Towards a Semantic of XML Signature

Sebastian Gajek, Lijun Liao, and Jörg Schwenk
Horst Görtz Institute for IT-Security
Ruhr University Bochum, Germany

{sebastian.gajek|lijun.liao|joerg.schwenk}@nds.rub.de

Abstract

McIntosh and Austel (SWS 2005, [6]) have shown that standard semantics of digital signatures in context of WS-Security fail: If parts of the document are signed and the signature verification applied to the whole document returns a Boolean value, then the document can be significantly altered without invalidating the signature. Rahaman, Schaad and Rits (SWS 2006, [8]) introduce the inline approach against the flaw. We analyze the inline approach and demonstrate weaknesses by the construction of counterexamples. Finally, we study solution ideas that mitigate XML wrapping attacks.

1 Introduction

1.1 Motivation

Security is important for any distributed computing environment: Many passive and active attacks have been described against such systems. Particularly challenging are service-oriented environments where the architecture is implemented based on a range of technologies, and where applications are created as loosely coupled and interoperable services. The Internet and its underlying infrastructure is the most pervasive IT system ever built—accordingly, more and more applications are implemented as Web services. Thus, preserving the privacy and integrity of these messages in service-oriented architectures becomes a challenging part of business integration, and secure message exchange a requirement for the proliferation of Web services.

The WS-* family of security schemes [1] aims to provide a security framework that addresses all the security issues around web services. However, this strong security framework is built on weak foundations: McIntosh and Austel [7, 6] have shown that the content of a SOAP message protected by an XML Signature as specified in WS-Security can be altered without invalidating the signature. In their model, the standard signature verification is implemented today, digital signatures are verified by a separate function, which checks the security policy related to the document and outputs a Boolean result. If this result is "true", the business logic processes the entire document (signed and unsigned parts); otherwise it is not processed. The *wrapping attack* presented by McIntosh and Austel in [7, 6] (sometimes also referred as XML rewriting attack [3, 8]) is based on the fact that unique identifier attributes are used to identify signed elements in WS-Security. The authors show that a signed element can be replaced with a faked element whereby the signature remains valid.

With regard to the presented vulnerabilities of XML Signatures, we reactivate the debate on wrapping attacks. We put into considerations measures against the attacks and outline research challenges by studying solution ideas to this problem:

- *Strict filtering.* The signature verification function acts as a filter between the input document and the business logic. It applies all transforms to the referenced elements and forwards only the byte stream resulting from this transform. This is the most radical approach, since it may result in non-XML data.

- *Returning location hints.* The signature verification function returns a hint about the location of elements referenced within XML Signature.

- *XML Signature semantics.* We discuss the requirements of a new semantic for XML Signatures, assum- ing a simple version of XPath or URI based selection (including Id attributes) is

used. Sender and receiver of a signed XML message must agree on a common semantic, which could be described as a WS-Policy construct.

The remainder sections are structured as follows: In Section 2, we review wrapping attacks and in Section 3, we sketch some novel approaches that are appealing to protect against wrapping attacks.

2 Wrapping Attacks

Wrapping attacks aim to inject a faked element into the message structure so that a valid signature covers the element while processed by the business logic. Then, a false sense of message authentication can be exploited. For instance, consider the attack where a malicious transaction service changes the amount due from "1000 $" to "2000 $" in a remittance signed by a honest customer, before it sends the remittance to a verification service. The signature covering the amount "1000 $" remains valid; however, the verification service's logic processes "2000 $". Hence, the verification service approves to pay the doubled sums.

The anatomy of wrapping attacks is best explained, using the example from [6]. Fig. 1 illustrates the structure of a SOAP message. WS-Security is used to authenticate a query for the price of some company's shares. An XML Signature protects the whole <soap:Body> element by canonicalizing, hashing and digitally signing it. The data to be protected in this way is referenced by a "wsu:Id" attribute with the value "theBody".

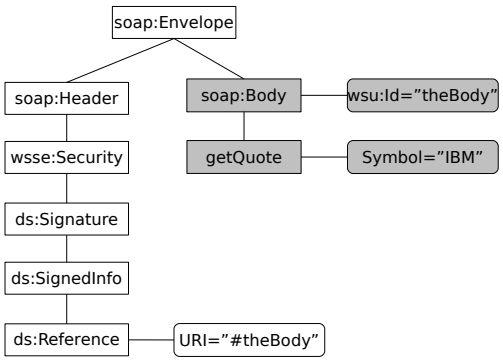

Figure 1: Example message before attack. Signed parts of the tree are colored in gray.

Upon mounting the wrapping attack, the <soap:Body wsu:Id="theBody"> element is moved to a position within the SOAP header unknown to the business logic, i.e. the logic never processes the element. In our example, this is done by embedding a new <Wrapper> element to the header and adding <soap:Body wsu:Id="theBody"> (preserving all text elements including white spaces). Additionally, a new <soap:Body wsu:Id="newBody"> element is added at the original position having correct structure, but different content. The result of this modification is depicted in Fig. 2. The Signature verification function then proceeds as follows:

1. The <Reference> element is validated to find the data against which the signature should be checked.

2. The DOM tree of the document is searched to find the element with Id="theBody" attribute. The element is located at the XPath position /soap:Envelope/soap:Header/Wrapper/soap:Body (instead of /soap:Envelope/soap:Body).

3. This element is transformed and hashed. The digest is compared with the hash value stored in the <Reference> element. Since we have preserved all subelements (including whitespaces in text elements) of <soap:Body wsu:Id="theBody">, both hash values are identical.

4. The hash value is included in the final processing of the <Signature> element and the signature verification function returns "true" to the business logic.

Since the digital signature is valid, the business logic processes the <soap:Body wsu:Id="newBody"> element. This results in answering an unauthenticated request and the attack terminates with a forged message.

The reason for the exploit is clear: Signature verification and business logic process different elements, because they use different methods to locate the elements.

3 Solution Ideas

The previous counterexample shows that we still lack a clear understanding what the *meaning* (semantic) of XML Signature is. Many XSLT transforms exist that can be applied to alter the message structure before signing. Then, content of the element to be signed can be completely changed before signing. In

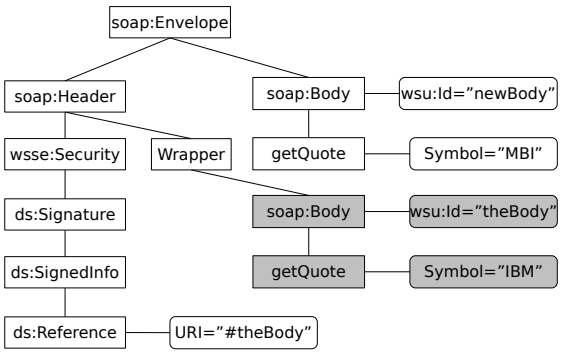

Figure 2: Example message after the attack. Parts of the tree processed by the signature verification algorithm are colored in gray. Parts of the tree processed by business logic are contained within "soap:Body".

this case, our solution proposal is to provide the business logic with the byte stream that is hashed (see Section 3.1). If element selection is permitted only (not alteration), our proposal is to communicate the business application the *location* of signed element(s). This is achieved by changing the return value of the signature verification function to provide the information (see Section 3.2).

3.1 Signature Verification as a Strict Data Filter

In this solution, the signature verification process blocks the delivery of the data to the business logic and passes through those byte streams to the business logic that are input to the digest value calculation in the <Reference> elements of the <Signature> element. This approach may result in non-XML data, if an XSLT transform outputs a set of nodes instead of a well-formed XML document, or if a base64-decoding results in binary data. It is then up to the business logic to understand this data.

On the positive side, this strict filtering approach forces Web Services' designers to avoid unnecessary transformations. On the negative side, even standard approaches (e.g., signing multiple parts of a document) result in a forest of DOM trees handed over to the business logic, instead of a single, well-formed document.

3.2 Position Hints

The basic idea is to change the return value of the signature verification function from a Boolean value to a value containing information for the business logic to locate the signed element. This approach is only sound if content of the elements is not changed (e.g. by a XSLT transformation) a subset of the document nodes is selected for signing. We distinguish two variants of this approach:

3.2.1 Solution 1: Returning a Spanning Tree

Instead of returning a Boolean value, we require the function to return the spanning tree connecting the signed elements to the root of the DOM tree.

Remark: We borrow the following notations from graph theory in order to recall the definition of a spanning tree. A graph G consists of a set $V(G)$ of objects called vertices together with a set $E(G)$ of unordered pairs of vertices called edges. If G is a graph, it is possible to choose some of the vertices and some of the edges of G in such a way that these vertices and edges again form a graph, say H. H is called a subgraph of G. A subgraph H of a graph G is called a spanning subgraph, if $V(H) = V(G)$. A tree is connected graph, if every edge is a connection of two vertices. A *spanning tree* is a spanning subgraph that is a tree when considered as a graph in its own right.

This solution does not need any security policy specification outside XML security and is appealing for client-sided implementations. Consider again the initial example as illustrated in Fig. 1 and 2. Applying the spanning tree approach to this example, the wrapping attack can be detected by the business logic. The spanning tree output of the signature verification function can be processed (Fig. 3); it contains a valid structure. The outputs of the signature verification function after the wrapping attack (Fig. 4) cannot be processed; it has a structure unknown to the business logic.

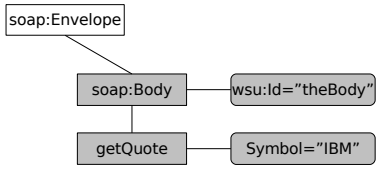

Figure 3: Spanning tree output of original Message from Fig. 1 after signature verification.

The main issue with the spanning tree approach is that the signature verification function still acts as a filter. The function removes the unsigned data except for information about the actual position of the

signed data in the document. To distinguish between signed and unsigned elements in the resulting DOM tree, the unsigned elements should be tagged by an appropriate attribute.

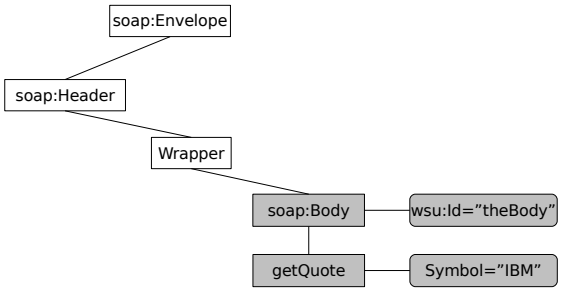

Figure 4: Spanning tree output of modified message from Fig. 2.

3.2.2 Solution 2: Returning an XPath with position information

In many settings, the business logic processes both unsigned data and signed parts. Then, the signature verification function is unable to act as a filter. To alleviate the problem, we tweak the output of the signature verification function in the following way:

- If the signature verification is successful, the signature verification function returns an XPath expression that is an absolute path in the DOM tree from the root to the signed element.
- If the signature verification fails, the value "false" is returned.

Consider again the example from Fig. 1. The verification function outputs "/soap:Envelope[1]/soap:Body[1]", whereas the modified document in Fig. 2 outputs "/soap:Envelope[1]/soap:Header[1]/Wrapper[1]/-soap:Body[1]." In order to protect the business application against wrapping attacks, a white list that consists of permitted XPath expressions can be used without sacrificing flexibility in SOAP message composition. The white list is expressed as a policy document that is tailored to certain applications.

Remark 1: Note that each element in the path is fixed by the position to avoid adding unsigned siblings with the same tag.

Remark 2: If more than one element is signed, this can be expressed by a logical "OR" combination of the respective absolute paths.

3.3 Towards a Semantic for XML Signature Elements

Wrapping attacks show that location information is an essential part of the semantic of XML Signatures. This contrasts to classical cryptographic data formats, such as OpenPGP [4] or PKCS#7 [5], where the location of signed content is implicitly known and the removal of content from this location immediately invalidates the signature.

In case of XML, wrapping attacks exploit the loosely semantics of XML Signatures. The use of a "wsu:Id" attribute to identify signed content implies a meaning like "if the hash value of the referenced data is the same as within the <DigestValue> element, then the signature is valid regardless where the data is located within the base document." If the business application *expects* the signed data at a certain location, an XML Signature format should be used whose semantic says that "the signature is only valid if it is located at or next to a certain location."

When we compare XPath (cf. [10, 9]), by contrast, the semantics either result in an ordered or unordered nodeset [2]. Previous XPath semantics have not distinguished between a subtree of the DOM tree referenced by an absolute XPath, a relative XPath, or a "wsu:Id" attribute. However, a valid semantic of XML Signature must take the location information into account as shown in [6]. Let us exemplify this concept by applying it to our running example. In Fig. 5, the signed element was referenced as recommended by a "wsu:Id" attribute. This means that the location of signed elements does not matter and that wrapping attacks do not violate the semantics of the signature.

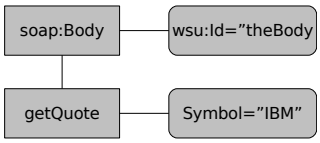

Figure 5: Semantic of the signed element when referenced via "wsu:Id".

In Fig. 6, a relative position of the signed element is given. The relative position is not altered because the the hash value includes the relative XPath—be it a transform or be it an XPointer part of the URI.

In Fig. 7, the vertical position of the signed element within the complete document is fixed. Other attacks (which may be called "horizontal wrapping attacks"), where a second path following the same pattern (i.e.,

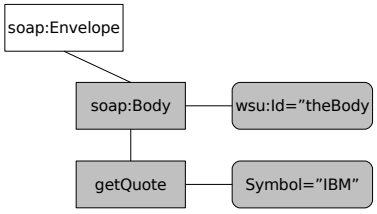

Figure 6: Semantic of the signed element when references via a relative path soap:Envelope/soap:Body.

an additional /Envelope/Body path in our example) is added. Since the additional path is included here, it is detected by the hash value calculation in the <Reference> element.

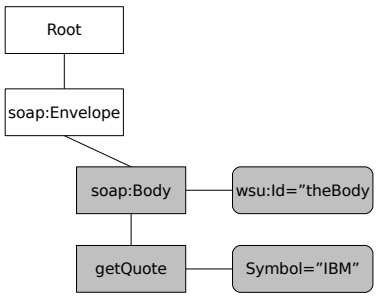

Figure 7: Semantic of the signed element when referenced via the absolute XPath /Root/-soap:Envelope/soap:Body.

References

[1] Security in a Web Services World: A Proposed Architecture and Roadmap, April 7, 2002. `http://www.ibm.com/developerworks/library/specification/ws-secmap/`.

[2] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-signature syntax and processing, Feb. 2002.

[3] K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying policy-based security for web services. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 268–277, 2004.

[4] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP message format, Nov. 1998.

[5] B. Kaliski. PKCS#7: Cryptographic message syntax standard, version 1.5, Mar. 1998.

[6] M. McIntosh and P. Austel. XML signature element wrapping attacks and countermeasures. In *Workshop on Secure Web Services*, 2005.

[7] M. McIntosh and P. Austel. XML signature element wrapping attacks and countermeasures. Technical report, IBM Research Devision, 2005.

[8] M. A. Rahaman, A. Schaad, and M. Rits. Towards secure soap message exchange in a soa. In *Workshop on Secure Web Services*, 2006.

[9] P. Wadler. A formal semantics of patterns in xslt. Technical report, Bell Labs, 2000.

[10] P. Wadler. Two semantics for xpath. Technical report, Bell Labs, 2000.