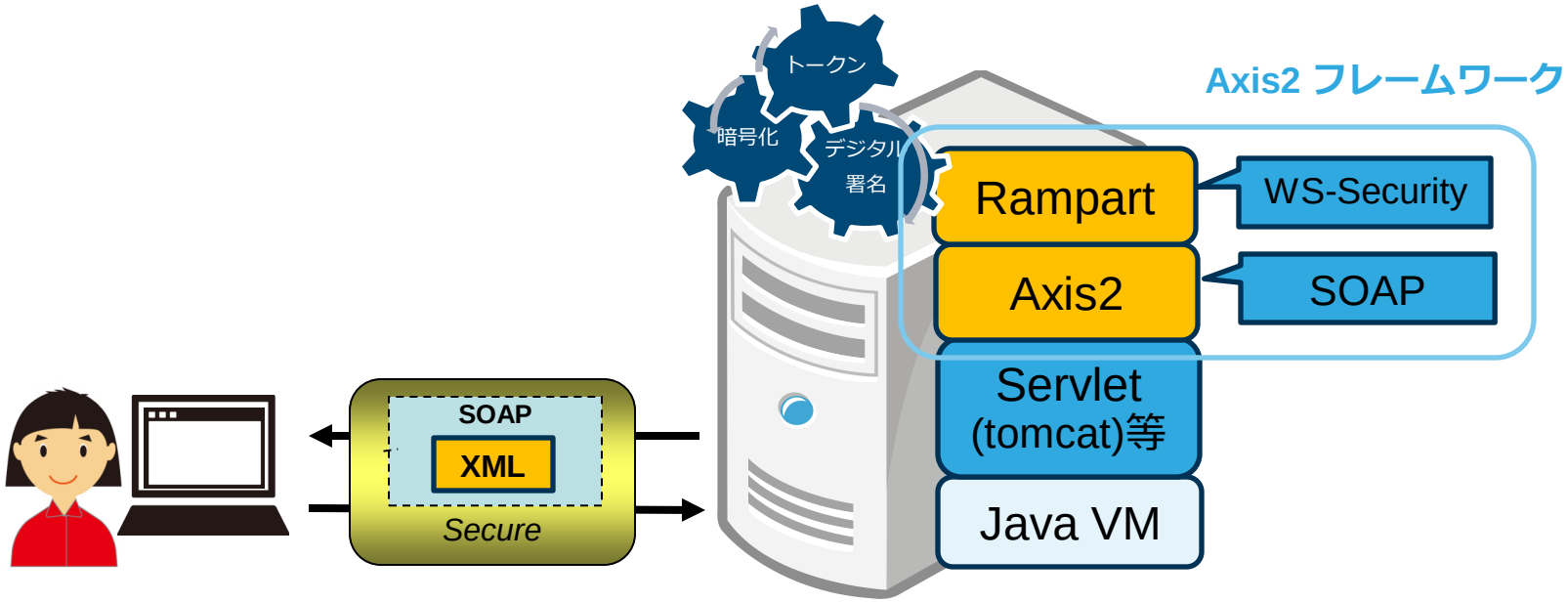
Apache Axis2 における XML署名検証不備

CVE-2012-4418

JVNDB-2012-004882

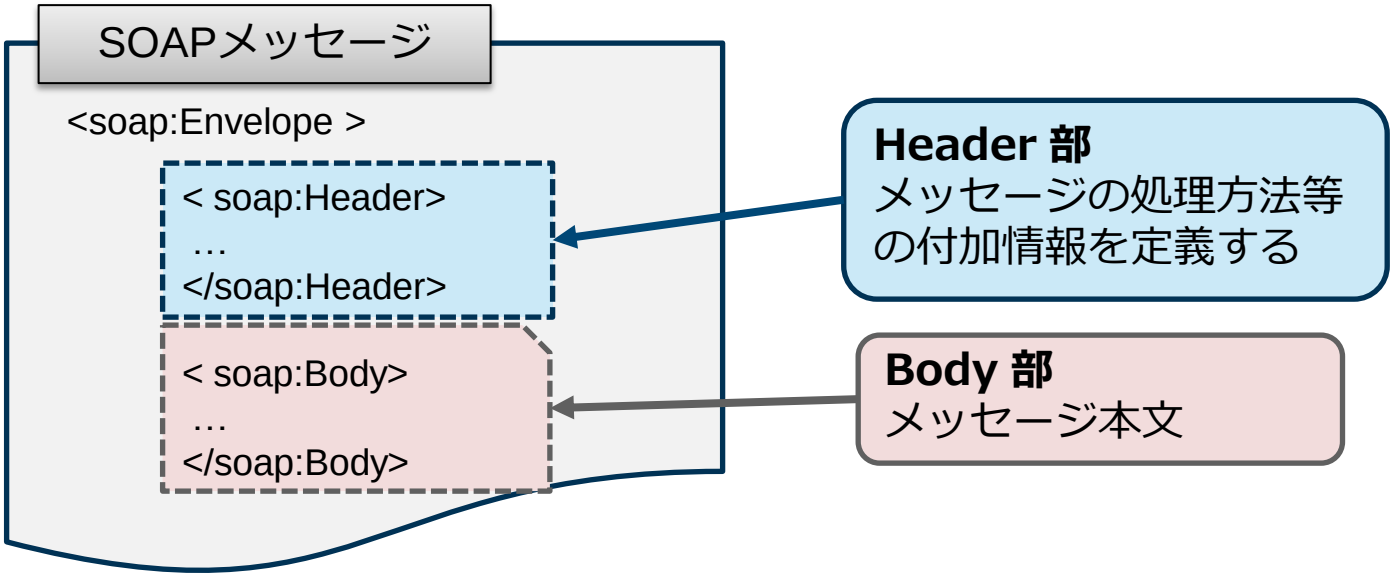
Apache Axis2とは

- XMLをベースとしたメッセージ交換プロトコル SOAP を実装した Webアプリケーションフレームワーク
- SOAP にセキュリティ機能を追加する WS-Security は、Axis2ではRampartモジュールで実装されている



SOAP(XML)とは

- **SOAP(Simple Object Access Protocol)**
- ネットワーク経由で情報を交換するためのプロトコル
- XML形式によるシンプルな構造
- Header部とBody部から構成される
- WS-Security を用いて安全な交換が可能



WS-Security とは

WS-Security(Web Services Security)

- SOAPメッセージ交換に対してセキュリティを提供するための仕様
- OASIS[*1]のWeb Service Security TC によって仕様策定された
- WS-Securityのセキュリティ機能
  1. セキュリティ トークン(認証および認可に利用)
  2. デジタル署名
  3. 暗号化

*1 OASIS … オープン規格標準を策定・推進する非営利団体。
https://www.oasis-open.org/

デジタル署名とは

■デジタル署名

- ネットワーク上での印鑑やサインに相当するもの
  以下のような効果がある

—なり済ましを検出

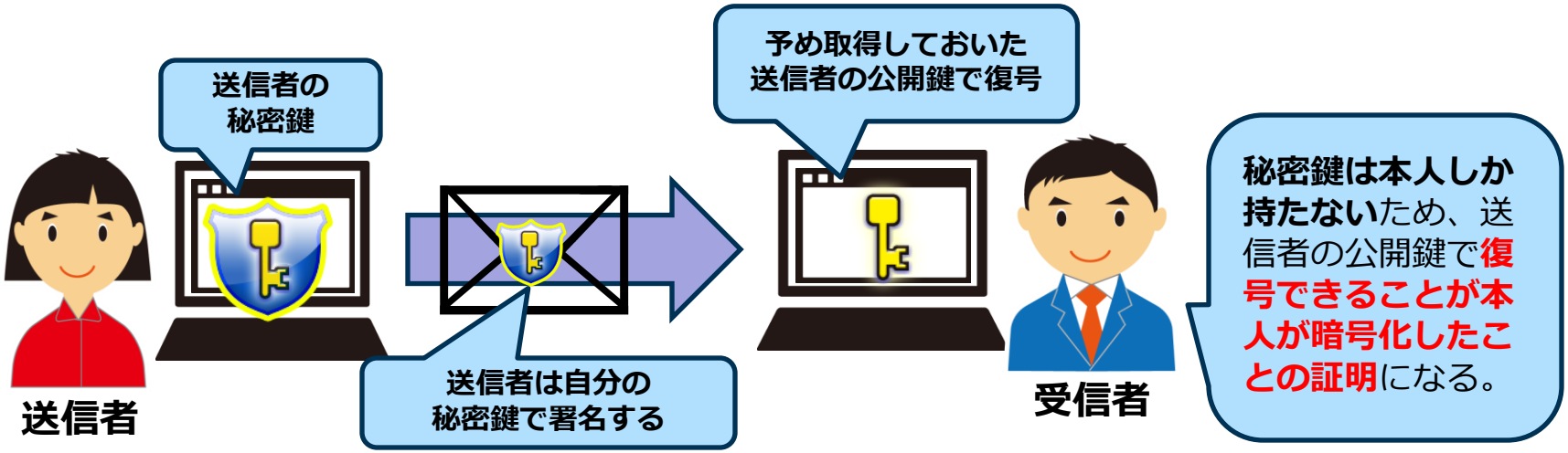
—改ざんを検出

—否認防止　※送信事実の否定を防止すること

デジタル署名の仕組み

■公開鍵暗号系の性質を利用

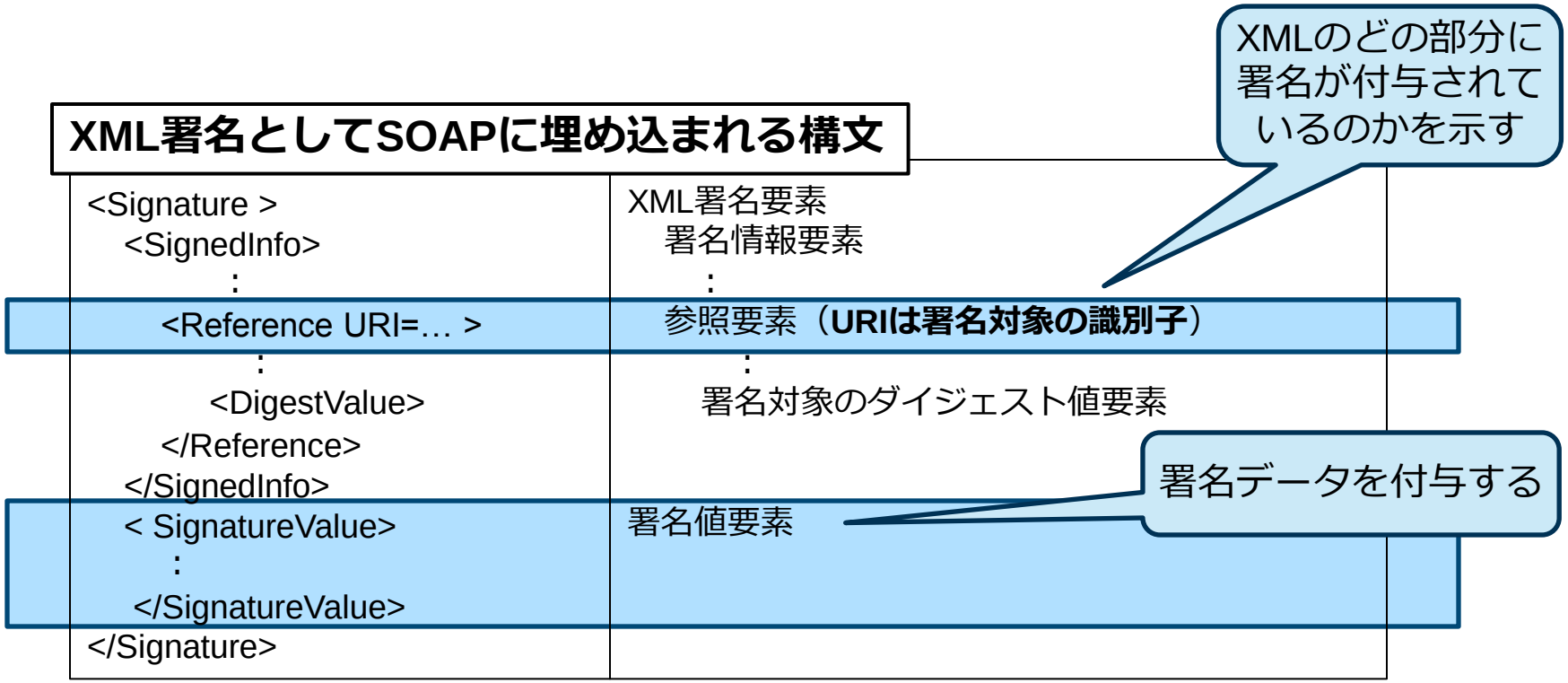
—秘密鍵とそれに対応する公開鍵が存在
—秘密鍵で暗号化したものは対応する公開鍵でしか復号できない
—公開鍵で暗号化したものは対応する秘密鍵でしか復号できない
—秘密鍵は本人だけが持つ、公開鍵は相手に持ってもらう



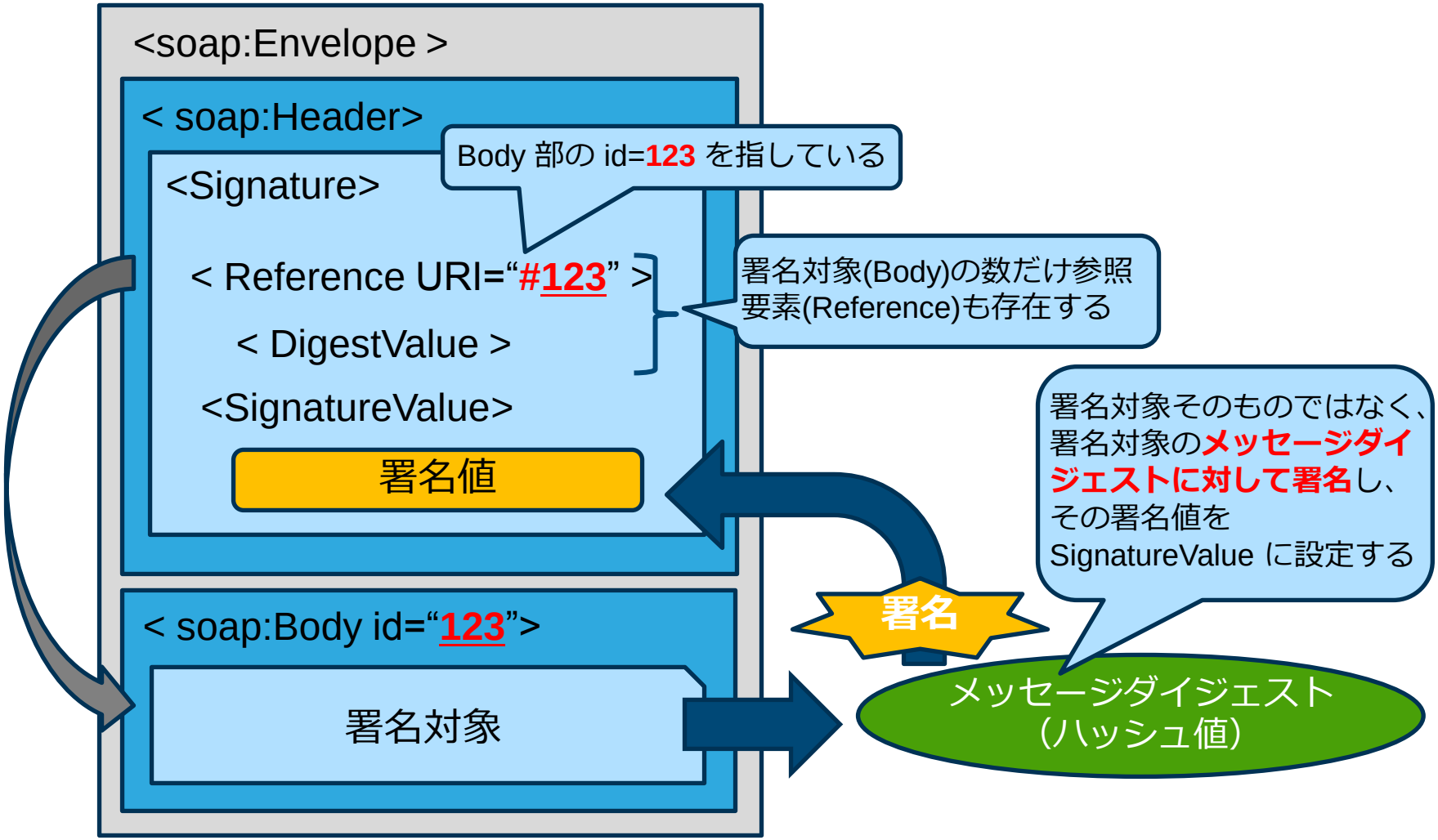
送信者本人が送信したものである(＝改ざんされていない)ことを検証する場合

XML署名とは: XML署名の構文

- XML署名とは、XMLに対しデジタル署名を付与すること
- XMLの文書全体だけでなく、XMLの部分的な要素に対して署名が可能



XML署名とは: Reference値による署名対象の指定



メッセージダイジェストとは

- メッセージの指紋のようなもの
- 固定長のバイト数で構成される
- メッセージが１ビットでも異なればメッセージダイジェスト値も異なる
- メッセージダイジェスト値が同一となる異なるメッセージの作成は困難
- 同一メッセージからは常に同じ値が生成される

Point!

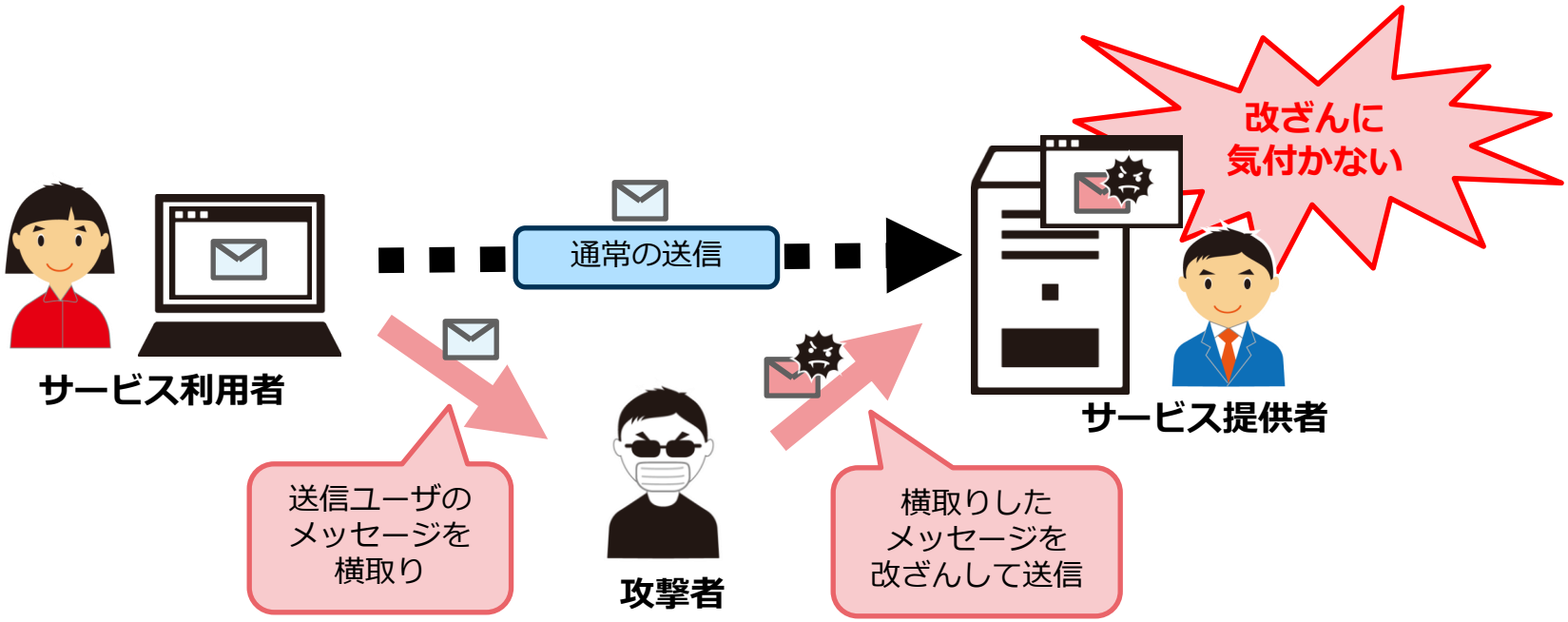
SOAPメッセージ全体の暗号化は時間的なコストが高いことから、
全体ではなくメッセージのダイジェスト値に対して処理をおこなう。

脆弱性の概要

- デジタル署名されたXMLメッセージの署名検証処理に不備
- XML署名を偽装したメッセージに対し、署名が不正であることを検出できず、正しく署名されたものとして扱ってしまう
- なりすまし行為などの脅威

脆弱性が悪用された場合のリスク

- 偽装したメッセージによる、サービスの不正利用
- 送信メッセージが認証や認可に利用されている場合は、認証回避につながる



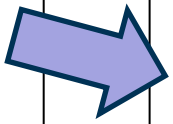
SOAPメッセージの処理フロー

クライアント側

②ポリシーファイル読み込み

③SOAPメッセージを組み立て、ポリシーに従い署名し送信する

署名、送信



サーバ側

①ポリシーファイル読み込み

④メッセージを受信・解析し、署名を検証する

受信、署名検証

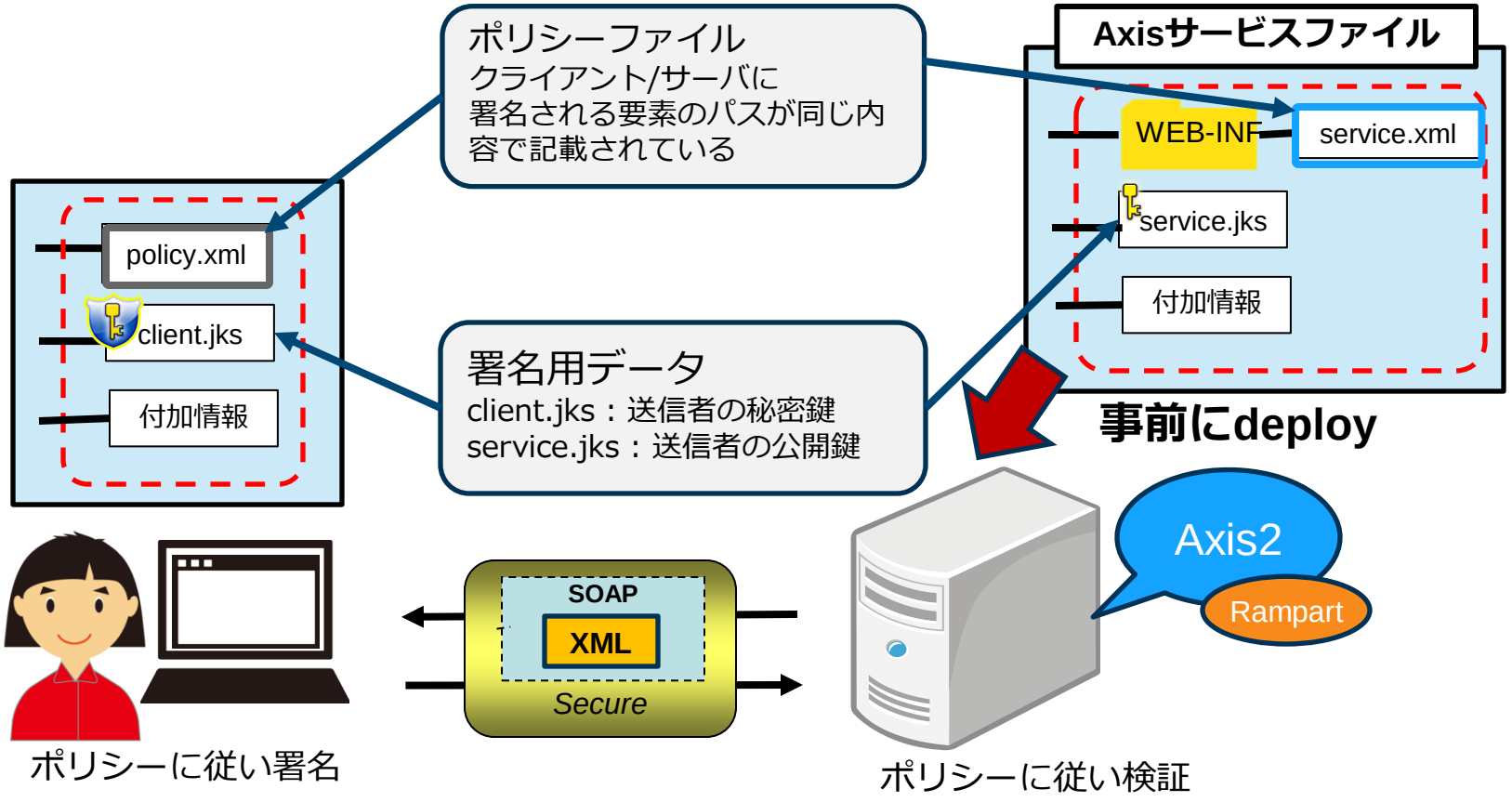
⑤ポリシーに違反していないか検証

ポリシーに従い検証

⑥メッセージからデータを取得する

①②ポリシーファイル読み込み(1/2)

- SOAP メッセージに適用するセキュリティポリシーが定義されている
- クライアント側とサーバ側の両方に署名される要素のパスが同じ内容のポリシーファイルを持つ



①②ポリシーファイル読み込み(2/2)

署名方法や署名個所が記載されているファイル

policy.xml

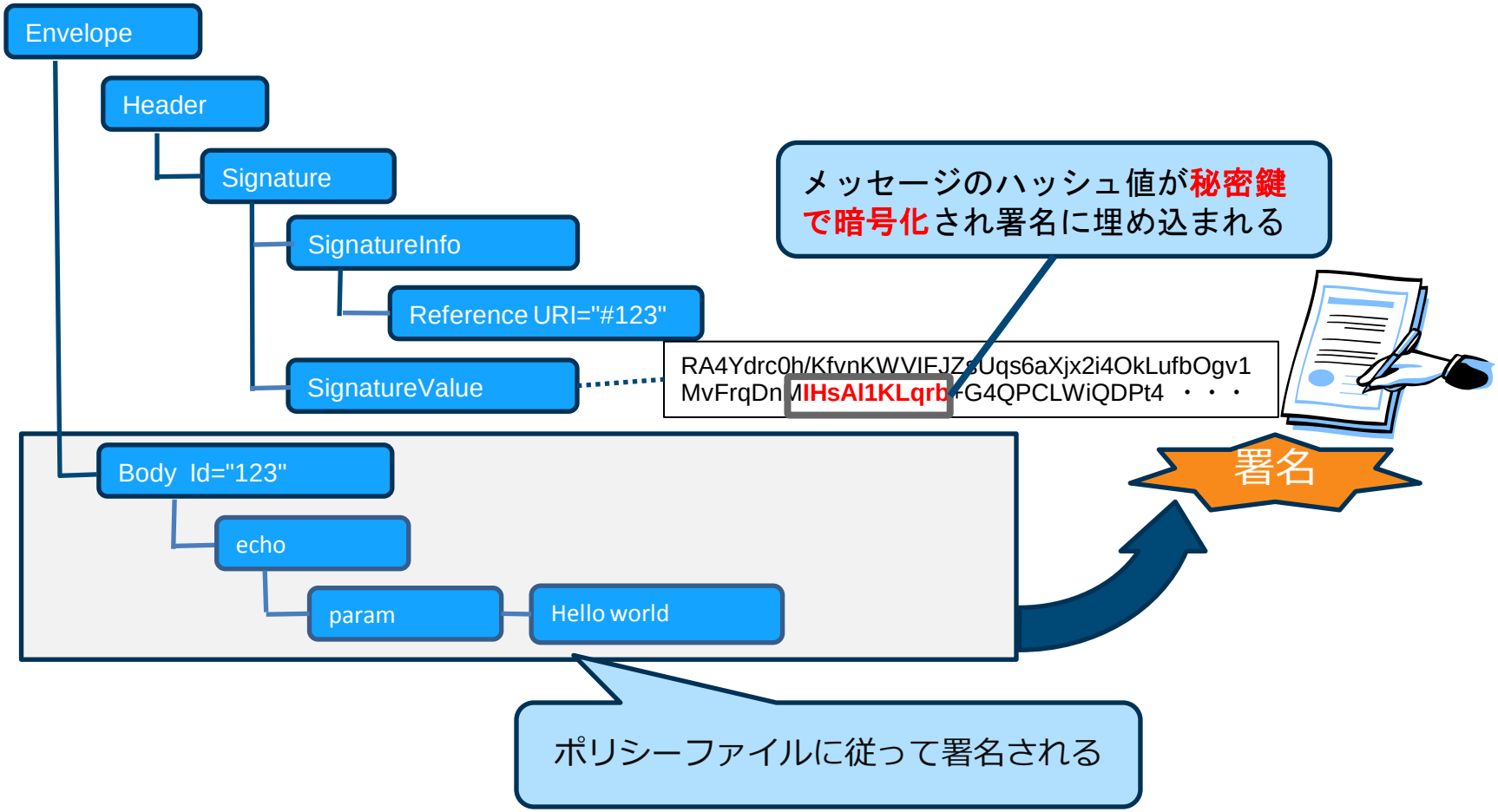
<sp:Body /> = Body 要素配下への署名

SigOnly = 署名のみ

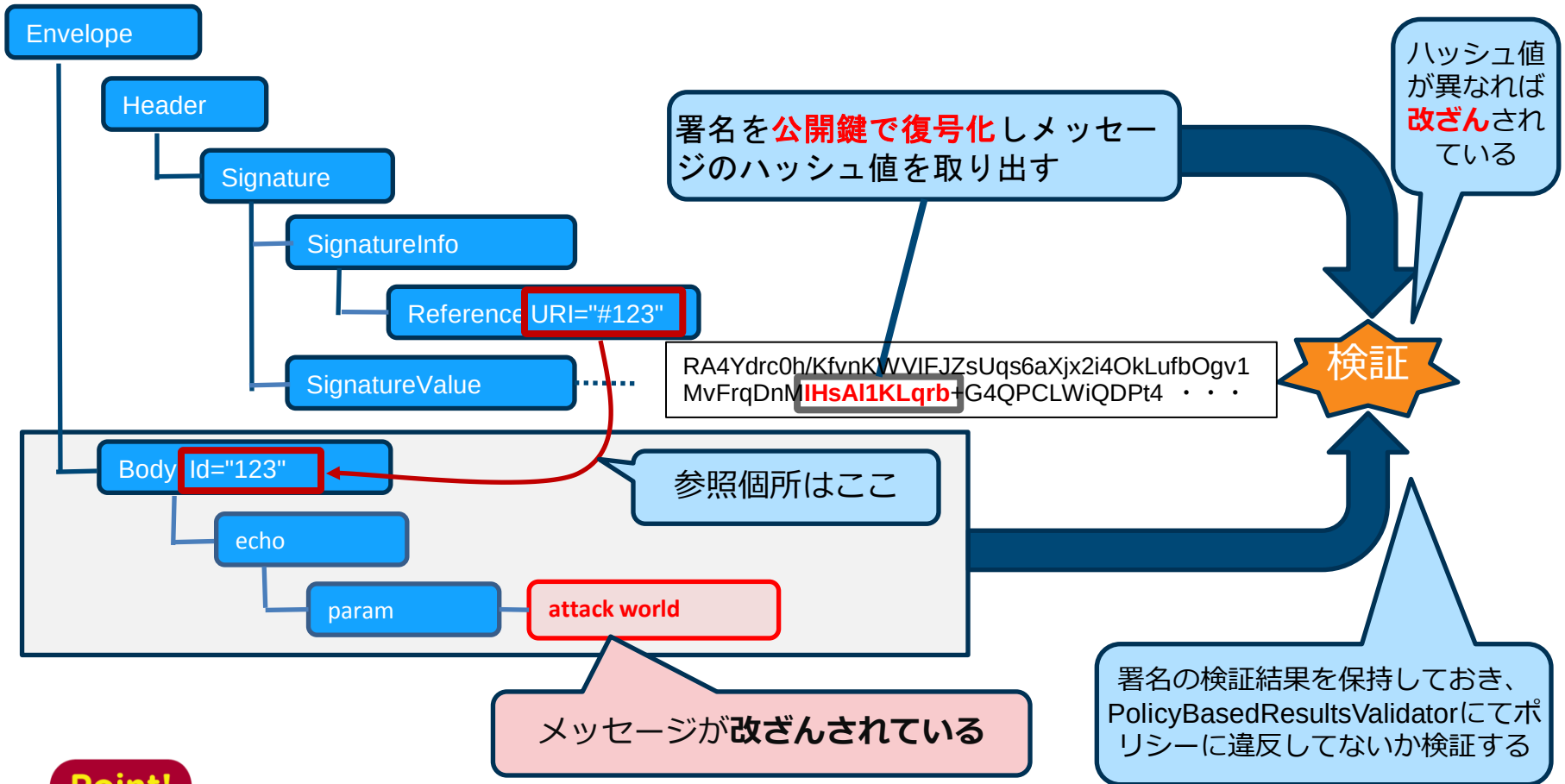
```
<wsp:Policy xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy”  wsu:Id="SigOnly" >
   <wsp:ExactlyOne>
      <wsp:All>
            :
         <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy" >
            <sp:Body />
         </sp:SignedParts>
         <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy" >
            <ramp:user>client</ramp:user>
            <ramp:encryptionUser>service</ramp:encryptionUser>

<ramp:passwordCallbackClass>org.apache.rampart.samples.policy.sample02.PWCBHandler</ramp:passwordCallbackClass>
            <ramp:signatureCrypto>
               <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin" >
                  <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.type" >JKS</ramp:property>
                  <ramp:property name="org.apache.ws.security.crypto.merlin.file" >client.jks</ramp:property>
                  <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.password" >apache</ramp:property>
               </ramp:crypto>
            </ramp:signatureCrypto>
         </ramp:RampartConfig>
      </wsp:All>
   </wsp:ExactlyOne>
```

③SOAPメッセージを組み立て署名し送信する



④メッセージを受信し署名を検証する



Point!

メッセージを改ざんした場合、メッセージ部分は異なるハッシュ値となるため**改ざんを検知することができる。**

⑤ ポリシーに違反していないか検証する

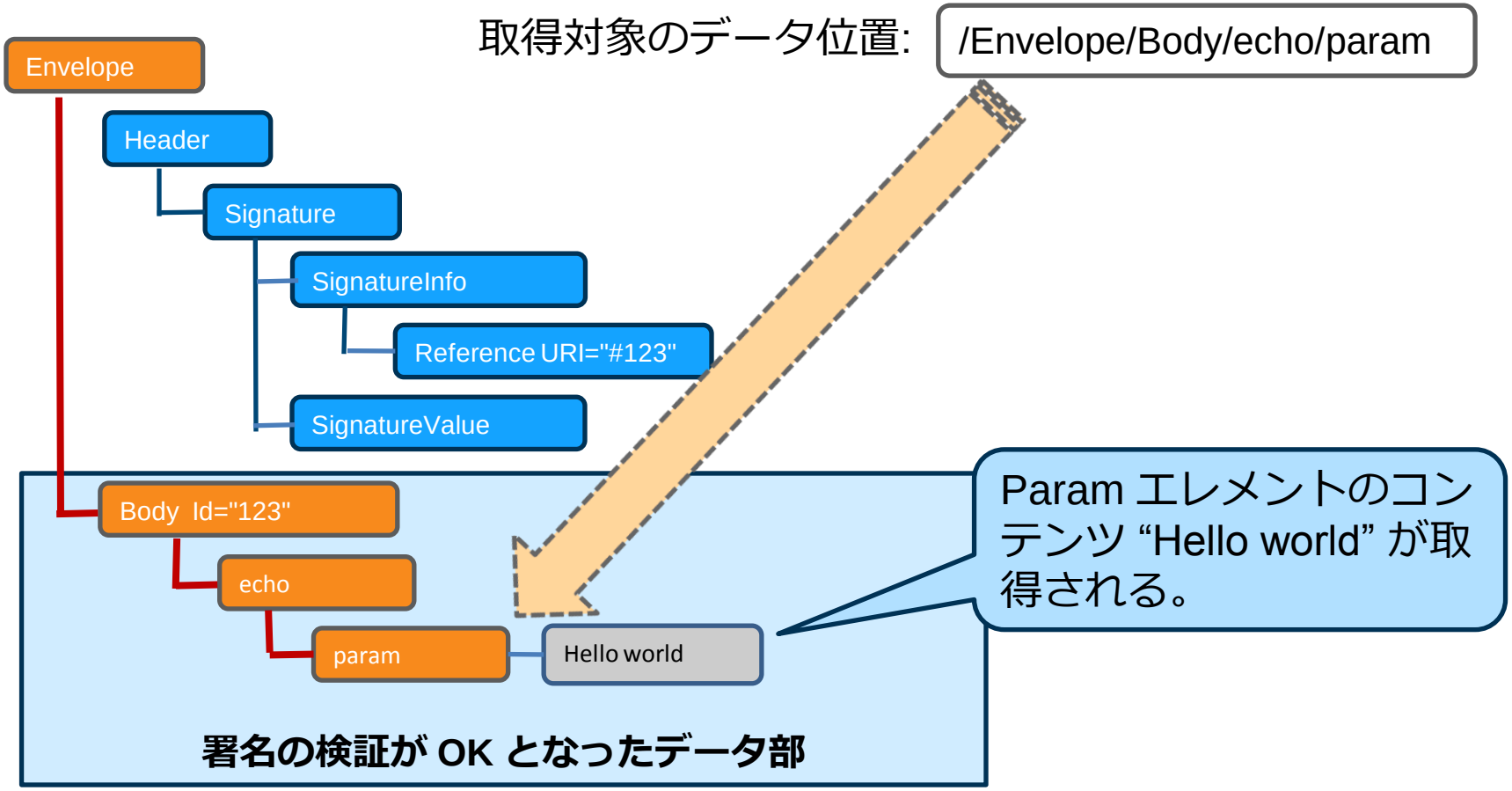
ポリシーファイルのデフォルトの検証処理は、
PolicyBasedResultsValidatorクラスで行われている。
「④メッセージを受信し署名を検証する」の検証結果を元にポリシーに違反していないか検証する。

メッセージに対する署名検証時の処理フロー(PolicyBasedResultsValidator)

1. **セキュリティトークン（ユーザ認証等）でエラーが発生している場合は Exception生成**
2. **暗号化要素でエラーが発生している場合は Exception生成**
3. **署名要素でエラーが発生している場合は Exception生成**
4. **要求されている要素がいずれか存在しない場合は Exception生成**
5. **信頼されていない署名エラーがある場合は Exception生成**
6. **soapヘッダのタイムスタンプが期限切れ等の場合は Exception生成**

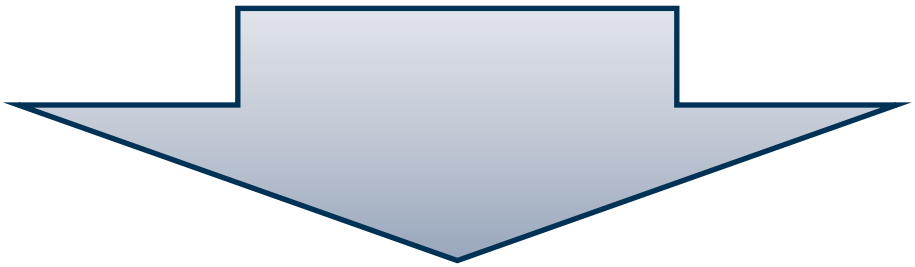
⑥メッセージからデータを取得する

- Axisサーバ上で稼働するアプリケーションがデータを取得



Axis の処理の問題点

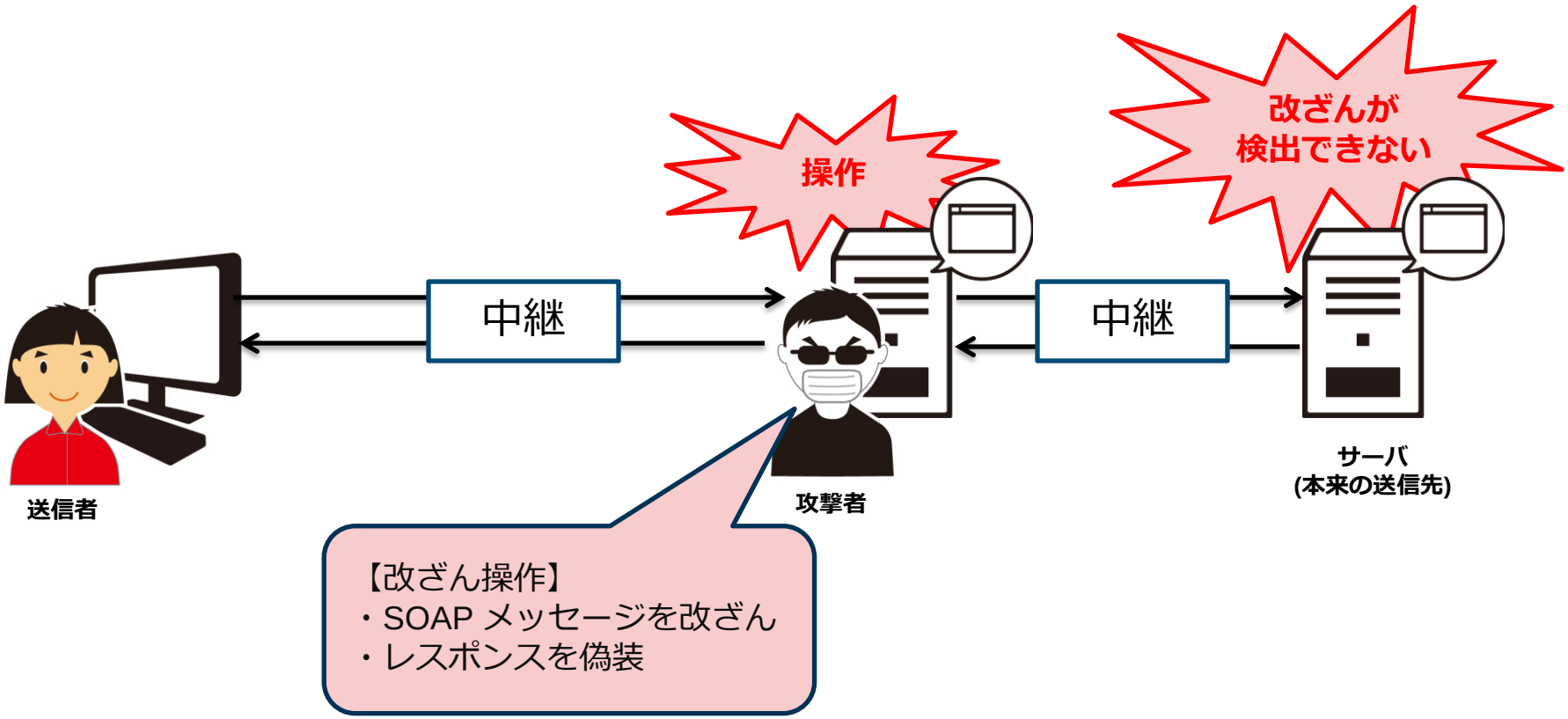
- 署名/署名検証機能は SOAP メッセージを扱う Axis 本体に対する付加機能 (rampart モジュール)
- 署名要素の位置とアプリケーションが処理するデータの格納位置は独立に設定される



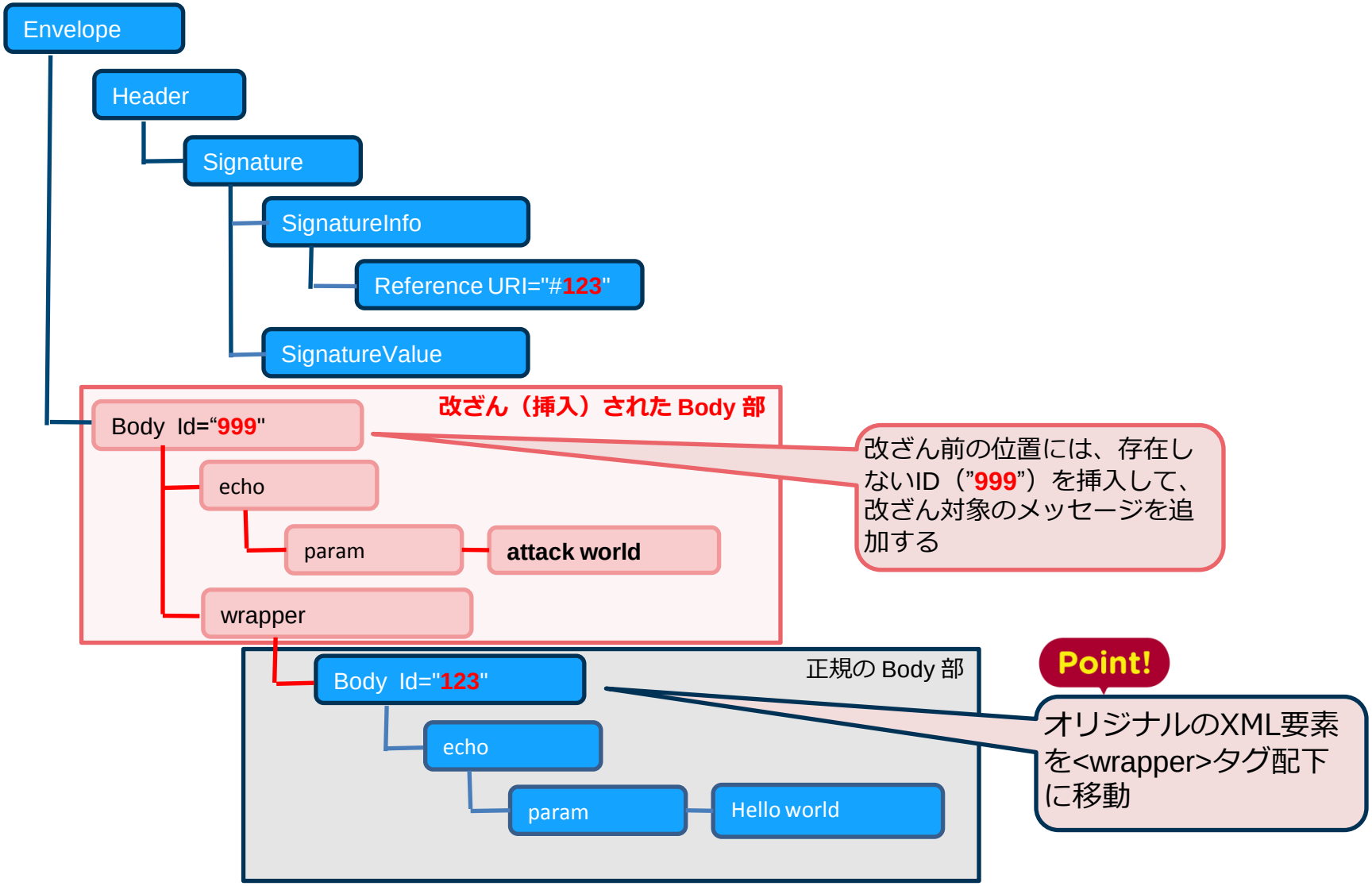
検証すべき署名データが本来想定している位置にあるかどうかを検証していない

攻撃リクエストフロー

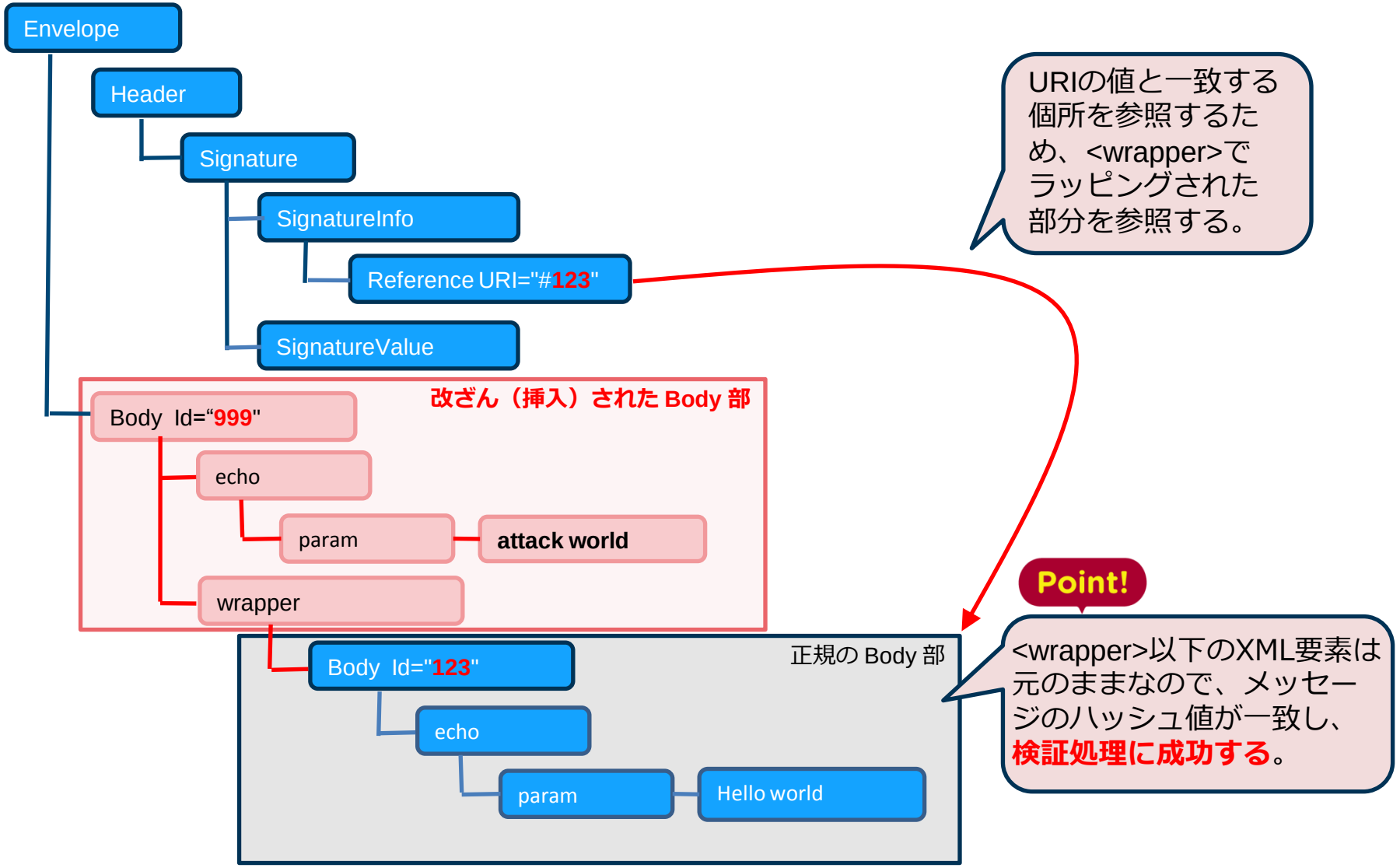
- ユーザとサーバの間に攻撃者が入り、SOAPメッセージを改ざんされた場合に、検出できない問題がある。



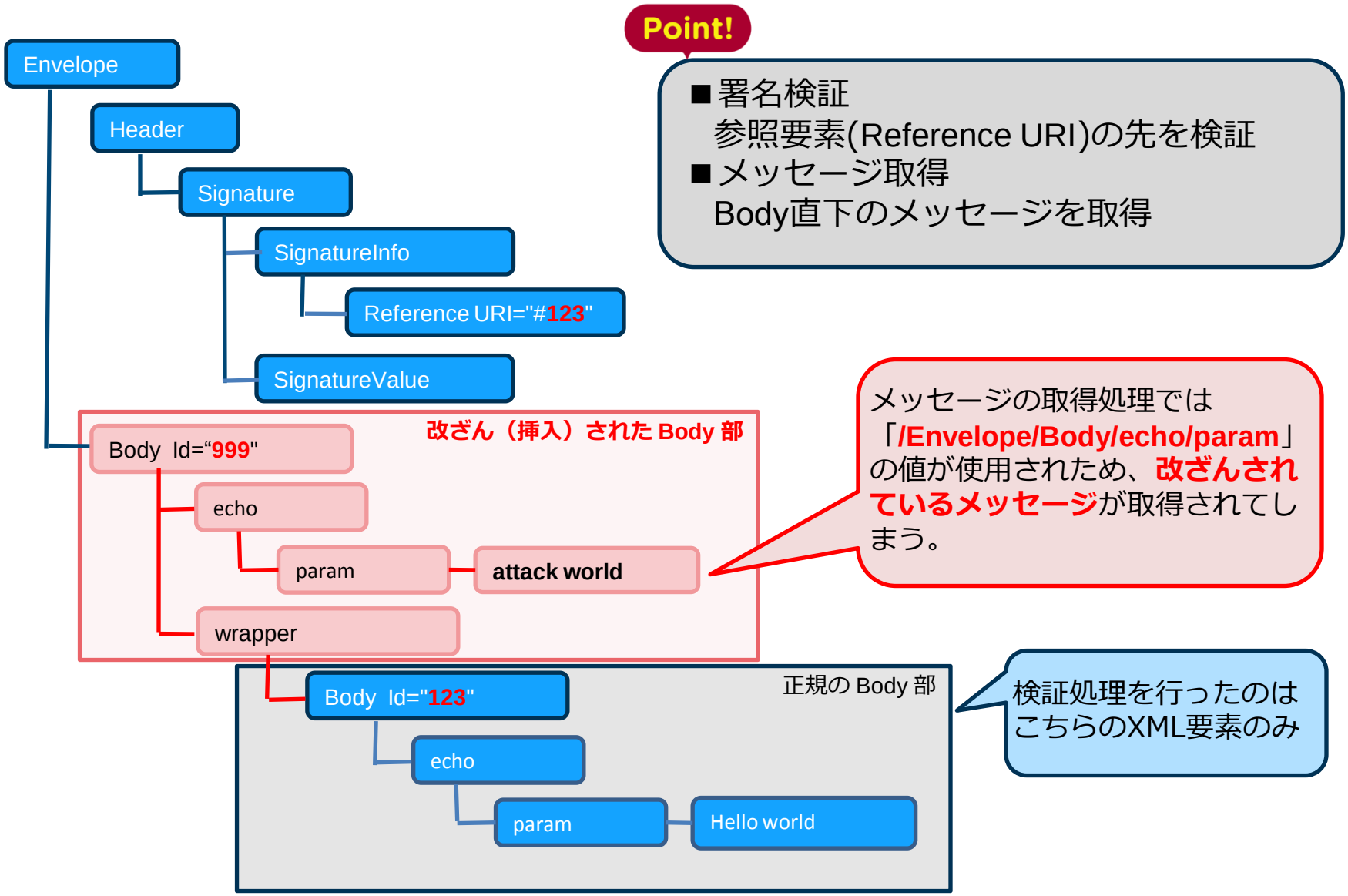
SOAPメッセージの改ざん



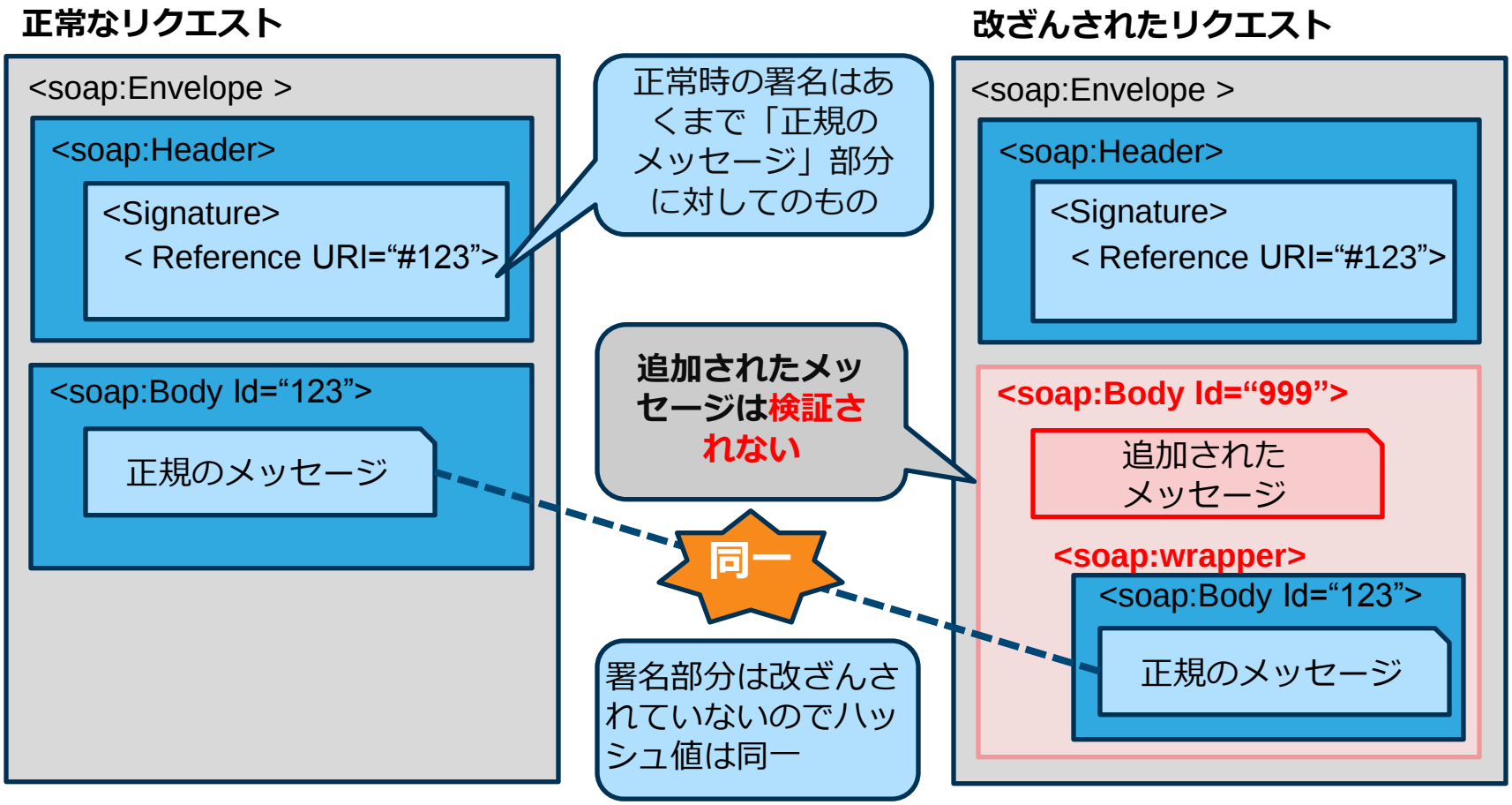
③改ざんされたSOAPメッセージを送信する



⑥メッセージからデータを取得する

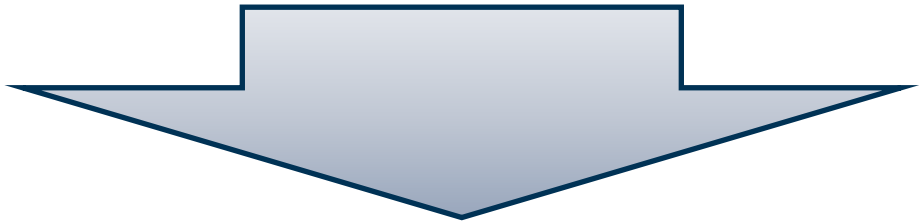


正常なリクエストと改ざんされたリクエストの比較



Axis2のXML署名検証処理の問題点

- 署名時のXML要素位置と検証時のXML要素位置が異なっていても検知しない
- 署名検証時にXML要素位置の検証をしていない



W3CのBest Practices ドキュメントに違反している!!

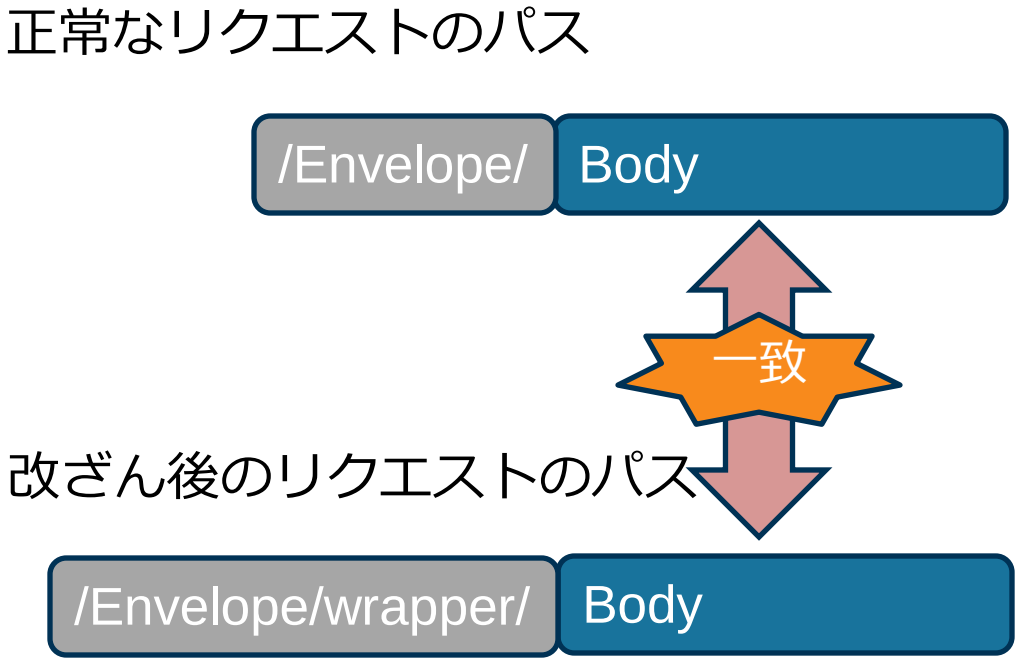
W3C XML Signature Best Practices

http://www.w3.org/TR/xmldsig-bestpractices/

Best Practice 14：検証者は、XML署名検証の過程で名前と位置の両方をチェックする必要があります。

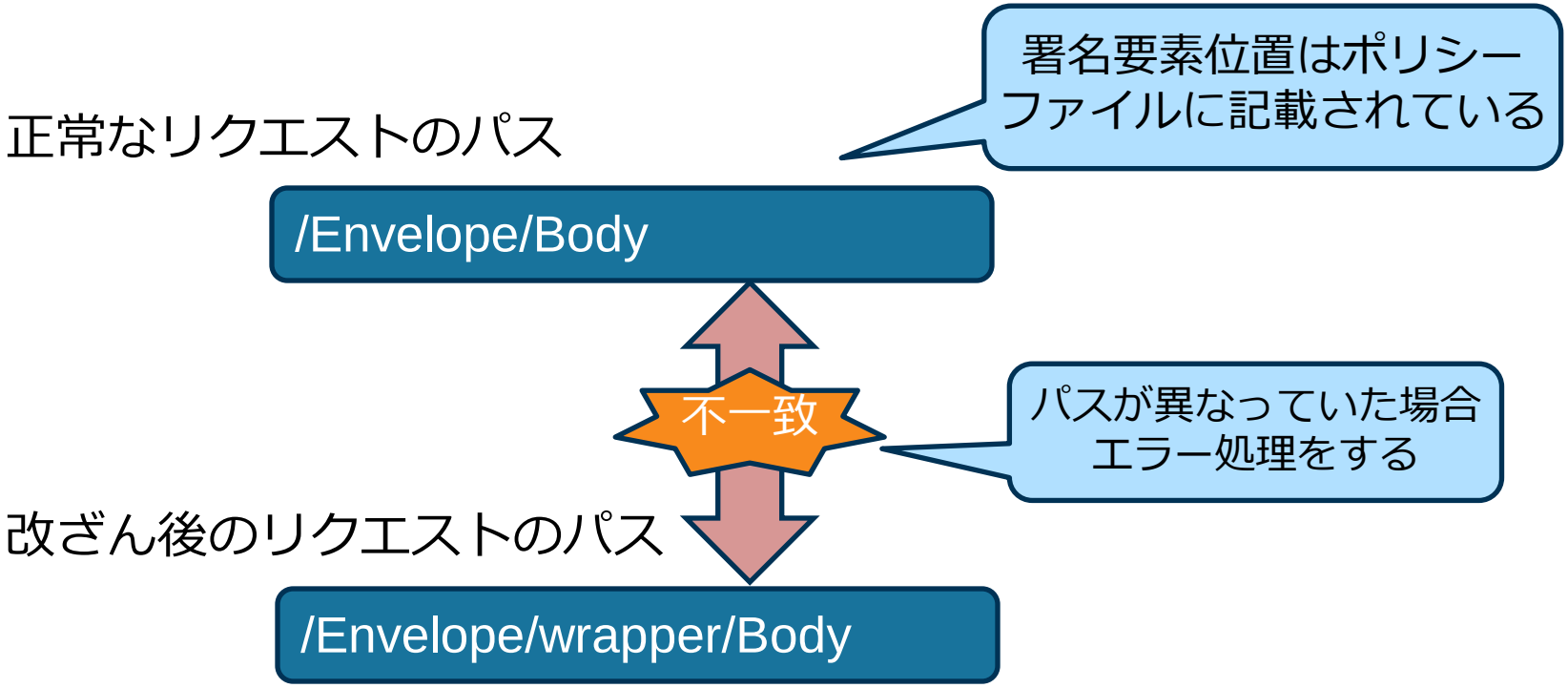
XML署名検証処理の修正: 対策前

- 署名された要素の**位置の比較なし**



XML署名検証処理の修正: 対策後

- 署名された要素の**名前と位置を比較**



名前と要素位置の比較は署名検証が完了した後のタイミングで行う

XML署名検証処理の修正: 対策コード作成

- ポリシーファイルの記載内容に基づく検証処理に要素のパスの検証処理を追加する

- ポリシーファイルのデフォルトの検証処理をおこなっている PolicyBasedResultsValidator クラスを修正する
- XML形式の妥当性検証やXML署名の検証結果についてはすでに完了しているタイミング
- 検証結果を元にポリシー違反をしていないかを検証する工程となっている

XML署名検証処理の修正: 対策コード作成

正常リクエストの「⑥ポリシーファイルに従いポリシーに違反していないかを検証する」の処理を修正する。

「要素の位置チェック」を追加する必要がある

メッセージに対する署名検証時の処理フロー(PolicyBasedResultsValidator)

1. **セキュリティトークン（ユーザ認証等）でエラーが発生している場合は Exception生成**
2. **暗号化要素でエラーが発生している場合は Exception生成**
3. **署名要素でエラーが発生している場合は Exception生成**
   ※ 署名要素に対するチェック方法を修正
4. **要求されている要素がいずれか存在しない場合は Exception生成**
5. **信頼されていない署名エラーがある場合は Exception生成**
6. **soapヘッダのタイムスタンプが期限切れ等の場合は Exception生成**

XML署名検証処理の修正: 対策コード(参考)

PolicyBasedResultsValidator

```
protected void validateSignedPartsHeaders(ValidatorData data,
                        List<WSEncryptionPart> signatureParts, List<WSSecurityEngineResult> results)
            throws RampartException {
  :
 for (final WSSecurityEngineResult actionResult : actionResults) {
            List wsDataRefs = (List) actionResult.get(WSSecurityEngineResult.TAG_DATA_REF_URIS);
  :
            for (final Object objectDataReference : wsDataRefs) {
                  WSDataRef wsDataRef = (WSDataRef) objectDataReference;
  :
                  List<WSEncryptionPart> signedParts = rpd.getSignedParts();
                  boolean find = false;
                  for (final WSEncryptionPart    encParts : signedParts) {
                     if (encParts.getXpath().equals(wsDataRef.getXpath())) {
                        find = true;
                        break;
                     }
                  }
            署名位置が一致しなければ例外生成
}
```

署名された参照URIを取得

ポリシーファイル記載の署名すべき要素

実際のXML署名とポリシーファイル記載のXpath比較

※本脆弱性は現時点で未修正(Apache Axis2/Java 1.6.2)
ここでは独自に作成した修正コードを示す。

参考情報

XML Signature Wrapping Attack

- ここで紹介した攻撃手法は “XML Signature Wrapping Attack” という名称で呼ばれている。
- Apache Axis2 の実装の問題は、USENIX Security 2012 で発表された論文において指摘されている。
- “On Breaking SAML: Be Whoever You Want to Be”, USENIX Security Symposium 2012
- https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final91.pdf

まとめ

- XML署名検証処理における注意点

  XML署名の検証では、署名データの検証だけではなく、XML特有の要素位置に関する検証も必要

- 上記への対策

  署名データの検証と署名データの要素位置の検証を行う。

著作権・引用や二次利用について

■本資料の著作権はJPCERT/CCに帰属します。

■本資料あるいはその一部を引用・転載・再配布する際は、引用元名、資料名および URL の明示をお願いします。

記載例

引用元：一般社団法人JPCERTコーディネーションセンター
Java アプリケーション脆弱性事例解説資料
Apache Axis2 における XML 署名検証不備
https://www.jpcert.or.jp/securecoding/2012/No.10_Apache_Axis.pdf

■本資料を引用・転載・再配布をする際は、引用先文書、時期、内容等の情報を、JPCERT コーディネーションセンター広報(office@jpcert.or.jp)までメールにてお知らせください。なお、この連絡により取得した個人情報は、別途定めるJPCERT コーディネーションセンターの「プライバシーポリシー」に則って取り扱います。

本資料の利用方法等に関するお問い合わせ
JPCERTコーディネーションセンター
広報担当
E-mail：office@jpcert.or.jp

本資料の技術的な内容に関するお問い合わせ
JPCERTコーディネーションセンター
セキュアコーディング担当
E-mail：secure-coding@jpcert.or.jp