

WS-SecurityPolicy Decision and Enforcement for Web Service Firewalls

Nils Gruschka, Ralph Herkenhöner and Norbert Luttenberger

Communication Systems Research Group, Department for Computer Science
Christian-Albrechts-University in Kiel, Germany
Email: {ngr|rhk|nl}@informatik.uni-kiel.de

Abstract—A known weakness of Web Services is their vulnerability to Denial of Service attacks exploiting XML processing characteristics. To protect Web Services from these attacks, extended validation of SOAP messages—considering WS-Security and WS-SecurityPolicy—is made. For SOAP security is message oriented, the processing of the security content itself is vulnerable to Denial of Service attacks. Hence, it is necessary to combine WS-Security processing and DoS protection.

In this paper, we present our solution for WS-SecurityPolicy-based policy decision within Web Service Firewalls. For this, we give a technical description and an algorithm addressing major parts of policy decision, as well as a proposal for enhancing message signature identification. Further, we argue for advancing protection of Web Services by improved policy enforcement. This paper contributes to understanding the complexity of protecting Web Services by security gateways.

I. INTRODUCTION

Web Services implement a new paradigm of distributed communication called Service Oriented Architecture (SOA). The recommended protocol [1] for communication is SOAP [2], which is XML based and message oriented.

Web Services do not rely on transport level security like SSL but security is needed anyway. The WS-Security standard [3] defines extensions for SOAP messages to ensure *integrity* and *confidentiality* [4]. Nevertheless, problems addressing the *availability* of Web Services remain unconsidered. In addition to common Denial of Service (DoS) attacks [5], Web Services are vulnerable to DoS attacks exploiting XML processing characteristics [6]. Examples for these attacks—aiming on service availability—are Oversize Payload and Coercive Parsing [7].

In order to protect Web Services from attacks, use of WS-Security and safeguard against DoS attacks is required. For the WS-Security-enriched SOAP protocol is message oriented, evaluating WS-Security requires processing the entire message. Since processing the whole message is the main vulnerability of Web Services (and other XML-based communication systems), integrating protection from DoS attacks into the WS-Security processing is required. A solution for combining WS-Security processing and DoS protection—implemented as a firewall prototype—is presented in [8]. Fig. 1 shows the overall architecture for this firewall.

Main problems for implementing this protection solution as a firewall system are:

- efficient processing (regarding memory usage and processing time) of SOAP messages, and

- defining the set of allowed SOAP messages for filtering to exclude all malicious messages.

Efficient SOAP message processing inside a firewall system is necessary to protect the firewall itself from being attacked. Therefore, a complete event-based XML processing model [9] was used to minimize memory consumption and processing time. This includes efficiently processing XML Signature and XML Encryption tokens combined with message validation. An evaluation of the firewall prototype proofed a processing runtime linearly dependent on SOAP message sizes and a memory consumption independent of message sizes.

For each Web Service an interface definition (Web Service description [10], WSDL) is published to define the set of allowed messages. While this excludes some malicious messages, others can still pass (e.g. containing malicious content within an encrypted blocks).

A Web Service server supporting WS-Security provides—in addition to a Web Service description—a policy for describing a server’s security requirements. This specifies e.g. which SOAP message parts must be signed or encrypted. The specification for defining this policy is WS-SecurityPolicy [11]. A policy expressed with WS-SecurityPolicy (in the following only named *security policy*) can be used to filter WS-Security-enriched SOAP messages: only messages valid according to the policy are forwarded (*policy enforcement* [12]). To achieve policy enforcement, WS-Security elements inside the SOAP message must be checked to meet policy requirements (*policy decision*). WS-SecurityPolicy decision and enforcement raise a number of problems. The major problems are discussed in this paper.

The remainder of the paper is organised as follows. In the next section the related work is presented. The problems of WS-SecurityPolicy decision are discussed in section 3. The following section propose improvements for WS-SecurityPolicy enforcement. The paper closes with a summary and an outlook. In the appendix a sample security policy and a SOAP message can be found, illustrating problems regarding policy decision.

II. RELATED WORK

Most research efforts regarding WS-SecurityPolicy cover the security of the cryptographic protocol implicitly defined by the security policy. These efforts examine WS-SecurityPolicys

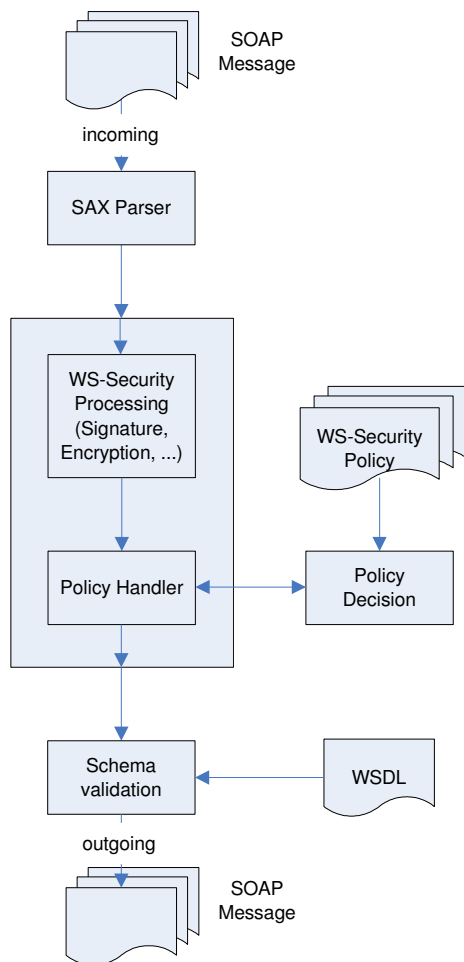


Fig. 1. Architecture for WS-Security firewall

to be vulnerable to so called *XML rewriting attacks*. These attacks base on the malicious interception, manipulation, and transmission of SOAP messages fulfilling such a policy.

Bhargavan et al. [13] present a formal threat method—in the above mentioned manner—for analyzing security policies. How to detect known weaknesses in security policy and create a tool helping to remove them is described in [14]. McIntosh and Austel [15] present a number of attacks exploiting incorrect use of signatures inside a SOAP message.

Thus, protection from these kinds of attacks is insufficient for protection of services from denial of service attacks. The requirements for protection against DoS are much stronger.

On the other hand there are commercial products—like IBM’s XML Security Gateway XS40 or Web Service Enhancements for Microsoft .NET (WSE)—supporting WS-SecurityPolicy. Most of these products use their own policy language for security modelling and can export policies using WS-SecurityPolicy. This way, the implementations do not have to bother with WS-SecurityPolicy specific characteristics for policy enforcement. At our knowledge there are no existing implementations working directly with WS-SecurityPolicy.

III. WS-SECURITYPOLICY DECISION

WS-Security [4] extends SOAP by defining security elements for ensuring confidentiality, integrity and authentication. For this purpose, it reuses XML security standards like XML Encryption [16] and XML Signature [17].

WS-SecurityPolicy [11] provides a syntax for defining a policy for SOAP message security (respective WS-Security elements). A security policy is composed from single *assertions* combined by operators `<wsp:Policy>`, `<wsp:All>` and `<wsp:ExactlyOne>`¹. WS-SecurityPolicy defines a number of assertions. The most important ones are *Protection Assertions*, *Token Assertions* and *AlgorithmSuite Assertions*.

The process of checking if a WS-Security-enriched SOAP message fulfills a security policy’s requirements, is called *policy decision*. The first step of policy decision is to map the security tokens inside a SOAP message to the corresponding assertions in a security policy. The second step is checking coherences within the security policy and other policy assertions.

The focus of this section is the major problem of the decision process, namely mapping tokens and checking connected coherences. Other details on policy decision can be found in [18].

Mapping tokens (e.g. *UsernameTokens*, *BinarySecurityTokens*) to the correct Token Assertions is discussed in the following subsection.

A. Mapping Tokens

A *Token Assertion* (inside a security policy) claims the existence of a security token (inside the SOAP message). By default Token Assertions request tokens only by type, e.g. a *UsernameToken Assertion* claims a token of type *UsernameToken* but not a specific username.

Identifying a token’s type is rather simple and part of the token processing. Mapping a token to a Token Assertion via its type highly depends on the type itself. Details needed for mapping can be gained from the respective WS-Security Token Profile. These details are out of scope of this paper and won’t be further considered. In the following we treat mapping of tokens by any type as solved.

A security policy may contain more than one Token Assertion for the same token type, especially if only one token type is used (i.e. implemented, recommended or simply most common).

Fig. 2 illustrates a common scenario of a business communication. A client is booking a flight and a car at a travel agency. The booking, the voucher, the ticket, the booking confirmation and orders are signed by the particular originator. For every message path it is shown how many tokens are involved. Usually these tokens are all of the same type, e.g. Kerberos tokens or X.509 tokens.

In cases like these tokens mapping by type are ambiguous and further criteria are needed. Therefore, it is necessary to

¹For all examples containing XML code, the “standard” namespace prefixes defined in the corresponding specifications are used.

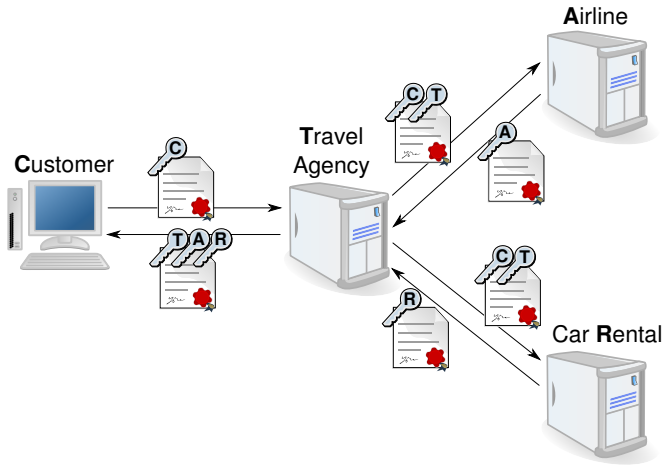


Fig. 2. Common scenario of certified business communication

consider superordinate coherences regarding Token Assertion with other Policy Assertions: namely *Protection Assertions* defining which message parts are to be signed and encrypted (e.g. `<SignedParts>` in figure 4), *AlgorithmSuite Assertions* proclaiming allowed algorithms, *SupportingToken Assertions* defining if and how tokens are signing each other, and finally *Property Assertions* like `<sp:IncludeTimestamp>` and `<sp:ProtectTokens>`.

Sample superordinate coherences are:

- Each Token Assertion is linked to all Protection Assertions and an AlgorithmSuite Assertion of the same scope. This defines which content must be signed or encrypted (by a token of that type) and which cryptographic algorithms are involved.
- WS-SecurityPolicy requests timestamp signing for message signature and recommends its signing for every supporting token. `<sp:IncludeTimestamp>` indicates the existence of a timestamp and hence this timestamp must be signed.
- SignedSupportingToken Assertions claim signing of the supporting token by the message signature.

As a conclusion, all superordinate coherences regarding Token Assertion can be deduced to three basic components: token type, reference (to signed or encrypted content), and algorithm suite. In the following we show a formal model for describing WS-SecurityPolicy superordinate coherences.

Let P be a security policy expressed in WS-SecurityPolicy, let T^P be the set of Token Assertions contained in P , let R^P be the set of references from all Protection Tokens in P and let A^P be the set of AlgorithmSuite Assertions in P . Let $Sig^P \subseteq T^P \times R^P \times A^P$ be the set of all superordinate coherences according to signing and $Enc^P \subseteq T^P \times R^P \times A^P$ the set of all superordinate coherences according to encryption. This means: if $(t, r, a) \in Sig^P$ then the policy P claims the signing of reference r by a token mapped to the Token Assertion t using algorithms conform to the requirements of the AlgorithmSuite Assertion a .

Sig^P can be built from the security policy P using the following rules:

- Within `<sp:SymmetricBinding>` or `<sp:AsymmetricBinding>` respectively pick the contained Token Assertion t for message signature and the contained AlgorithmSuite Assertion `<sp:AlgorithmSuite>` a : For each reference r inside `<sp:SignedParts>` or `<sp:SignedElements>` add (t, r, a) to Sig^P .
- Within each `<sp:SupportingTokens>`, `<sp:SignedSupportingTokens>`, `<sp:EndorsingSupportingTokens>` and `<sp:SignedEndorsingSupportingTokens>` for each Token Assertion t and the contained AlgorithmSuite Assertion `<sp:AlgorithmSuite>` a (or the AlgorithmSuite Assertion within the Binding Assertion respectively): For each reference r inside `<sp:SignedParts>` or `<sp:SignedElements>` add (t, r, a) to Sig^P .
- Within each `<sp:SignedSupportingTokens>` and `<sp:SignedEndorsingSupportingTokens>` for each Token Assertion t and the AlgorithmSuite Assertion a within the Binding Assertion: add (t_{Sig}, r_t, a) to Sig^P , with t_{Sig} Token Assertion for message signature and r_t reference to the token mapped to t .
- Within each `<sp:EndorsingSupportingTokens>` and `<sp:SignedEndorsingSupportingTokens>` for each Token Assertion t and the contained AlgorithmSuite Assertion a (or the AlgorithmSuite Assertion contained within the Binding Assertion respectively): add (t, r_{Sig}, a) to Sig^P , with r_{Sig} reference to the token mapped to Token Assertion for the message signature.
- If the [Timestamp] Property is set, for each token t : add (t, r_{TS}, a) to Sig^P , with r_{TS} reference to `<wsu:Timestamp>` and a applying AlgorithmSuite Assertion.
- If the [Token Protection] Property is set, for each token t : add (t_{Sig}, r_t, a) to Sig^P , with r_t reference to token mapped to the Token Assertion t , t_{Sig} Token Assertion for message signature and a AlgorithmSuite Assertion within the Binding Assertion.

Enc^P can be built from the security policy P using the following rules:

- Within `<sp:SymmetricBinding>` or `<sp:AsymmetricBinding>` respectively pick the contained Token Assertion t for encryption and the contained AlgorithmSuite Assertion `<sp:AlgorithmSuite>` a : For each reference r inside `<sp:EncryptedParts>` or `<sp:EncryptedElements>` add (t, r, a) to Enc^P .
- Within each `<sp:SupportingTokens>`, `<sp:SignedSupportingTokens>`, `<sp:EndorsingSupportingTokens>` and `<sp:SignedEndorsingSupportingTokens>` for each Token Assertion t and the contained AlgorithmSuite Assertion `<sp:AlgorithmSuite>` a (or the AlgorithmSuite Assertion contained within the Binding

Assertion respectively): For each reference r inside `<sp:EncryptedParts>` or `<sp:EncryptedElements>` add (t, r, a) to Enc^P .

- If the [Signature Protection] Property is set, for each Token Assertion t having a claim to sign something: add (t_{Enc}, r_{Sig_t}, a) to Enc^P , with t_{Enc} Token Assertion for encryption within the Binding Assertion, a Algorithm-Suite Assertion within the Binding Assertion and r_{sig_t} reference to the signature of the token that is mapped to the Token Assertion t .

Analogue, let M be a WS-Security-enriched SOAP message to be checked against P , let T^M be set of tokens contained in P , let R^M be the set of references to elements contained in M , let A^M be the set of algorithm groups in M —containing all necessary algorithms of a specific cryptographic operation (e.g. all transforming, hashing and signing algorithms for a given signature). Let $Sig^M \subseteq T^M \times R^M \times A^M$ be the set of all signed fragments and $Enc^M \subseteq T^M \times R^M \times A^M$ the set of all encrypted fragments (each with encrypting token and involved algorithms).

These four sets (Sig^P , Enc^P , Sig^M , Enc^M) can be used for mapping message tokens to Token Assertions. In detail these can be computed using the *Policy Token Mapping Algorithm* (PTMA):

- Check if $|T^M| = |T^P|$. Let $n := |T^M| = |T^P|$, $T^M = \{t_1^M, \dots, t_n^M\}$ and $T^P = \{t_1^P, \dots, t_n^P\}$
- Check every possible mapping of $|T^M|$ to $|T^P|$: For every permutation π of $\{1, \dots, n\}$ and for each $i \in \{1, \dots, n\}$:
 - 1) Check if token t_i^M has type claimed by token assertion $t_{\pi(i)}^P$.
 - 2) Let $Z^M = \{(r^M, a^M) \mid (t_i^M, r^M, a^M) \in Sig^M\}$ and $Z^P = \{(r^P, a^P) \mid (t_{\pi(i)}^P, r^P, a^P) \in Sig^P\}$ (the reference/algorithm pairs for the token t_i^M and the Token Assertion $t_{\pi(i)}^P$).
 - 3) Check if $|Z^M| = |Z^P|$
 - 4) Check for each $(r^M, a^M) \in Z^M$ if there exists a $(r^P, a^P) \in Z^P$ with $r^P \equiv r^M$ and $a^M \subseteq a^P$.
 - 5) Let $Z^M = \{(r^M, a^M) \mid (t_i^M, r^M, a^M) \in Enc^M\}$ and $Z^P = \{(r^P, a^P) \mid (t_{\pi(i)}^P, r^P, a^P) \in Enc^P\}$ (the reference/algorithm pairs for the token t_i^M and the Token Assertion $t_{\pi(i)}^P$).
 - 6) Check if $|Z^M| = |Z^P|$
 - 7) Check for each $(r^M, a^M) \in Z^M$ if there exists a $(r^P, a^P) \in Z^P$ with $r^P \equiv r^M$ and $a^M \subseteq a^P$.

An upper bound for the algorithm’s runtime is given by $O(|T^M||T^P|(|Sig^M| + |Enc^M|)(|Sig^P| + |Enc^P|))$. The “maximum requirement” limitation proposed in section IV implies $|Sig^P| \leq |Sig^M|$ and $|Enc^P| \leq |Enc^M|$. Additionally, $|T^M| \leq |Sig^M| + |Enc^M|$ and $|T^P| \leq |Sig^P| + |Enc^P|$. Hence, the algorithm’s runtime is $O((|Sig^P| + |Enc^P|)^4)$, which means that it is limited only by the policy and not by the message. Thus, the mapping algorithm can not be exploited by attacks using a large number of security elements.

The PTMA is used inside the firewall system presented in the introduction. As the firewall works completely event-based,

the algorithm had to be adapted:

- While reading the SOAP message header the sets Sig^M and Enc^M are created:
 - for every newly read token t^M , check if there exists a Token Assertion $t^P \in T^P$ and if $T^M \leq T^P$.
 - for every signed fragment the triple (t^M, r^M, a^M) is added to Sig^M , with t^M the token used for signing, r^M the fragment’s X-Path reference and a^M the algorithms used in the corresponding signature element.
- At the end of the SOAP message header:
 - for every $t^M \in T^M$: let O^M be the outstanding references for token t^M —i.e. the header contains a signature element using the token t^M but not the fragment signed by this signature.
 - run PTMA with the following modification: step 3): “Check if $|Z^M| + |O^M| = |Z^P|$ ”; step 4) *omitted*; step 6) “Check if $|Z^M| \leq |Z^P|$ ”; step 7) *omitted*
- While reading the SOAP message body:
 - for every signed fragment the triple (t^M, r^M, a^M) is added to Sig^M , with t^M the token used for signing, r^M the fragment’s X-Path reference and a^M the algorithms used in the corresponding signature element.
 - for every encrypted fragment the triple (t^M, r^M, a^M) is added to Enc^M , with t^M the token used for decrypting, r^M the fragment’s X-Path reference and a^M the involved algorithms.
 - for every signed or encrypted fragment run PTMA as at the end of the message header.
- At the end of the SOAP message: run PTMA (without modifications).

As superordinate coherences have been used for creating Sig^P , Enc^P , Sig^M and Enc^M , the mapping algorithm shown above simultaneously checks all superordinate coherences regarding Token Assertions. This eases the major part of the decision process.

A not yet discussed problem of policy decision is message signature identification. This is argued in the following section.

B. Message Signature Identification

The *message signature* as claimed by WS-SecurityPolicy is the central signature of a WS-Security-enriched SOAP message and is created by the message’s originator. In the section above the mapping algorithm of tokens to Token Assertions does not differentiate between supporting tokens and tokens regarding the message signature. If the claims (i.e. the algorithms and signed parts) for supporting tokens and message signature tokens resemble, this mapping is ambiguous. The following example illustrates such a situation.

In the appendix the listings 2, 3 and 4 show an example for equal claims on the token for the message signature and a supporting token. Each token must be a X.509 token of the same version, signing the SOAP body and the timestamp

with the same algorithm suite. Referring the creation rules for Sig^P they both will create equivalent sets of triples, only discriminable by the Token Assertion. If a token maps to one of these Token Assertions, it also maps to the other Token Assertion.

To solve this problem, every token $t^M \in T^M$ and respectively every Token Assertion $t^P \in T^P$ is extended by a flag indicating if it is used for message signature or not. For Tokens Assertions this can be derived from the security policy. However, the message signature token is not emphasised within a SOAP message.

Consequently, mapping tokens to Token Assertions—the major part of policy decision—depends on unambiguously identifying the message signature. In general this is not given, shown by the following example.

Listing 5 contains a SOAP message with two signatures and two X.509 tokens. Both signatures are signing the SOAP body and the timestamp using the same algorithms. It is undecidable which signature is the message signature. In both ways the message will be valid to the security policy given in listings 2, 3 and 4.

Motivated by this observation, the message signature identification is not definite, resulting in a chance for DoS attacks. A common strategy preventing attacks by non-authorized clients is to authenticate the message sender before further processing [19]. For applying this strategy to SOAP messages the message signature must be identified. Hence, additional criteria are needed to clearly identify the message signature. WS-Security only gives a weak hint on identifying the message signature². Thus, it is necessary to emphasise the message signature. For general purpose this should be claimed by the security policy. Possible solutions are:

- claiming exclusive algorithms for the message signature token,
- claiming an exclusive token type for the message signature token, or
- claiming an exclusive signed element for the message signature token.

While exclusive usage of algorithms or token types would highly restrict the usage of other tokens, extending the message signature by an exclusively signed element does not—provided that no other token ever needs to sign this element. Consequently, this element can not be a common signed element (particularly not an element inside the security header or inside the SOAP body). Additionally, it should be an element commonly used in SOAP messages to ensure interoperability. Good candidates are elements from the WS-Addressing standard [20].

As `<wsa:Action/>` is required for WS-Addressing and can only be created by the message originator, it is the best candidate to always and exclusively be signed by the message signature. This way, definite identification of the

²For signature confirmation `<wsse1:SignatureConfirmation>` is exclusively signed by the message signature, but this is optional and applies only to responses to initiator's messages.

message signature can be guaranteed by adding the following assertions within the message subject of the security policy. The additional policy assertions are shown in listing 1.

```
<sp:SignedParts>
  <sp:Header Name="Action"
    Namespace="http://www.w3.org/2005/08/addressing"/>
</sp:SignedParts>

<sp:RequiredElements>
  <sp:xPath>/Envelope/Header/Action[namespace="http://www.w3.org/2005/08/addressing"</sp:xPath>
</sp:RequiredElements>
```

Listing 1. Policy assertions for WS-Addressing signing

With message signature identification solved, the mapping algorithm is deterministic, enabling the WS-SecurityPolicy decision.

IV. IMPROVING WS-SECURITYPOLICY ENFORCEMENT

By enforcing a security policy inside a security gateway (i.e. non-policy-valid messages are rejected), Web Services can be protected from attacks using malicious messages [8]. Despite doing this, possibilities for DoS attacks still remain.

An important class of DoS attacks are called *Resource Exhaustion*. Resource Exhaustion attacks aim on consuming the resources necessary to provide the service (i.e. network bandwidth, memory and computation resources).

Attacks consuming a service's complete network bandwidth are indeed very hard to affect, usually only as *Distributed Denial of Service attack* (DDoS). For XML based protocols like Web Services, attacks consuming a service's memory are much easier (in [6] concrete attacks on .NET and AXIS servers are presented). This is a result of the processing method for XML documents. Inside a Web Service a SOAP message is converted to an internal representation—often a tree-like structure—which is much larger than the original message, for example in a DOM [21] tree representation a document scales by factor 5 to 40 (depending on XML structure). Furthermore, creating and processing this structure also produces computation costs.

The observations above demonstrate the importance of restricting a SOAP message's size for protection from DoS attacks. It is not reasonable to measure and limit the serialised message. As the in-memory size strongly depends on the message's XML structure, it is necessary to restrict XML structure itself (e.g. XML tree depth, number of elements inside sequences, size of simple type content).

For functional aspects the Web Service Description plus some heuristic modifications can be used for restricting SOAP messages. Security (non-functional) requirements of a Web Service—defined by a security policy—can be used for restricting security elements in SOAP messages.

However, WS-SecurityPolicy only defines a *minimum* set of security tokens. An attacker may add an unbounded number of additional tokens to a SOAP message, engaging the service

or—if existing—the firewall in costly cryptographic computations and forcing high memory consumption.

To avoid this, we propose to consider the requirements from the WS-SecurityPolicy document not only as a *minimum requirement*, but also as a *maximum requirement*. This means, a SOAP message must contain all required security tokens and no further tokens not specified by the security policy. Only SOAP messages fulfilling this condition are forwarded by a WS-SecurityPolicy-aware firewall, all other messages are discarded.

The question remains, if limiting the number of tokens restricts the functionality of the Web Service; this is not the case. A Web Service client adds security elements by using the server's security policy. All security needs of the server are covered by the "minimum requirements" defined inside the security policy. There is no need to add additional security elements: as the server does not expect them, they are normally not processed. On the other hand, if the the security needs of the client are not covered by the security policy, the client is supposed to abort the service invocation.

Even in a scenario with multiple targets (e.g. SOAP intermediaries) the "maximum requirement" limitation is applicable. In this case, the SOAP message has separate `<wsse:Security>` header blocks for each target [3]. Thus, a Web Service firewall only analyses the appropriate header and ignores the remaining ones. For these headers memory usage and processing time is negligible. Malicious elements within omitted `<wsse:Security>` headers are not detected at this point, but at the corresponding actor's firewall.

V. SUMMARY AND OUTLOOK

WS-SecurityPolicy enforcement can be used inside a Web Service firewall to assist in protection from Denial of Service attacks. This paper presents which problems WS-SecurityPolicy decision induces and how to modify the policy enforcement to improve DoS protection.

For policy decision a technical description and an evaluation algorithm for token mapping is given. The algorithm combines token mapping with checking token-linked coherences inside the policy. This way, the problem of ambiguous token mapping is solved. Additionally, a proposal for signing `<wsa:Action>` from WS-Addressing by message signature is presented. This solves the problem of identifying the message signature, also needed for definite token mapping.

Finally, a suggestion is made to take policy requirements not only as minimum but also as maximum constraints. This modification of policy decision improves policy enforcement regarding protection from DoS.

For now, we have achieved decision and enforcement for WS-SecurityPolicy. For further improvement of Web Service protection by a security gateway, additional requirements—which are out of the scope of WS-SecurityPolicy—must be taken into account. This includes access control, coordinated with WS-SecurityPolicy. There is also an open issue for stateful checking and integrating composed Web Service scenario, e.g. WS-BPEL [22].

REFERENCES

- [1] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C. K. Liu, M. Nottingham, and P. Yendluri, "Basic Profile Version 1.1," *WS-I Organisation*, 2004.
- [2] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," *W3C Recommendation*, 2003.
- [3] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)," 2006.
- [4] M. Hondo, N. Nagarathnam, and A. Nadalin, "Securing Web Services," *IBM Systems Journal*, vol. 41, pp. 228–241, 2002.
- [5] G. Schäfer, "Sabotageangriffe auf Kommunikationsstrukturen: Angriffstechniken und Abwehrmaßnahmen," *PIK 28*, pp. 130–139, 2005.
- [6] 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.
- [7] P. Lindstrom, "Attacking and Defending Web Service," *A Spire Research Report*, 2004.
- [8] N. Gruschka, N. Luttenberger, and R. Herkenhöner, "Event-based SOAP Message Validation for WS-SecurityPolicy-Enriched Web Services," 2006.
- [9] The SAX Project, "Simple API for XML – SAX 2.0.1," 2002.
- [10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL)," *W3C Note*, 2001.
- [11] C. Kaler and A. Nadalin (editors), "Web Services Security Policy Language (WS-SecurityPolicy)," 2005.
- [12] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser, "Terminology for Policy-Based Management, Request for Comment 3198," *Internet Engineering Task Force*, 2001.
- [13] 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*. New York, NY, USA: ACM Press, 2004, pp. 268–277.
- [14] K. Bhargavan, C. Fournet, A. D. Gordon, and G. O'Shea, "An advisor for web services security policies," in *SWS '05: Proceedings of the 2005 workshop on Secure web services*. New York, NY, USA: ACM Press, 2005, pp. 1–9.
- [15] 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.
- [16] T. Imamura, B. Dillaway, and E. Simon, "XML Encryption Syntax and Processing," *W3C Recommendation*, 2002.
- [17] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-Signature Syntax and Processing," *W3C Recommendation*, 2002.
- [18] R. Herkenhöner, "Eventbased and Policy-driven Web Service Firewall," *Diploma thesis*, 2005.
- [19] C. Meadows, "A Formal Framework and Evaluation Method for Network Denial of Service," in *PCSF: Proceedings of the 12th Computer Security Foundation Workshop*, 1999.
- [20] M. Gudgin, M. Hadley, and T. Rogers, "Web services addressing 1.0 - soap binding," *W3C Recommendation*, May 2006.
- [21] A. L. Hors, P. L. Hégaré, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne, "Document Object Model (DOM) Level 3 Core Specification," *W3C Recommendation*, 2004.
- [22] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business process execution language for web services version 1.1," May 2003.

APPENDIX

```
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:ProtectionToken>
        <wsp:Policy>
          <sp:X509Token>
            <wsp:Policy>
              <sp:WssX509V3Token10/>
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:ProtectionToken>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:TripleDesRsa15/>
          <sp>InclusiveC14N/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:Layout>
        <wsp:Policy>
          <sp:Strict/>
        </wsp:Policy>
      </sp:Layout>
      <sp:IncludeTimestamp/>
    </wsp:Policy>
  </sp:SymmetricBinding>
</wsp:Policy>
```

Listing 2. Example Endpoint Policy

```
<wsp:Policy>
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:X509Token>
        <wsp:Policy>
          <sp:WssX509V3Token10/>
        </wsp:Policy>
      </sp:X509Token>
      <sp:SignedParts>
        <sp:Body/>
      </sp:SignedParts>
    </wsp:Policy>
  </sp:SupportingTokens>
</wsp:Policy>
```

Listing 3. Example Operation Policy

```
<wsp:Policy>
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
</wsp:Policy>
```

Listing 4. Example Message Policy

```
<env:Envelope>
  <env:Header>
    <wsse:Security>
      <wsu:Timestamp Id="timestamp">
        <wsu:Created>
          2006-08-04T21:20:00Z
        </wsu:Created>
        <wsu:Expires>
          2006-08-04T21:21:00Z
        </wsu:Expires>
      </wsu:Timestamp>
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        Id="X509Token1">
        MIICKD...
```

```
</wsse:BinarySecurityToken>
<wsse:BinarySecurityToken
  ValueType="wsse:X509v3"
  Id="X509Token2">
  cNAQEF...
</wsse:BinarySecurityToken>
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod .../>
    <ds:SignatureMethod
      Algorithm="...#rsa-shal"/>
    <ds:Reference URI="#body">
      <ds:DigestMethod
        Algorithm="...#shal'"/>
      <ds:DigestValue>
        XJb0lm...
      </ds:DigestValue>
    </ds:Reference>
    <ds:Reference URI="#timestamp">
      <ds:DigestMethod
        Algorithm="...#shal"/>
      <ds:DigestValue>
        LyLsF0...
      </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    HplZkm...
  </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#X509Token1"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod .../>
    <ds:SignatureMethod
      Algorithm="...#rsa-shal"/>
    <ds:Reference URI="#body">
      <ds:DigestMethod
        Algorithm="...#shal"/>
      <ds:DigestValue>
        4bHplZ...
      </ds:DigestValue>
    </ds:Reference>
    <ds:Reference URI="#timestamp">
      <ds:DigestMethod
        Algorithm="...#shal"/>
      <ds:DigestValue>
        94hPi4...
      </ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    FZ/2kQ...
  </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#X509Token2"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</env:Header>
<env:Body wsu:Id="body">
  ...
</env:Body>
</env:Envelope>
```

Listing 5. Message with two 'non-discriminable' Signatures