XSpRES
Robust and Effective XML Signatures for Web Services

Christian Mainka[1], Meiko Jensen[1], Luigi Lo Iacono[2] and Jörg Schwenk[1]
[1]*Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Bochum, Germany*
[2]*Institute of Media and Imaging Technology, Cologne University of Applied Sciences, Cologne, Germany*

Keywords: XML Signature, XML Signature Wrapping, Web Services, WS-security, SOA, Cloud.

Abstract: XML Encryption and XML Signature are fundamental security standards forming the core for many applications which require to process XML-based data. Due to the increased usage of XML in distributed systems and platforms such as in SOA and Cloud settings, the demand for robust and effective security mechanisms increased as well. Recent research work discovered, however, substantial vulnerabilities in these standards as well as in the vast majority of the available implementations. Amongst them, the so-called XML Signature Wrapping attack belongs to the most relevant ones. With the many possible instances of this attack type, it is feasible to annul security systems relying on XML Signature and to gain access to protected resources as has been successfully demonstrated lately for various Cloud infrastructures and services.
This paper contributes a comprehensive approach to robust and effective XML Signatures for SOAP-based Web Services. An architecture is proposed, which integrates the required enhancements to ensure a fail-safe and robust signature generation and verification. Following this architecture, a hardened XML Signature library has been implemented. The obtained evaluation results show that the developed concept and library provide the targeted robustness against all kinds of known XML Signature Wrapping attacks. Furthermore the empirical results underline, that these security merits are obtained at low efficiency and performance costs as well as remain compliant with the underlying standards.

1 INTRODUCTION

XML is a dominant standard for encoding documents or messages. The range of XML applications is broad. It can be roughly divided into file formats for data at rest such as Docbook, Open Office and WordML for documents or SVG for images and messages for data in transit such as SOAP, XML-RPC or ebXML. The ability of being platform-independent made XML a driving force especially in terms of systems integration. Here resides one important reason why XML is widely used in distributed system and platform contexts such as SOA and Cloud.

With the increased adoption of SOA and Cloud in sensitive application domains, the demands for security increase as well. The use of message-oriented security for business information based on standards such as Universal Business Language (UBL)[1], eXtensible Business Reporting Language (XBRL)[2] and Bank Internet Payment System (BIPS)[3] is one example. Others can be found in the e-government domain where in Europe, e.g., the digital agenda explicitly includes SOA concepts as one building block for establishing ICT for public services. Based on pan-European interoperable e-signatures and e-IDs, SOA is recognized as the enabler for cross-border interoperability of e-government systems (Ticau, 2010). Many pilots build upon these foundations such as the e-PRIOR platform for e-procurement, which key design principles include standardized XML documents and SOAP Web Services[4].

XML Encryption (Imamura et al., 2002) and XML Signature (Bartel et al., 2008) are the core security standards to protect XML data. Recent research results show, however, that these upmost important security mechanisms include serious flaws. Amongst them, the so-called XML Signature Wrapping attack is the most relevant one. As described in detail in Section 2, this attack bypasses security means based

[1]http://ubl.xml.org/
[2]http://www.xbrl.org/
[3]http://www.bits.org/
[4]http://www.osor.eu/projects/openeprior

Mainka C., Jensen M., Lo Iacono L. and Schwenk J..
XSpRES - Robust and Effective XML Signatures for Web Services.
DOI: 10.5220/0003925701870197
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 187-197
ISBN: 978-989-8565-05-1
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

on digital signatures. Thus, from the XML Signature Wrapping attack arises a serious security threat which is of practical relevance especially in scenarios as the ones depicted above. Therefore, the XML Signature Wrapping attack needs to be carefully considered and treated in these environments.

The proposed countermeasures—if effective at all—usually provide protection only in very specific settings. No comprehensive approach is available yet. This paper contributes such a holistic and integrated approach—named XML Spoofing Resistant Electronic Signature (XSpRES)—by providing an architecture and an open-source implementation of an XML Signature library that enables the standard-compliant, robust and effective protection of XML data.

Although the emphasize of the developments has been on protecting SOAP-based Web Services and the paper will remain focused on this, still the general protection strategies can be applied to other XML messages or documents to form equivalent solutions.

The rest of the paper is structured and organized as follows. The next section provides a brief review of the technologies necessary for understanding the paper. It introduces the required background and discusses the available countermeasures in particular. It concludes by motivating the lack of a comprehensive defense approach which is effective and in conformance with current standardization. Section 3 then introduces an architecture that describes such a comprehensive approach named XSpRES. It integrates and combines relevant components out of the already available work with newly developed ones which are deployed on the client as well as on the server side. An in-depth description of the components and their implementation is part of Section 4 and 5. Section 6 analyzes the developments in terms of effectiveness and efficiency. The paper concludes with future work in Section 7.

2 XML SIGNATURE WRAPPING ATTACK FOUNDATIONS

To build the foundations for this paper, the basics behind XML Signature Wrapping attacks are described in the following accomplished with an analysis of available countermeasures.

2.1 XML Signature

XML Signature (Bartel et al., 2008) is the standard protection means for XML data. It specifies how to digitally sign XML fragments for ensuring integrity and proofing authenticity. The XML Signature element has the following (slightly simplified) structure:

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod Algorithm="..."/>
    <SignatureMethod Algorithm="..."/>
    <Reference URI="..." >
      <DigestMethod Algorithm="...">
      <DigestValue>...</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
</Signature>
```

The signing process undertakes the following flow: for each document part to be signed, a `Reference` element is created and the corresponding part is canonicalized and hashed. The resulting digest is added into the `DigestValue` element and a reference to the signed message part is inserted into the `URI` attribute. Finally the `SignedInfo` element is canonicalized and signed. The result of the signing operation is placed in the `SignatureValue` element.

2.2 XML Signature Wrapping Attack

The so-called XML Signature Wrapping attack introduced in 2005 by McIntosh and Austel (McIntosh and Austel, 2005) illustrates that the naive use of XML Signature may result in signed XML documents remaining vulnerable to undetectable modifications. Thus, with the typical usage of XML Signature an adversary may be able to alter valid documents in order to gain unauthorized access to protected resources.

In general, the attack injects unauthorized data into a signed XML document alongside with a possible reconstruction of that document so that the integrity and authenticity is still verified but untruly verified. The consequence is that the undetected modifications are treated as authorized input during any further processing steps.

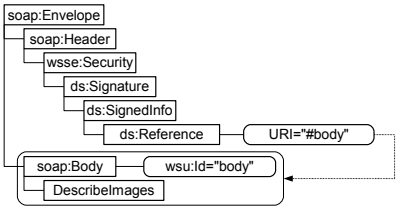

Figure 1: Signed SOAP message.

To illustrate this attack, let's assume that an attacker intercepts an XML-based SOAP message. The slightly simplified structure and content of the obtained SOAP message is shown in Figure 1. The message addresses a Web Service interface for a particular

Cloud service, that allows controlling the Cloud resources via such a SOAP-based API. In this example, the intercepted message has been issued by the legitimate user in order to get an overview of the available virtual machine images. The attacker needs to transform the operation in the SOAP body in order to reach the attack goals.

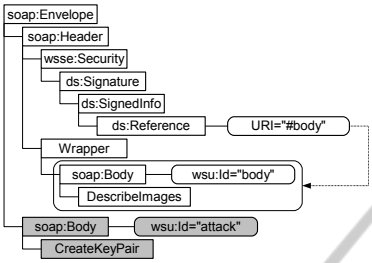

Figure 2: XML signature wrapped SOAP message.

One possible result of a modified SOAP message based on XML Signature Wrapping is shown in Figure 2. The original SOAP body element is moved to a newly added bogus wrapper element in the SOAP security header. Note that the moved body is still referenced by the signature using its identifier attribute `Id="body"`. The signature is still cryptographically valid, as the body element in question has not been modified (but simply relocated). Subsequently, in order to make the modified message again compliant to the XML Schema of SOAP messages, the attacker changes the identifier of the cogently placed SOAP body (in this example the `Id="attack"` is used). In this newly added and still empty SOAP body the attacker can now enter any of the operations defined by Cloud control API. In the given example, the adversary initiates a key generation process on behalf of the legitimate user being attacked.

2.3 Related Countermeasures

Since the discovery of XML Signature Wrapping attacks by McIntosh and Austel, several countermeasures against them have been proposed and intensively discussed in the last years. In fact, McIntosh and Austel themselves discussed in their original paper on XML Signature Wrapping attacks what requirements a server-side security policy must contain in order to uncover the attack. The final policy included assertions such as:

- a signature must be present in the security header
- the element specified by `/soap:Envelope/soap:Body` must be referenced from the signature
- the element matching `/soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp` must be referenced from the signature
- the signature verification key must be provided by an X.509 certificate issued by a trusted CA

In 2009 Gruschka and Lo Iacono showed with the first practical XML Signature Wrapping attack that the proposed checks by McIntosh and Austel are not sufficient to effectively detect XML Signature Wrapping attacks (Gruschka and Lo Iacono, 2009).

Bhargavan, Fournet and Gordon also used security policies for fending XML Signature Wrapping attacks. They developed a formal model for policy verification (Bhargavan et al., 2005a) and derived from these results a policy adviser (Bhargavan et al., 2005b) for testing and generating security policies. They use an abstract policy language for proofing security properties and map between this proprietary language and WS-SecurityPolicy (Kaler and Nadalin, 2005). The policy adviser proposes the following security assertions:

- **Mandatory Elements:** `wsa:To`, `wsa:Action`, `soap:Body`.
- **Signed Elements:** all mandatory, `wsa:MessageID`, `wsu:Timestamp`.
- **Recommended:** use of X.509 certificates for authentication.

The security requirements for an incoming message to the Amazon EC2[5] service are not stated as a formal security policy but in a human readable form. These requirements are:

- **Mandatory Elements:** `wsu:Timestamp`, `soap:Body`, `wsse:BinarySecurityToken` containing X.509 certificate.
- **Signed Elements:** `wsu:Timestamp`, `soap:Body`.

One can see that the Amazon security policy fulfills the requirement from Bhargavan et al. except for the WS-Addressing (Gudgin et al., 2006) requirement, which have no influence on this attack (as AWS does not honor WS-Addressing headers). But despite the fact that these requirements were formally proven, obviously such a pure policy-driven approach is not sufficient for mitigating signature wrapping attacks. The gap between the formal policy requirement and a real-world policy checking application can still be misused for attacks. A further problem with the policy adviser approach is that strong restrictions to the security policy are made. For example a lot of elements are claimed as mandatory and the signature of the body is absolutely required which reduces the flexibility of SOAP security mechanisms. However, this is supposably not a restriction for most practical applications.

Most XML Signature Wrapping attacks modify the structure of the original message from the legitimate sender in some way. In the example attack

[5] http://aws.amazon.com/ec2

described in Section 2.2 a second SOAP body is inserted by the attacker while the original one is being relocated into the SOAP header. Therefore, Rahaman, Schaad and Rits introduced a method—called *inline approach*—to protect some key properties of the SOAP message structure (Rahaman et al., 2006). In this system some characteristic information are collected over the SOAP message and inserted into a new element called *SOAP Account*. This element is added to the SOAP header and additionally signed by the sender. The protected properties are:

- Number of child elements of `soap:Envelope`.
- Number of child elements of `soap:Header`.
- Number of references in each signature.
- Successor and predecessor of each signed object.

If an attacker changes the structure of the message in a way that one of these properties are modified the attack can be uncovered. This is for example true for the example attack given in Section 2.2. The number of child elements of the SOAP header is changed from one to two. Thus the usage of the inline approach would have detected this attack. Nonetheless, this protection method has some disadvantages. First, the introduced SOAP Account element as well as the verification of this element is not standardized. Thus, it can for example not be claimed by a WS-SecurityPolicy (Lawrence and Kaler, 2007). Second and more importantly, this method does not generally protect from XML Signature Wrapping attacks. If an attacker is able to modify the message structure while keeping the structure properties the inline approach can be circumvented as has been shown by Gajek, Liao and Schwenk in (Gajek et al., 2007). They improve the above discussed inline approach, but for this improved version the just mentioned disadvantages—especially the standardization issue— still remain.

The authors of (Gajek et al., 2007) give in their paper some more solution ideas for fending XML Signature Wrapping attacks. The main idea is using the verification component as a filter. In contrast to common methods where the signature verification just returns a boolean value, here the result of the transformation and canonicalization step is returned. This ensures that the following processing entities inside the Web Service framework operate truly on the message that was originally signed. One problem of this approach—as already remarked by the authors—is that the Web Service cannot operate on the SOAP envelope as a single well-formed message document but only on parts of the message which may also be divided into a forest of message trees. This problem can be solved by passing the signed elements together with its parent nodes as a spanning DOM tree to the business logic part of the Web Service. This works only, if the signature transformation does not change the content of the elements. But even this improved version is inadequate if the Web Service operates on signed as well as on unsigned parts of the SOAP message. In this case the signature component cannot operate as a filter.

Until 2009 this research work was mainly treated as theoretical, due to the rare usage of WS-Security in sensitive applications and the absence of a real-life XML Signature Wrapping attack. In 2009 it was discovered, that Amazon's Cloud services were vulnerable to XML Signature Wrapping attacks (Gruschka and Lo Iacono, 2009). Using a variation of the attack example presented in Section 2.2 an attacker was able to perform arbitrary operations in the Cloud on behalf of a legitimate user. A number of related results have been published in the following, leading to the discovery of novel attack instances (Somorovsky et al., 2011).

Along with these developments, novel defense techniques have been proposed. In (Gajek et al., 2009) the authors turn toward a major characteristic of the XML Signature Wrapping attack, which is the missing confidence of location information when using ID-based references to link the XML Signature metadata to its signed content. Since the ID attribute does not provide any details on the signed content's location in the document (and sometimes not even on its property of uniqueness), the default referencing scheme possess major challenges with respect to XML Signature Wrapping attacks. Hence, the use of a location-aware referencing scheme is favorable. With XML Signature, this can be achieved using XPath expressions for referencing. For instance, an XPath of /soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp leaves little doubt on which part of the XML document is intended to be protected by a digital signature. Furthermore, an attacker can only trick this reference in an XML Signature Wrapping attack if he manages to trick the XPath evaluation into mapping to another XML subtree than anticipated—a way more challenging task as compared to moving an ID-equipped subtree to an arbitrary location within the XML document tree as has been used in the above example.

A critical issue with respect to XPath-based referencing is the robustness of the actual XPath expression. For instance, an XPath expression of `//*[@ID="foo"]` is a valid XPath expression, but is equivalent to an ID-based reference with respect to XML Signature Wrapping attempts. As analyzed in (Gajek et al., 2009), the most favorable subset of XPath when it comes to XML Signature Wrapping

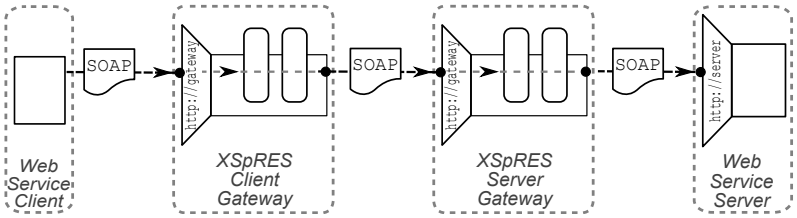

Figure 3: XSpRES Architecture overview.

robustness is the subset called *FastXPath*, which prohibits the use of several potentially vulnerable XPath properties. For instance, the use of so-called *wildcard axes* such as `descendant::` are prohibited, since they allow an attacker to move the referenced XML fragment arbitrarily within the range of that XPath location step. Analogously, the use of position indicator predicates is mandatory in FastXPath. An XPath of `/soap:Envelope[1]/soap:Body[1]` clearly selects exactly one—the first—child element of matching name, thereby fending XML Signature Wrapping attacks that use element duplication (as e.g. used against Amazon EC2 in (Gruschka and Lo Iacono, 2009)).

In (Jensen et al., 2011) the authors introduce and evaluate the possibility of hardening the XML Schema in conjunction with XML Schema validation to construct an effective protection against XML Signature Wrapping attacks. The authors identified the following weak definitions in the SOAP Schema:

- **Element:** `xs:any` allows an element to have any kind of child elements, which are not defined by any schema. The main idea of this is to have an extensible layout, e.g. the SOAP `Header` can have any not yet defined child elements (*reserved for future use*).
- **Attribute:** `processContents="lax"` instructs the validator to only process the content if an XML Schema for it is present, otherwise just leave it out. A more drastically direction is `processContents="skip"` which causes simply no validation.
- **Attribute:** `namespace="##any"` and `namespace="##other"` allows the usage of elements from *any*, respectively from *any but its parents* namespace.

Each of these nodes allows an attacker to inject own elements, e.g. to place a XML Signature Wrapping element. To eliminate this leakage, each instruction is removed from the hardened Schema by substituting it directly with the needed Schema parts (e.g. WS-Security). This constricts the whole document to deny any user-defined elements. It has been shown that XML Schema validation with a hardened XML Schema is capable of fending XML Signature Wrapping attacks, but bears some pitfalls and disadvantages amongst which the increased resource consumption is the biggest obstacle.

This analysis of the related work can be concluded by noting the fact, that the XML Signature Wrapping attack has been moved from a hypothetical to a practical security threat which needs to be urgently targeted and that there is no comprehensive approach available which provides the required protection against this attack in a standard-compliant and effective manner. This lack is targeted by XSpRES as will be introduced in the following.

3 THE XSPRES APPROACH AND ARCHITECTURE

As can be seen from the discussions on available countermeasures, several proposals for fending the XML Signature Wrapping attack threat exist. Unfortunately, each countermeasure has shown to become ineffective at some point for certain XML Signature Wrapping attack variations. Hence, in order to establish a robust protection, it is necessary to combine a suitable subset of these countermeasures to come up with a holistic, integrated approach.

3.1 Attack Model

An initial step for the development of such a holistic defense architecture against XML Signature Wrapping attacks consists in the definition of a formalized model for the attack scenario. Based on the existing previous work presented in (Gajek et al., 2009; Jensen et al., 2009), and derived from the semi-formal *Web Services Attacker Model* described in (Jensen, 2011), the scenario model for the XML Signature Wrapping attack consists of three main entities: (1) a Web Service client, (2) a Web Service server (3) and an external attacker (cf. Figure 4). The attacker is assumed to be able to access and alter XML messages exchanged between the Web Service client and the Web Service server, but is not able to interfere with the client-side or server-side implementations of the Web Services

software stack directly.

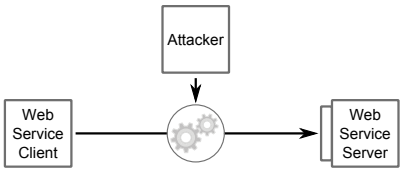

Figure 4: Attack Model for XML Signature Wrapping.

Based on this abstract yet formal scenario model, the XSpRES approach derives the ability to extend the Web Service stacks at both Web Service client and Web Service server arbitrarily, as long as these extensions remain within the particular trusted domain and outside of the scope of the attacker. Thus, the XSpRES architecture introduces two new entities to the scenario: a client-side extension and a server-side extension. Depending on the real-world instantiation of this model, these extensions can be realized as additional modules in the particular Web Services software stack or as external gateways interlinked to the network connection between client and server. For the proof-of-concept implementation of the XSpRES approach (which is discussed in detail in Sections 4, 5 and 6) the realization as gateways has been chosen, as shown in Figure 3.

Either way, the two new entities are responsible for performing a set of additional processing and verification steps in relation to the creation and verification of signed XML messages.

3.2 Client-side Approach

Though XML Signature Wrapping attacks commonly only affect the Web Service server operations, there nevertheless exists the need to consider also the Web Service client-side to improve the overall robustness of the secured XML-based communication. The goal of the XSpRES client-side processing is to bind the XML Signature in the course of signature creation as uniquely and strongly as possible to its referring content. This is achieved by chaining a set of separate modules, which seamlessly integrate into the client-side Web Service processing flow. The developed modules deal with *Referencing Verification*, *Prefix Transformation* and the *Signature Creation* itself (cf. Figure 5). Each of these components is described in detail in Section 4.

3.3 Server-side Approach

In correspondence with the client-side approach, the server-side Web Service processing flow is extended using an integrated architecture of five modules that protect the server-side implementation from being compromised by a XML Signature Wrapping attack. The goal of the XSpRES server-side processing is to extend the verification steps to include checks on the message structure and to evaluate the strict and unique binding enforced by the client-side modules. The developed modules deal with the tasks of *Denial of Service Detection*, *XML Schema Validation*, *Referencing Verification*, *Prefix Transformation* and the *XML Signature Verification* itself (cf. Figure 6). Each of these modules is described in detail in Section 5.

3.4 Backwards Compatibility and Standards Compliance

An important characteristic of the XSpRES architecture is the *backward compatibility*, meaning that all parts of the XSpRES architecture are able to handle communication not originating from another XSpRES-instrumented client or server. More precisely, the server-side XSpRES extension is able to process arbitrary types of XML Signatures, even if they originate from a different XML Signature creation framework than the XSpRES client-side modules. Vice versa, the XML Signatures generated by the XSpRES client-side modules remain fully compliant to the XML Signature specification, hence can also be verified by any other XML Signature verification implementation, even if it is not following the XSpRES approach.

However, obviously, the effectiveness of the XSpRES defense against the XML Signature Wrapping attack is reduced drastically by using non-XSpRES components at either side, since this breaks the comprehensiveness of XSpRES falling back to the present state of the art.

4 THE XSPRES CLIENT-SIDE MODULES

The XSpRES client-side modules ensure the creation of secure signatures. Figure 5 illustrates the process flow. At first, the WS-SecurityPolicy is verified to only use FastXPath expressions. Afterwards, those expressions are transformed to their prefix-free equivalent. Finally, the document is signed using standard mechanisms but building on the previous processing steps.

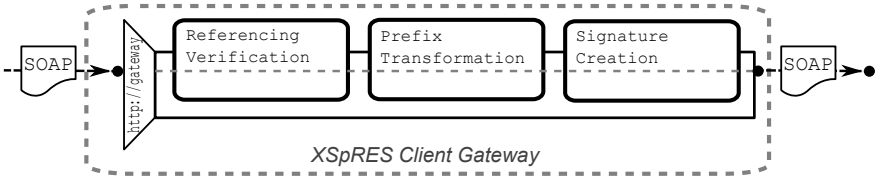

Figure 5: The XSpRES client-side modules and message processing flow.

4.1 Referencing Verification

As discussed in Section 2.3, when sticking to the FastXPath subset defined in (Gajek et al., 2009), the robustness of the resulting XML Signatures against XML Signature Wrapping is improved significantly. XSpRES integrates FastXPath-based referencing on the client-side for binding the XML Signature metadata more tightly to its signed contents.

The *Referencing Verification* module of the XSpRES client-side is not directly accessing the SOAP messages, but is merely used to preprocess the WS-SecurityPolicy file that defines which parts of the bypassing SOAP messages are to be signed. Commonly, the WS-SecurityPolicy document is created and provided by the Web Service server, describing its expectations with respect to signed parts in SOAP messages it is targeted with. Hence, it acts as a basis for determining the parts-to-be-signed at the client-side. However, if the XPath expressions given in the WS-SecurityPolicy file are not following the FastXPath grammar as discussed above, their effectiveness in terms of fending XML Signature Wrapping attacks are reduced. Thus, this module's task is to preprocess all XPath expressions given in the server-provided WS-SecurityPolicy document, verifying that all of the contained XPath expressions strictly stick to the FastXPath grammar. If the WS-SecurityPolicy file fulfills this requirement, it is used in the FastXPath-based referencing module introduced previously, simply by using the FastXPath expressions from the WS-SecurityPolicy document as the reference in the XML Signatures created. Otherwise, the module throws an error and stops the client gateway.

4.2 Prefix Transformation

As pointed out in (Jensen et al., 2009), the use of XPath in conjunction with XML namespaces has its issues, potentially leading to an exploitable XML Signature Wrapping vulnerability despite the existence of a strict XPath expression for referencing. By binding the same namespace prefix to different namespace URIs at different locations within a signed SOAP message document, the server-side processing stack can be misled into a XML Signature Wrapping attack, even when using a strict FastXPath expression.

The *Prefix Transformation* module transforms the FastXPath expressions from the server-provided WS-SecurityPolicy into an equivalent, but prefix-free variant. The FastXPath expression of `/soap:Envelope[1]` gets e.g. transformed into the semantically equivalent XPath expression of `/*[local-name()="Envelope" and namespace-uri()="http://www.w3.org/2003/05/soap-envelope"][1]`. As can be seen, the problematic use of the `soap:` prefix is resolved into an equivalent representation holding the full namespace URI of the SOAP specification. Hence, binding the `soap:` prefix to another namespace URI does no longer affect the result of the transformed XPath expression, protecting against the XML Signature Wrapping threat of Namespace Injection as described in (Jensen et al., 2009).

In principle, it would be a valid approach to use the transformed version of the FastXPath expressions already within the WS-SecurityPolicy document itself, however, due to limited readability and considerations in respect to available tools, the XSpRES prototype sticks to the presented XPath transformation approach. This implies, that after the successful FastXPath compliance verification of each XPath expression extracted from the server-provided WS-SecurityPolicy, this expression is transformed into the prefix-free notation, and then used directly within the created XML Signature as XPath Filter2 referencing expression. Thereby, it can be guaranteed that a location change of a signed XML subtree within a SOAP message automatically causes an invalidation of the respective XML Signature. Even without a server-side defense mechanism, this approach provides already an effective protection against numerous XML Signature Wrapping attack variations. However, since the server-side application logic itself might not be implemented in a way that recognizes and uses the same fixed position as input for its operations, XML Signature Wrapping attacks remain possible even with using the prefix-free FastXPath referencing as outlined here. Thus, a strong server-side defense is still required in addition, in order to further reduce possible sources of errors and to decouple the verification processes from the application logic pro-

cesses as much as possible.

4.3 Signature Creation

The *Signature Creation* module then creates the XML Signature for the to be signed document parts. The signature creation is in conformance with the corresponding standards. To reach the targeted goal of a more unique and strict binding of the signature to its content all references are given as prefix-free FastXPath expressions.

5 THE XSPRES SERVER-SIDE MODULES

The XSpRES server-modules use a modular chain of detectors to identify an attack as early as possible. Figure 6 illustrates the processing flow. First, the incoming message is processed by the *DoS Protector module* to ensure that the XML document is of finite length. The *XML Schema Validator module* uses a hardened XML Schema to guarantee that there are no unexpected elements. The *Referencing Verification module* and the *Prefix Transformation module* assure that the signed parts are accessed by the correct FastXPath expression and the *XML Signature Verification module* finally verifies the signature.

5.1 DoS Detection

The *DoS Detection* module is not a required component from the XML Signature Wrapping perspective, but is a general must have protection. Such a DoS detector ensures that the XML document is of finite length and henceforth prevents the overflooding of the machine's memory – especially in the case in which a DOM based parser is used, since each element in the XML message is instantiated as an object in memory.

The XSpRES system includes a DoS detector and uses it to also check on the appearance of ID attributes. If an ID attribute occurs twice, the processing of the message on the server-side is aborted. This addition to the DoS Detector suppresses basic XML Signature Wrapping attacks, in which the signed message part is duplicated or moved (including the ID attribute), in an very early processing stage.

5.2 XML Schema Validation

The *XML Schema Validation* module uses a hardened XML Schema to validate the incoming SOAP messages. This schema overrides the default XML Schema for SOAP messages by removing any possibility for placing arbitrary elements in the document as described in (Jensen et al., 2011) and pointed out in Section 2. The drawback of this approach is that the schema validation of a hardened XML Schema is slower compared to the standard one, as each element must be validated. Therefore, the XSpRES implementation merges as few schemas as possible to minimize the total schema size. The considered schemas include WS-Security, WS-Utilities, XML Signature, XPathFilter2 and WS-Addressing (whereas the latter might also be negligible depending on the requirements of the underlying application scenario).

Note, that the server gateway schema must be adjusted to the document structure of each to be protected Web Service, since the SOAP body element does no longer allow `xs:any` child elements.

5.3 Referencing Verification

The *Referencing Verification* module verifies the security policy. Therefore, it extracts the XPath expressions from a local policy file and, analogue to its client-side complement, validates if these are valid FastXPath expressions. This ensures that the horizontal and vertical position of the signed fragments is fixed.

5.4 Prefix Transformation

Afterwards, the *Prefix Transformation* module transforms the FastXPath expressions to their namespace-free equivalents to prevent namespace injection attacks.

In contrast to the corresponding client-side module, the received message is checked in addition. An incoming message is parsed and a lookup for a valid `Timestamp` element is made. Thereafter, the expressions in the XPath element children of the signature's `Reference` element are string-compared to those transformed FastXPath expressions from the policy file. This assures that both, the client and the server side, use the same policy.

5.5 Signature Verification

The *XML Signature Verification module* simply verifies the XML Signature in the document. As the WS-SecurityPolicy modules assures that the signed elements use the correct XPath expressions, the signed fragments are horizontally and vertically fixed, so that no known attack moved these message parts.

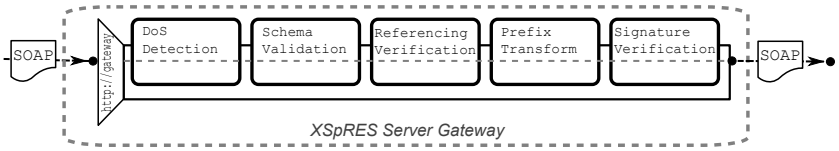

Figure 6: The XSpRES server-side modules and message processing flow.

6 IMPLEMENTATION AND EVALUATION

The implementation of each single XSpRES component is realized by only standard Java libraries. The client gateway acts as a simple HTTP server and signs an incoming message by using the FastXPath expressions extracted from a local policy file. These expressions are validated and transformed as described in Section 3.2. The signed message is afterwards forwarded to the server gateway.

The server gateway is based on the Apache Axis2[6] Web Services Framework and the XSpRES components are integrated as an Axis2 module. Thus, by this integration method, the XSpRES module can replace the signature verification of the commonly used Apache Rampart[7] security module.

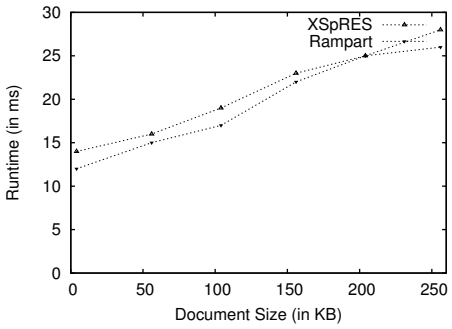

Figure 7: Runtime comparison of XSpRES and Rampart.

Figure 7 compares the signature verification of the Rampart module with all security features of the XSpRES prototype. The setup uses an AMD Athlon II X3 440 Processor with 4GB RAM.

Both, the XSpRES and the Rampart module do semantically the same: They validate the timestamp in the message header and verify the signature over the `Timestamp` element as well as the SOAP body element. The technical difference is that Rampart uses two ID-based references, one for the timestamp and the other for the SOAP body. XSpRES instead uses only one reference which selects both elements with the transformed FastXPath expressions from the policy file.

The measurement of the time required to verify the signature starts after the HTTP request has been received and ends just before the verified message is forwarded to the application logic. The processing time is then computed as the average of 1000 messages. The measurements have been conducted for different message sizes.

As can be seen from the visualized evaluation results given in Figure 7 both modules operate approximately equal in speed, but the Rampart module has the lack of the additional security features in relation to XML Signature Wrapping attacks. It is also notable, that the runtime for Rampart is the same for any kind of invalid messages, whereas the XSpRES module will abort the verification process in an early stage if the message violates the schema or the policy.

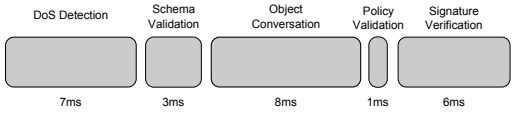

Figure 8: Runtime analysis of single XSpRES components for a 200KB message.

A detailed runtime analysis for each XSpRES component is shown in Figure 8 for one common message size. The DoS detection takes 7ms although the StAX parser has to process the whole document. The DOM based Schema validation needs only 3ms, because the instantiation of the XMLSchemaFactory, which processes the Schema files, is done once in the startup phase and thus saves 9ms per validation. The signature component can be divided into three parts: The slowest part is the conversion from the Axis2 Object Model to a Java Document Object. The policy validation is extremely fast, as it just searches for the transformed FastXPath expressions in the `Reference` element and string-compares them to those in the local policy file. The signature validation requires 6ms. In its current implementation, XSpRES uses the common DOM-based processing model. In future work, this will be replaced by a faster streaming-based signature validation (Gruschka et al., 2011; Somorovsky

[6]http://ws.apache.org/axis2/

[7]http://axis.apache.org/axis2/java/rampart/

et al., 2010).

It must be mentioned, that the XSpRES prototype implementation is focused on security and not on efficiency in the first place. Nevertheless, Figure 7 shows that it is comparable to the Rampart security module. One reason for this is, that XSpRES is very light-weight, i.e., it is only capable of handling digital signatures. Rampart, in contrast, is much more complete in the sense of standards-compliance, including features such as encryption, username token, which impacts on the time required for signature verification. On the other hand, the processing time of the XSpRES modules can be improved significantly. By aligning the object models of the various deployed Java components, the need for the costly object conversion would be eliminated, reducing the processing time by one third.

7 CONCLUSIONS

The use of SOA and Cloud concepts for the construction of distributed applications handling more and more sensitive data is on the rise. XML is playing an important role in such applications, since it is used for encoding data at rest as well as data in transit. The security demands coming with the processing and storage of sensitive data rely on robust and effective security technologies. Recent discoveries showed that the XML Security specifications include serious flaws and thus can currently not fulfill the required protection levels readily.

This paper contributes a comprehensive approach to face these vulnerabilities providing an architecture which compiles a set of inter-linked protection mechanisms for the client-side as well as the server-side. The selection and composition of the protection means have been guided by the requirements to realize the targeted architecture in an effective, but still standard-compliant and cost efficient way. Based on this ground work, an open-source XML Signature Wrapping attack protection library has been implemented, which is robust against all known instances of the XML Signature Wrapping attack and thus provides a vehicle to generate and verify signed XML documents and messages in a fail-safe and standard-compliant manner. The seamless integration of the developed library into standard Web Services frameworks has been another requirement, enabling—amongst others—to evaluate the developments in a common SOAP setting. The obtained results emphasize that the proposed approach fulfills the targeted goals and provides an effective protection against XML Signature Wrapping attacks at low computational extra costs and by still being standard-compliant.

Future work will target the publishing of the presented XML Signature Wrapping protection library XSpRES. Additionally, a more in-depth and formal analysis of the protection provided by XSpRES in case of not yet known variations of the XML Signature Wrapping attack will bexy performed.

ACKNOWLEDGEMENTS

This work was funded by the Federal Office for Information Security in Germany (BSI) under the contract number 882/2010.

The authors would like to thank Holger Junker and Juraj Somorovsky for many fruitful discussions and their valuable input.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., and Simon, E. (2008). XML Signature Syntax and Processing. *W3C Recommendation*.

Bhargavan, K., Fournet, C., and Gordon, A. D. (2005a). A semantics for Web Services authentication. *Theoretical Computer Science*, 340(1):102–153.

Bhargavan, K., Fournet, C., Gordon, A. D., and O'Shea, G. (2005b). An advisor for Web Services Security policies. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 1–9, New York, NY, USA. ACM Press.

Gajek, S., Jensen, M., Liao, L., and Schwenk, J. (2009). Analysis of signature wrapping attacks and countermeasures. In *ICWS*, pages 575–582.

Gajek, S., Liao, L., and Schwenk, J. (2007). Breaking and fixing the inline approach. In *Proceedings of the 2007 ACM Workshop on Secure Web Services (SWS'07)*, pages 37–42, Fairfax, Virginia, USA. Association for Computing Machinery.

Gruschka, N., Jensen, M., Lo Iacono, L., and Luttenberger, N. (2011). Server-side streaming processing of ws-security. *IEEE T. Services Computing*, 4(4):272–285.

Gruschka, N. and Lo Iacono, L. (2009). Vulnerable Cloud: SOAP Message Security Validation Revisited. In *ICWS '09: Proceedings of the IEEE International Conference on Web Services*, Los Angeles, USA. IEEE.

Gudgin, M., Hadley, M., and Rogers, T. (2006). Web Services Addressing 1.0 - SOAP Binding. *W3C Recommendation*.

Imamura, T., Dillaway, B., and Simon, E. (2002). XML Encryption Syntax and Processing. *W3C Recommendation*.

Jensen, M. (2011). *Analysis of Attacks and Defenses in the Context of Web Services*. PhD thesis, Ruhr-University Bochum.

Jensen, M., Liao, L., and Schwenk, J. (2009). The curse of namespaces in the domain of xml signature. In *SWS*, pages 29–36.

Jensen, M., Meyer, C., Somorovsky, J., and Schwenk, J. (2011). On the effectiveness of xml schema validation for countering xml signature wrapping attacks. In *First International Workshop on Securing Services on the Cloud (IWSSC 2011)*.

Kaler, C. and Nadalin, A. (2005). Web Services Security Policy Language (WS-SecurityPolicy) 1.1.

Lawrence, K. and Kaler, C. (2007). Web Services Security Policy Language (WS-SecurityPolicy) 1.2.

McIntosh, M. and Austel, P. (2005). XML signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 20–27, New York, NY, USA. ACM Press.

Rahaman, M. A., Schaad, A., and Rits, M. (2006). Towards secure SOAP message exchange in a SOA. In *SWS '06: Proceedings of the 3rd ACM workshop on Secure Web Services*, pages 77–84, New York, NY, USA. ACM Press.

Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., and Lo Iacono, L. (2011). All your clouds are belong to us security analysis of cloud management interfaces. In *Proceedings of the ACM Cloud Computing Security Workshop (CCSW)*.

Somorovsky, J., Jensen, M., and Schwenk, J. (2010). Streaming-based verification of xml signatures in soap messages. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 637–644, Washington, DC, USA. IEEE Computer Society.

Ticau, S.-A. (2010). Security – a centrail issue of the future EU digital agenda. *Service Oriented Architecture pushed to the limit in eGovernment*.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS