On the Effectiveness of XML Schema Validation for Countering XML Signature Wrapping Attacks

Meiko Jensen, Christopher Meyer, Juraj Somorovsky, and Jörg Schwenk
Chair for Network and Data Security
Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
{meiko.jensen, christopher.meyer, juraj.somorovsky, joerg.schwenk}@rub.de

***Abstract*—In the context of security of Web Services, the XML Signature Wrapping attack technique has lately received increasing attention. Following a broad range of real-world exploits, general interest in applicable countermeasures rises. However, few approaches for countering these attacks have been investigated closely enough to make any claims about their effectiveness.**

In this paper, we analyze the effectiveness of the specific countermeasure of XML Schema validation in terms of fending Signature Wrapping attacks. We investigate the problems of XML Schema validation for Web Services messages, and discuss the approach of Schema Hardening, a technique for strengthening XML Schema declarations. We conclude that XML Schema validation with a hardened XML Schema is capable of fending XML Signature Wrapping attacks, but bears some pitfalls and disadvantages as well.

***Index Terms*—Security, XML Signature, Signature Wrapping, XML Schema, Schema Validation, Schema Hardening hardest argument against**

I. Introduction

One of the cornerstones of security in the cloud computing age consists in the robustness of its underlying technologies. A vulnerability in the virtualization or network stack used by a cloud provider, for instance, puts all of the cloud customers' data and processes at risk (cf. [1], [2]). Since many of today's cloud offers utilize XML-based technologies like Web Services for accessing and controlling the cloud, these are of particular importance for the security assessment of cloud systems.

In this paper, we analyze one of the most critical types of vulnerability found for XML-based technologies so far: the XML Signature Wrapping attack [3], [4], [5]. Based on a fundamental flaw in W3C's XML Signature specification [6], the number of vulnerable application stacks identified lately was tremendous (see e.g. [7], [8], [2], [9]), and is still on the rise.

When coping with this type of threat, the validation of a Web Service request's XML Schema is often cited as a sufficient countermeasure for fending most of such XML-based attacks (cf. [10], [11], [12]). However, as we show in this paper, server-side XML Schema validation of incoming XML message documents is not capable of fending the XML Signature Wrapping attack completely. We present our analysis on the actual effectiveness of XML Schema validation, giving examples of attack schemes that are fended this way. However, we also show that even the most restrictive XML Schema validation has its limitations, allowing Signature Wrapping attacks to succeed even in presence of such validation at the server side.

The paper is organized as follows. The next section provides a brief review of the technologies necessary for understanding the rest of the paper. Section III then provides our anaylsis on the interrelation of Signature Wrapping vulnerability and XML Schema validation in general. Section IV analyzes the approach of Schema Hardening. Afterwards, Section V underpins this analysis with real-world examples, and the paper concludes with future work in Section VI.

II. Foundations

In this section we will give an overview about the usage of XML Signature, XML Schema, and the Web Services related specifications.

A. XML Schema Validation

The W3C XML Schema set of specifications allows for precise description of any XML document's contents. By defining the XML element tree's structure, element names, data types, attributes etc., an instance of an XML Schema allows for validating conformance of any XML document against that schema instance. If a particular XML document successfully passes such a validation, it is called a *valid* XML document w.r.t. the given XML Schema instance.

Though being a very powerful language for restricting the actual apperance of an XML document (e.g. a Web Service message), the active use of XML Schema validation is often omitted in XML-processing applications due to performance reasons. However, recent works have shown that missing XML Schema validation in Web Service server systems enables various XML-based attack vectors (cf. [12], [13]). Hence, the market for XML firewalls capable of performing XML Schema validation on-the-fly is emerging, leading its customers to the false assumption that such a firewall also fends all other types of XML-related attacks—such as the XML Signature Wrapping attack.

B. WS-* Specifications

When considering active XML Schema validation in the context of Web Service message verification, it is important

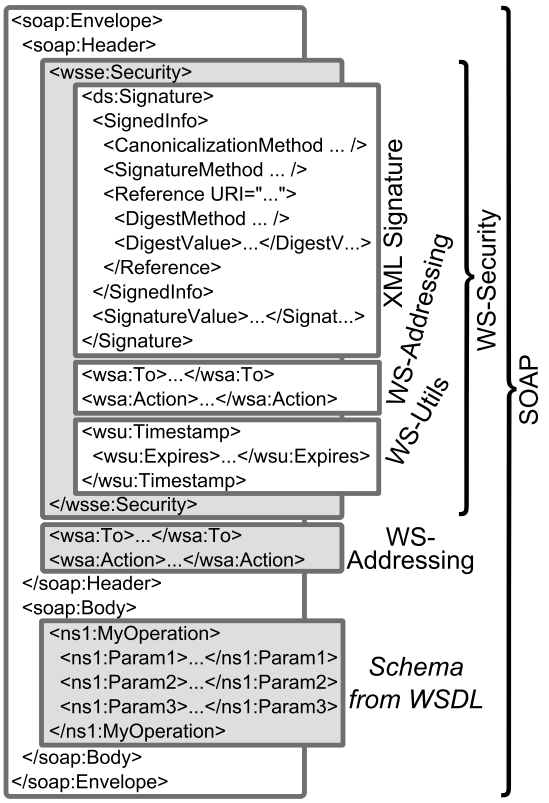

Fig. 1. Interrleation of common WS-* specifications for a single SOAP message

to understand the set of WS-* specifications that define the XML Schema of each single Web Service message. First and most important, the SOAP specification defines an initial XML Schema that any SOAP-based Web Service message has to be valid against. It mostly describes a document's `<Envelope>`, consisting of an optional SOAP `<Header>` and a mandatory SOAP `<Body>`. The SOAP body contains the actual Web Service requests; its XML Schema is hence described in the Web Service Description (WSDL), which contains definitions of XML Schemata for each type of operation a Web Service supports. In contrast to SOAP, the XML Schema to use for validation of the SOAP Body is dependent on the particular Web Service, and is not explicitly given in a standardized W3C[1] or OASIS[2] specification document. This leads to some challenges when trying to perform XML Schema validation for the SOAP Body, see GRUSCHKA ET AL. [10].

The SOAP header contains non-functional data that is used for processing a SOAP message at the recipient side. For instance, in common settings it contains some WS-Addressing headers [14] that provide message routing information, and a WS-Security header [15] that provides security-related information. The latter typically provides a message timestamp (as defined by the WS-Utilities Schema [16]), an XML Signature [6] covering critical parts of the SOAP message,

[1]http://www.w3c.org

[2]http://www.oasis-open.org

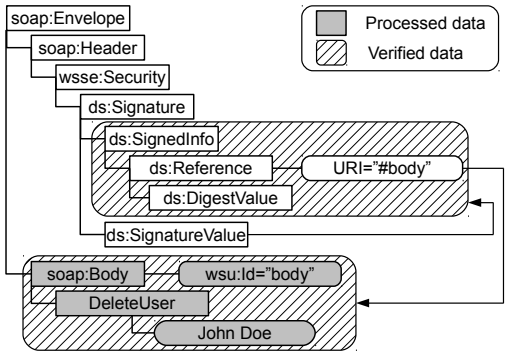

Fig. 2. Example of XML Signature applied on the SOAP body

and optionally some WS-Addressing information. Note that WS-Addressing headers can actually be used within both SOAP header or WS-Security header. Figure 1 shows the interrelation of these WS-* specifications and their particular XML Schemata in the context of a single SOAP message.

C. XML Signature Wrapping

In order to ensure integrity and authenticity of exchanged XML documents, the XML Security working group defined the XML Signature specification [6]. XML Signature allows to apply cryptographic primitives on the XML messages and thereby secure arbitrary XML elements. A simplified structure of an XML Signature applied to a SOAP message according to the WS-Security standard gives Figure 2.

The depicted SOAP message includes a function invocation deleting the user `"John Doe"`, which is defined in the body of the SOAP message. The authenticity and integrity of the SOAP body is ensured by the XML Signature defined in the SOAP header. The XML Signature element consists of two mandatory elements: `<SignedInfo>` and `<SignatureValue>`. The `<SignedInfo>` element includes an Id-reference[3] pointing to the SOAP body and a digest value computed over the referenced element. In order to secure the `<SignedInfo>` element, a signature value over this element is computed and put to the `<SignatureValue>` element. This is typically done by a public-key algorithm such as RSA or DSA.

Processing of the given SOAP message would usually look as follows: The recipient first *searches for the referenced element* given in `<SignedInfo>`. He computes the digest value over this element and compares it to the value given in the `<DigestValue>` element. Afterwards, he verifies the signature value over `<SignedInfo>`. At the end, he can execute the function *defined in the SOAP body*.

As can be seen, processing of the SOAP message and verifying the included XML Signature consists of *two independent steps*. This observation was first made by McIntosh and Austel [3], who misused the different processing parts for a new attack: XML Signature Wrapping. We give an

[3]XML Signature specification also supports usage of other referencing mechanisms. These mechanisms are however out of scope of our paper.

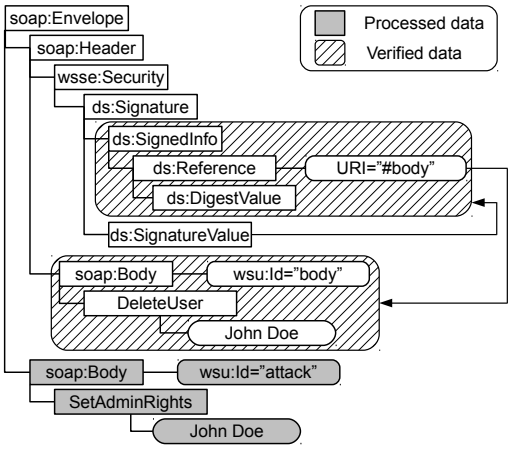

Fig. 3. Signature Wrapping attack

example of the XML Signature Wrapping attack in Figure 3. In this example an attacker who eavesdrops the message moves the original SOAP body to the SOAP header. Afterwards he creates a SOAP body with a new `Id="attack"` and defines an arbitrary function: in this example, he enforces the business logic to execute a function giving admin rights to `"John Doe"`. As the Id of the original SOAP body stays the same and the concerning parts are not altered, the security logic can verify its integrity and authenticity. The business logic however takes the newly defined SOAP body as input.

III. Schema Validation as a Countermeasure

As introduced XML Schema offers an opportunity to formally describe the structure of concerning XML documents. Limiting the attack surface through bounding element appearance to clearly defined positions in the document may lead to more secure and less vulnerable documents.
In order to come to the point schema validation can not be seen as the ultimate defense against Signature Wrapping attacks, but remains one brick towards Signature Wrapping resistant XML data structures.

The following subsections will outline how schema validation may tighten signature security within XML Signature containing documents, as well as its boundaries.

A. Fending the Classic Signature Wrapping Attacks

In 2009 Gruschka and Lo Iacono successfully attacked Amazons *EC2*[4] (refer to [7]) by doubling the body of a SOAP message. This attack could have been easily detected and defeated by proper schema validation at the processors side. The sample code snippet in Figure 4 is taken from the official XML Schema file for SOAP messages[5].

An important note is that one has to be aware of the fact that prior schema definitions defined default settings. Important for this example is the default value related to attribute `maxOccurs="1"`. So every time an element applicable for

[4]http://aws.amazon.com/ec2/
[5]http://www.w3.org/2003/05/soap-envelope/

```
01 <xs:complexType name="Envelope">
02  <xs:sequence>
03    <xs:element ref="tns:Header" minOccurs="0"/>
04    <xs:element ref="tns:Body" minOccurs="1"/>
05  </xs:sequence>
06  <xs:anyAttribute namespace="##other"
      processContents="lax"/>
07 </xs:complexType>
```

Fig. 4. Code Snippet from official XML Schema file for SOAP messages

a `maxOccurs` attribute that does not overwrite the default setting (by setting a new value) inherits value ”1”.

The code above defines a structure how a valid message has to look like. This includes the optional occurrence of a `tns:Header` element at line 03 (but only exactly ONE element is allowed according to the default setting `maxOccurs="1"`) and exactly one single, mandatory `tns:Body` element at line 04.

How could this prevent the prior introduced attack of doubling the body element within a SOAP message? According to the part of the schema definition above, direct childs of an envelope element may only be exactly one body element, optionally exactly one header element and an element of a different namespace (`namespace="##other"`). So if an attacker tried to double the body element within the envelope element, as a direct child, the schema validation will fail, rendering the attack unsuccessful.

Without going into detail, schema validation can also be used to protect against nested `<soap:Body>` tags. The concept remains the same, restricting valid elements at this position, but this reveals to be only one side of the medal as we can see in the following section.

B. Inefficiency in Presence of Potential Weakness Indicators

Validating incoming messages by schema validation may seem to be a promising approach to fight Signature Wrapping. But due to too flexible and permissive crafted standard schemata security issues arise which are not obvious at first sight.

For example the attack introduced by McIntosh and Austel in 2005 [3] where the `<wsse:Security>` in the `<soap:Header>` element is simply doubled, can not be countermeasured since the schema does not restrict doubling of elements from arbitrary - `"##any"` - namespaces (`maxOccurs="unbounded"` and `namespace="##any"`) as can be seen in Figure 5 at line 08.

Further attack variants such as locating a `<soap:Body>` element in a `<soap:Header>` or wrapping a `<soap:Body>` by any other element allowed at the place of occurence can not be countermeasured either.

The question arises what is repsonsible for this behaviour. We identified a couple of potential elements, attributes and their variants in usage that may indicate weaknesses in a schema definition. All occurences of these elements provide an attacker a surface for injecting arbitrary elements.

```
01 <xs:complexType name="Header">
02  <xs:annotation>
03    <xs:documentation>
04        Elements replacing the wildcard MUST be namespace
          qualified, but can be in the targetNamespace
05    </xs:documentation>
06  </xs:annotation>
07  <xs:sequence>
08    <xs:any namespace="##any" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
09  </xs:sequence>
10  <xs:anyAttribute namespace="##other"
       processContents="lax"/>
11 </xs:complexType>
```

Fig. 5. Inefficiency of schema validation - code snippet from official XML Schema file for SOAP messages

- **Element: `<xs:any>`**
  Enables the document to contain elements at this position which are not defined by the schema
- **Attribute: `namespace="##any"`**
  Allows the usage of elements from any namespace(s)
- **Attribute: `namespace="##other"`**
  Allows the usage of elements from any namespace(s) except the namespace of the parent element
- **Attribute: `processContents="lax"`**
  Only validate the content if a valid schema could be obtained
- **Attribute `processContents="skip"`**
  No validation will take place on the namespace(s) referenced by the current element

C. Discussing the Weaknesses

To understand the weaknesses of the identified elements and attributes one has to look for side-effects that are attended by their usage.

<**xs:any**>. The usage of `<xs:any>` offers an attacker the option to insert *any* arbitrary elements which are not defined by the schema. This includes the option to insert wrapper code to wrap moved elements as for example `<soap:Body>`. Figure 6 shows a sample for this.

Code like the example above will ensure that existing signatures remain valid, since the original `<soap:Body>` is simply moved to a different location. The newly created `<soap:Body>` element with the id `newBodyProcessedByApplicationLogic` will be processed by the application logic. This implies that the application logic is looking for a `<soap:Body>` element at the usual position.

`namespace="##any"`. Occurrences of this attribute and value combination allow the usage of elements independently from their namespace. Any namespace is valid here. This may lead to an attack vector due to the fact that the parser can be tricked by equal sounding elements or attributes but located in another namespace. Thus schema validation can be ”bypassed” by simply crafting an own schema definition that unleashes the schema bounds.

```
<soap:Header>
  .....

  <myownnamespace:wrapper myownnamespace="http://.....">
    <soap:Body id="referencedBySignature">
    ...
    </soap>
  </myownnamespace:wrapper>
</soap:Header>
<soap:Body id="newBodyProcessedByAppLogic">

</soap:Body>
```

Fig. 6. Wrapping of SOAP bodys in wrapper elements

```
<xs:any namespace="##any" processContents="lax"
  minOccurs="0" maxOccurs="unbounded"/>
```

Fig. 7. Pitfalls with ”lax” content processing

`namespace="##other"`. In fact nearly equal to the attribute and value combination above, despite the fact that only namespaces differing from the parent namespace are allowed.

`processContents="lax"`. Lax processing announces the parser to try fetching a schema definition file and parsing the content, if the schema can not be gathered validation is skipped. If an attacker manages to block schema fetching or introducing own namespaces he gains attack surface with little to no effort.
Considering the schema definition given in Figure 7 would allow the code given in Figure 8 without violating the schema.

`processContents="skip"`. Completely disables schema validation by the parser for the child elements. The resulting security issues are obvious. Without schema validation everything enclosed by the element is legal and valid. Arbitrary code can be included without violating the schema definition, as can be seen in Figure 9.

D. Recap the Security Impact

The weaknesses described above are not theoretical only but can be found in real world scenarios. To give an example, it is not possible to detect and prohibit the construct given in Figure 10 with default schema definitions due to flimsy and untightened schema definition provided by the related standards.

The same behaviour can be achieved by replacing the `<wsse:Security>` element with an `<object>` element. Both variants are valid according to the schema definition.

```
<myownnamespace:xyz xmlns="http://unfetchable">
  ... XML Schema not validated ...
</myownnamespace:xyz>
```

Fig. 8. Tricking ”lax” content processing with unfetchable namespaces

```
<dontValidateMe>
  ... XML Schema not validated ...
</dontValidateMe>
```

Fig. 9. Skipping schema validation

```
 <soap:Header>
  ....
  <wsse:Security>
    <soap:Body id="referencedBysignature">
    ...
    </soap>
  </wsse:Security>
</soap:Header>
<soap:Body id="newBodyProcessedByAppLogic">
  ...
</soap:Body>
```

Fig. 10. Wrapping Body in Security headers

```
<element name="Object" type="ds:ObjectType"/>
<complexType name="ObjectType" mixed="true">
  <sequence minOccurs="0" maxOccurs="unbounded">
    <any namespace="##any" processContents="lax"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="MimeType" type="string" use="optional"/>
  <attribute name="Encoding" type="anyURI" use="optional"/>
</complexType>
```

Fig. 11. XML Schema of Object element from XML Signature

IV. Schema Hardening

As discussed previously, the effectiveness of XML Schema validation as a countermeasure to Signature Wrapping attacks largely depends on the XML Schema definitions. If the XML Schema used for validation is too lax, e.g. containing `<xs:any>` declarations, this opens up loopholes where an attacker may move the signed XML contents to. In fact, one such loophole in the XML Schema is sufficient to enable Signature Wrapping attacks again.

Hence, it becomes necessary to develop a definite, strict, loophole-free XML Schema of common Web Service messages. This XML Schema has to allow the same set of valid XML documents (here Web Services messages) as the original XML Schemata from the specifications themselves describe, but additionally has to be *hardened* to strictly prohibit any other content that is not contained in the XML Schema. This approach is called *Schema Hardening* [13], [12], and refers to the technique of removing all of the XML Schema extension points given in the original XML Schema files from the specifications themselves. Thus, a hardened XML Schema allows the same set of valid XML documents as the non-hardened XML Schema, but explicitly disallows all other documents.

Based on the set of specifications given in Section II-B, we created such a hardened XML Schema for a typical SOAP request message. Therefore, we subsequently removed all extension points from all of the XML Schema descriptions contained in these specifications. More precisely, we investigated every occurrence of the `<xs:any>` declaration, verifying whether it is used for common SOAP messages, and removed or restructured it appropriately. This approach is explained with two examples below.

A. Example 1: Unused Extension Points

An unused extension point is an extension point that is in fact not used by any subsequent WS-* specification involved in shaping a common SOAP message. For this example, we decided to take the XML Schema declaration of the `<ds:Object>` element from the XML Signature schema declaration given in [6]. Figure 11 shows its XML Schema declaration before the hardening process. As can be seen, a common `<ds:Object>` element is declared to contain arbitrary contents. In fact, it is commonly used for realizing *enveloping signatures*, meaning that the XML fragment signed by an XML Signature instance is contained within the XML Signature metadata block itself—located within the `<ds:Object>` element. Since XML Signature allows to sign arbitrary XML contents from arbitrary namespaces, the `<ds:Object>` element hence is declared to contain XML elements from any namespace and of any element name.

However, since WS-Security explicitly states that an XML Signature used within SOAP messages must be a *detached signature*, meaning that the signed XML fragment resides anywhere outside the XML Signature metadata block, there will never be any signed XML fragments contained in the `<ds:Object>` element of an XML Signature found in a SOAP message. Nevertheless, the XML Schema still allows the `<ds:Object>` element to be present and contain arbitrary contents—a loophole for performing an XML Signature Wrapping attack.

```
<element name="Object" type="ds:ObjectType"/>
<complexType name="ObjectType" mixed="true">
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="MimeType" type="string" use="optional"/>
  <attribute name="Encoding" type="anyURI" use="optional"/>
</complexType>
```

Fig. 12. Hardened XML Schema of Object element from XML Signature

The hardened version of this XML Schema does no longer contain the `<xs:any>` declaration, as shown in Figure 12. The declaration simply has been removed during the hardening process. Hence, an XML Signature used in the context of a SOAP message may still contain an arbitrary amount of `<ds:Object>` elements, but all of them have to be empty—otherwise the schema validation against the hardened XML Schema would fail. It is no longer possible to move a signed XML fragment to the `<ds:Object>` element without invalidating the SOAP message, hence this loophole is closed. Note that the `<ds:Object>` element thus becomes useless, and most likely would never be found in any valid SOAP message at all. However, it was allowed to be present in the non-hardened version of an XML Signature instance, therefor its presence must be tolerated as well after the hardening process. The fact that it became useless springs from the specific semantics of this element in the scope of XML Signature, which is not cared for in the plain approach of XML Schema hardening. The only result cared for here is that the extension point itself got closed.

B. Example 2: Used Extension Points

The hardening of unused extension points can rather trivially be achieved by removing the `<xs:any>` declaration, reducing

```
<xs:element name="Header" type="tns:Header"/>
<xs:complexType name="Header">
  <xs:sequence>
    <xs:any namespace="##other" minOccurs="0"
            maxOccurs="unbounded" processContents="lax"/>
  </xs:sequence>
  <xs:anyAttribute namespace="##other"
                   processContents="lax"/>
</xs:complexType>
```

Fig. 13. XML Schema of Header element from SOAP specification

the set of valid child elements to the empty set. However, this approach is not viable if an extension point is actually made use of, e.g. from within another specification. An example is shown in Figure 13, which presents the extension point schema declaration of the <soap:Header> element. Obviously, the header of a SOAP message is used to place message metadata from many other WS-* specifications in, so it naturally has to be declared using <xs:any>. However, this implies the SOAP header to become a standard loophole for performing Signature Wrapping attacks—e.g. by adding a <wrapper> element as a new child of the <soap:Header> element.

Using the same approach of XML Schema hardening as with unused extension points would imply removing the <xs:any> declaration itself. The result would be that the existence of any child element within the <soap:Header> element would immediately cause a schema violation—even for well-known and allowed header elements like <wsse:Security>. Obviously, this is not an acceptable XML Schema for real-world SOAP messages, hence the approach for resolving such used extension points in XML Schema hardening is a little more complex. It consists in determining all immediate child elements that are explicitly allowed to occur at the given extension point, and in adapting the XML Schema to explicitly allow the use of exactly those XML elements only. For the given example of the <soap:Header> element, this implies that the <wsse:Security> header and all of the WS-Addressing headers (like e.g. <wsa:ReplyTo> or <wsa:MessageID>) must explicitly be listed as allowed in the hardened XML Schema version of the extension point. Figure 14 shows this hardened XML Schema for the <soap:Header> element.

If all extension points are hardened like this, the use of the <xs:any> declaration — which always opens up a loophole for Signature Wrapping attacks — can be annihilated, rendering the hardened XML Schema resistant to this attack threat. The downside of this approach is that any additional extensions used within the SOAP messages must be explicitly embedded in the hardened XML Schema declaration of the particular Web Service. For instance, if a SOAP message is to be extended with a new kind of logging mechanism that utilizes a new SOAP header, that new SOAP header's XML Schema must be hardened itself, and inserted at the right place into the hardened XML Schema of the overall SOAP message. Otherwise, the SOAP messages containing logging headers would be rejected at the recipient's side due to their schema

```
<xs:element name="Header" type="tns:Header"/>
<xs:complexType name="Header">
  <xs:all>

    <!-- WS-Security -->
    <xs:element ref="wsse:Security" minOccurs="0"/>

    <!-- WS-Addressing -->
    <xs:element ref="wsa:MessageID" minOccurs="0"/>
    <xs:element ref="wsa:RelatesTo" minOccurs="0"/>
    <xs:element ref="wsa:To" minOccurs="0"/>
    <xs:element ref="wsa:Action" minOccurs="0"/>
    <xs:element ref="wsa:From" minOccurs="0"/>
    <xs:element ref="wsa:ReplyTo" minOccurs="0"/>
    <xs:element ref="wsa:FaultTo" minOccurs="0"/>

  </xs:all>
  <xs:anyAttribute namespace="##other"
                   processContents="lax"/>
</xs:complexType>
```

Fig. 14. Hardened XML Schema of Header element from SOAP specification

Document Processing	Processing Time [ms]
Without XML Schema	1,252
With WS-* XML Schema	4,301
With Hardened XML Schema	52,001

Fig. 15. Summary of Experimental Analysis

invalidity. However, if the set of specifications used within the context of a SOAP message exchange is known in advance or negotiated otherwise (see e.g. [17]), this approach still remains viable, and protects a Web Service effectively against the XML Signature Wrapping attack in general.

V. Real-World Example

In order to verify our approach, we validated a generated SOAP message against our hardened XML Schema. The generated SOAP message included XML Signature, WS-Addressing, Timestamp, and one element in the SOAP body. For performance validation purposes, we compared our results to the SOAP message verification with original WS-* XML Schema files. We tested our data using the Java SAX Parser. The tests ran on a machine with Intel Core 2 DUO (2,8 GHz) and 8 GB of RAM.

The summary of our evaluation can be found in Figure 15. As can be seen, the SOAP message processing without XML Schema validation is about 4 times faster than processing with a WS-* XML Schema and about 50 times faster than processing with our hardened XML Schema. The performance disadvantage by the usage of our hardened XML Schema comes most probably from the exact comparison of XML elements and their namespaces on each document layer.

The performance numbers are the strongest argument against application of the hardened XML Schema in real-world scenarios on the servers that serve many customers. On the other hand, restriction of arbitrarily defined and deeply nested elements is a good prevention against XML Signature Wrapping and Denial-of-Service attacks and therewith brings the application to a higher security level.

VI. Conclusion and Future Work

One major conclusion that can be drawn is the fact that XML Schema hardening can only be one brick towards Signature Wrapping resistant documents. With schema validation alone it cannot undoubtly be ensured that Signature Wrapping is completely avoided, but it raises the bar for potential attackers and reduces the attack surface demonstrably.

Another point to highlight is that the concept of schema hardening has to be applied to *all* included namespaces. A single too lax defined schema could result in bypassing all efforts to tighten and bound the formal definition of valid XML documents and opens up attack surface to adversaries.

As shown in our experiments, application of the hardened XML Schema brings also huge performance disadvantages. Future work will be to optimize our hardened schemata regarding performance and minimalism. A brighter testing in real world scenarios and detailed evaluation remains also work to be performed.

Acknowledgements

This work was partially funded by the `Sec2` project of the German Federal Ministry of Education and Research (BMBF, FKZ: 01BY1030).

References

[1] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security and Privacy*, vol. 8, pp. 40–47, 2010.

[2] Eucalyptus Systems, Inc., "Esa-02: Soap interfaces vulnerable to xml signature element wrapping attacks," 2011. [Online]. Available: http://open.eucalyptus.com/wiki/esa-02

[3] M. McIntosh and P. Austel, "XML signature element wrapping attacks and countermeasures," in *SWS '05: Proceedings of the 2005 workshop on Secure web services*. New York, NY, USA: ACM Press, 2005, pp. 20–27.

[4] S. Gajek, L. Liao, and J. Schwenk, "Breaking and fixing the inline approach," in *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*. New York, NY, USA: ACM, 2007, pp. 37–43.

[5] S. Gajek, M. Jensen, L. Liao, and J. Schwenk, "Analysis of signature wrapping attacks and countermeasures," in *ICWS*, 2009, pp. 575–582.

[6] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML-Signature Syntax and Processing*, W3C Recommendation, Feb. 2002. [Online]. Available: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

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

[8] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono, "All your clouds are belong to us – security analysis of cloud management interfaces," *(in submission)*, 2011.

[9] Canonical Ltd., "Usn-1137-1: Eucalyptus vulnerability," *Ubuntu Security Notice*, 2011. [Online]. Available: http://www.ubuntu.com/usn/usn-1137-1/

[10] N. Gruschka and N. Luttenberger, "Protecting Web Services from DoS Attacks by SOAP Message Validation," in *Proceedings of the IFIP TC-11 21. International Information Security Conference (SEC 2006)*, 2006.

[11] N. Gruschka, N. Luttenberger, and R. Herkenhöner, "Event-based SOAP Message Validation for WS-SecurityPolicy-enriched Web Services," in *Proceedings of the 2006 International Conference on Semantic Web & Web Services*, 2006. [Online]. Available: http://www.comsys.informatik.uni-kiel.de/publications/

[12] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on web services," *Computer Science - Research and Development (CSRD)*, vol. 24, no. 4, pp. 185–197, 2009.

[13] M. Jensen, N. Gruschka, R. Herkenhöner, and N. Luttenberger, "SOA and Web Services: New Technologies, New Standards – New Attacks," in *Proceedings of the 5th IEEE European Conference on Web Services*, 2007.

[14] M. Gudgin, M. Hadley, and T. Rogers, "Web Services Addressing 1.0 - SOAP Binding," *W3C Recommendation*, May 2006.

[15] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," *OASIS Standard*, 2006.

[16] OASIS Open, "WS-Utils Schema," http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd, 2004.

[17] M. Jensen and C. Meyer, "Expressiveness considerations of xml signatures," *35th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW)*, 2011.