Vulnerable Cloud: SOAP Message Security Validation Revisited

Nils Gruschka and Luigi Lo Iacono
NEC Laboratories Europe
Rathausallee 10
D-53757 Sankt Augustin (Germany)
{gruschka, lo_iacono}@it.neclab.eu

Abstract

The service-oriented architecture paradigm is influencing modern software systems remarkably and Web Services are a common technology to implement such systems. However, the numerous Web Service standard specifications and especially their ambiguity result in a high complexity which opens the door for security-critical mistakes.
This paper aims on raising awareness of this issue while discussing a vulnerability in Amazon's Elastic Compute Cloud (EC2) services to XML wrapping attacks, which has since been resolved as a result of our findings and disclosure. More importantly, this paper discusses the verification steps required to effectively validate an incoming SOAP request. It reviews the available work in the light of the discovered Amazon EC2 vulnerability and provides a practical guideline for achieving a robust and effective SOAP message security validation mechanism.

1. Introduction

Web Services [2] are a key technology to implement Service-Oriented Architectures (SOA) [10]. Initially used to facilitate Enterprise Application Integration (EAI), Web Services evolved—also with the emerging of Web Service Security specifications [23, 3, 4]—to a widly-deployed technology outside of organisational boundaries for cross-domain federation as well as service provisioning to external partners. The importance of Web Services is becoming more and more visible due to its steadily increasing adoption in commercial settings. One prominent example is Amazon's Elastic Compute Cloud (EC2) [1] which uses Web Services to make compute and storage infrastructure accessible as a service.

Web Services come, however, with their own threats, which need to be carefully considered and addressed. These threats include Denial of Service attacks [19] and XML Signature Element Wrapping attacks [21] (henceforth denoted as XML wrapping attacks). The latter is a general attack to XML Signature [6]. Since WS-Security makes extensive use of XML Security, XML wrapping attacks need to be carefully considered when implementing security mechanisms for Web Services.

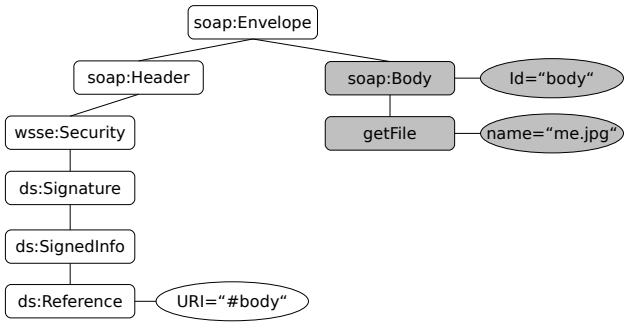

Figure 1. Example SOAP message with signed SOAP body

To build the foundation for this paper, the basics behind XML wrapping attacks are described in the following. The SOAP message shown in Figure 1 will be used to illustrate the attack. The SOAP `Body` contains an operation requesting a file resource identified by its name: 'me.jpg'. The SOAP `Body` is signed and the signature is contained in the SOAP `Security Header`.

By moving the SOAP `Body` to a newly introduced "Wrapper" element in the SOAP `Security` header, the `Body` can still be referenced by its identifier (see Figure 2). Thus, the XML Signature processing layer would still be able to address the original Body and it would still be able to hash it. Since the original Body has not been modified but just moved, the simple verification gives a valid signature result. This means, that the SOAP `Body` can now be

978-0-7695-3709-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICWS.2009.70

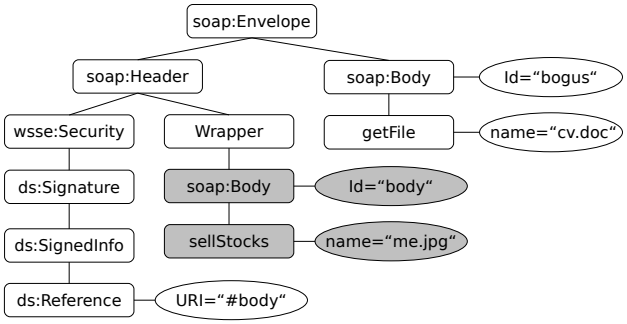

Figure 2. Example SOAP message after attack; orignal body has been moved to bogus header and new body has been inserted.

filled with a bogus `Body` containing any attacker defined operation, which is then executed after the successful signature verification. Refering back to the example given in Figure 2, the victim's CV is retrieved by the attacker.

The XML wrapping attack makes clear, that the effective SOAP message verification is far away from being straightforward. In this paper, the importance of a careful SOAP message security validation is emphasized. To raise the awareness of this issue, the first published vulnerability to XML wrapping attacks in an operational enviornment—a commercial Cloud service—is presented in Section 2. In the light of this vulnerability, in Section 3 the available work on related countermeasures is reviewed. It is shown, that in general the known countermeasures provide the required means to achieve an effective protection against this type of attack. However, to build a robust and effective SOAP message validation mechanism the simple application of the mainly theoretical work is not sufficient, since each of them focuses just on a part of the problem only. An adequate combination in form of a more concrete usage pattern is required on top of this. Section 4 suggests such an usage pattern and shows what needs to be considered or done to implement an effective chain of verification steps to authenticate a SOAP message as accurate as possible.

2. Vulnerable Cloud

To protect against the aforementioned XML wrapping attacks, careful SOAP message security validation is vital. However, it is more than obvious that this is not a straightforward task. Even carefully designed and implemented commercial environments are not necessarily well protected against these kinds of attacks. We found, for example, that the Amazon Elastic Compute Cloud (EC2) had weaknesses in the SOAP request validation component and thus allowed to take unprivileged actions in the Cloud on a victim's account. A successful attack required that the attacker is able to perform a man-in-the-middle interception of a SOAP request from a legitimated user in real-time.

Before continuing the discussion, the open and professional management of the disclosure of the vulnerability is worthwhile mentioning. Amazon understood the disclosure as a chance to improve their software—which is still in beta—while respecting the authors aim of publishing this paper as soon as possible to raise the awareness for this issue and provide guidance to ensure that service developers do not fall into the same trap repeatedly. In this respect, it has been a responsible disclosure protocol from both sides, which should really become the default approach to handle disclosures.

To understand the vulnerability, a basic understanding about Amazon's EC2 is required (see [1] for more information). After registering for Amazon Web Services (AWS) a user has two choices for using asymmetric signing mechanisms (for signing SOAP request with WS-Security); they can either (1) request that a self-signed certificate and RSA private key be generated for them for convenience, or (2) they can register a public certificate of their choice with AWS. In either case, once a user has established a certificate it is used to sign the SOAP requests originated to the EC2. In order to interact with the Cloud, Amazon provides command line tools which provide the fundamental functionality to search for virtual machine images (so-called Amazon Machine Image, AMI), to run a certain number of AMI instances, to monitor the own instances and to terminate them.

AWS maintains command line tools leveraging open-source software for SOAP messaging and SOAP message security as well as SDKs for Java, C# and PHP. In addition, developers can implement their own SOAP messaging by following the descriptions and protocols annotated in the public documentation. SOAP messages follow standard WS-Security mechanisms; specifically, this means using a `BinarySecurityToken` containing the requesting user's certificate as well as a `Timestamp` (`Created`, `Expires`) with a validity of five minutes (as replay protection and to account for clock skew) in the SOAP `Security Header`. Both the SOAP `Body` as well as the `Timestamp` in the SOAP `Security Header` must be signed when sent over an insecure transport; when sent over a secure transport, only the `Timestamp` needs to be signed. These policies are in accordance with the WS-I Security Profile Version 1.0 [22]. Figure 3 shows an example EC2 request in which the SOAP `Body` as well as the `Timestamp` is singed. The SOAP responses are not signed by AWS. Developer's wishing to authenticate the EC2 endpoint are advised to use SSL/TLS when issuing requests. The communication is SSL/TLS protected by default (meaning when using the command line tools and the SDKs).

```
<soapenv:Envelope>
  <soapenv:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken
        wsu:Id="CertId-1923154">
        MIIfnoewprsi...
      </wsse:BinarySecurityToken>
      <wsu:Timestamp
        wsu:Id="Timestamp-7461949">
        <wsu:Created>
          2008-08-27T08:30:59.201Z
        </wsu:Created>
        <wsu:Expires>
          2008-08-27T08:35:59.201Z
        </wsu:Expires>
      </wsu:Timestamp>
      <ds:Signature>
        <ds:SignedInfo>
          ...
          <ds:Reference
            URI="#id-17547166">
            ...
          </ds:Reference>
          <ds:Reference
            URI="#Timestamp-7461949">
            ...
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          ...
        </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference
              URI="#CertId-1923154"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body wsu:Id="id-17547166">
    <ec2:DescribeImages>
      <ec2:imagesSet></ec2:imagesSet>
      <ec2:ownersSet>
        <ec2:item>
          <ec2:owner>amazon</ec2:owner>
        </ec2:item>
      </ec2:ownersSet>
    </ec2:DescribeImages>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 3. Original Amazon EC2 Request

```
<soapenv:Envelope>
  <soapenv:Header>
    <!-- same header as
          original message -->
  </soapenv:Header>
  <soapenv:Body wsu:Id="id-17547166">
    <ec2:RunInstances>
      <ec2:imageId>
        ami-2bb65342
      </ec2:imageId>
      <ec2:minCount>1</ec2:minCount>
      <ec2:maxCount>100</ec2:maxCount>
      <ec2:groupSet></ec2:groupSet>
    </ec2:RunInstances>
  </soapenv:Body>
  <soapenv:Body wsu:Id="id-17547166">
    <ec2:DescribeImages>
      <ec2:imagesSet></ec2:imagesSet>
      <ec2:ownersSet>
        <ec2:item>
          <ec2:owner>amazon</ec2:owner>
        </ec2:item>
      </ec2:ownersSet>
    </ec2:DescribeImages>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4. Manipulated Amazon EC2 Request

In order to exploit the SOAP message security validation vulnerability of EC2, a signed SOAP request of a legimitate, subscribed user needs to be intercepted. Since the channel is protected by means of SSL/TLS by default, this is largely an ineffective attack vector. Nonetheless, there exists various means how to intercept such a SOAP request. In the most easy case, assume that a developer has implemented a client application from scratch without establishing an SSL/TLS secured channel. Since the EC2 Web Services allow the access via simple HTTP as well, a passive attack would be enough to get in possession of such a request. This moreover assumes that an attacker holds significant privilieges in the network. Please note already, that it does not matter what kind of request the attacker has at its disposal, since the vulnerability in the SOAP request validation allows to interfer any kind of operation and have it executed.

The SOAP message depicted in Figure 3 shows a signed request to list all the public AMIs available in EC2. The SOAP `Body` is signed and the according hash value is included in the SOAP `Security Header`. Additionally, the `Timestamp` element is signed and the `BinarySecurityToken` used to sign the request is included in the `Header` as well. If an attacker is in posses-

sion of such a request from whatever legimitate and subscribed user, the modifications to carry out the XML wrapping attack are illustrated in Figure 4.

The inclusion of the bogus Body in front of the original Body was enough to annul the SOAP verification process. The only requirement here is, that the bogus Body needs to have exactly the same Id as the original one. In the example show in Figure 4 one instance of a particular AMI is executed instead of listing all the publicly available AMIs.

This vulnerability has been discovered, while implementing own client programs to access EC2 services without any of the provided AWS tools or SDKs but just following the instructions and annotations included in the public documentation. Test runs of these own implementations revealed, that the initial SOAP message validation component of the EC2 services included already checks to mitigate the risk of XML wrapping attacks. Nonetheless, the above described variant was successful. This shows that the implementation of a effective and robust verification of SOAP requests is neither a trivial nor a straightforward task.

3. Revisiting Common Countermeasures

Signature wrapping attacks and countermeasures against them have been discussed intensively in the last years. This section presents this related work and shows how the proposed countermeasures apply to the real-world attack shown in the previous section.

One of the first works on wrapping attacks was presented by McIntosh and Austel [21]. They show a number of (hypothetical) wrapping attacks and discuss which requirements a server-side security policy must contain in order to detect the attack. These requirements are improved with every example attack circumventing the previous security policy. The final policy has the following assertions (here only those parts relevant for the Amazon example):

- 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

Obviously all these requirements are fullfilled by the attack message above. Thus, the proposed checks would not detect the attack. Additionally, the referenced article claims that the timestamp has to be referenced by an additional XPath expression. This is not fullfilled by the message shown above, but could also easily added without influencing the attack. Further, the claimed XPath references raise additional problems. First of all, the Basic Security Profile [22] does not recommend the usage of XPath references as it may imply security issues. Additionally, XPath expressions are harder to evaluate than IDs. This is especially important in the context of streaming SOAP processing [16].

Bhargavan, Fournet and Gordon also used security policies for fending wrapping attacks. They developed a formal modell for policy verification [7] and derived from these results a policy advisor [8] 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 [20]. The policy advisor proposes the following security assertions:

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

The security requirements for an incoming message to the Amazon EC2 service are not stated as a formal security policy but in a human readable form. These requirements are [1]:

- 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 fullfills the requirement from Bhargavan et al. except for the WS-Addressing [18] 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 advisor 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 signature wrapping attacks base on modifying the structure of the original message from the legitimate sender. For example in the attack message above a second SOAP body is inserted by the attacker. Therefore, Rahaman,

Schaad and Rits introduced a method – called *inline approach* – to protect some key properties of the SOAP message structure [24]. 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 header elements in inside the 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 detected. This is for example true for the sample attack on the Amazon EC2 service. The number of child elements of the SOAP envelope are changed from 2 to 3. Thus the usage of the inline approach would have detected the attack message. Neverless this protection method has some disadvantages. First, the 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. Second, this method does not generally protect from signature wrapping attacks. If an attacker is able to modify the message structure while keeping the structure properties the attack can not be detected. For example in [12] it is shown, that the inline approach can be broken.

Gajek, Liao and Schwenk [12] proposed a number of possibilities for detecting signature wrapping attacks. First, they improve the above discussed inline approach. But even for this improved version the just mentioned disadvantages remain. Additionally, the authors give some solution ideas for fending 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 components inside the Web Service framework operate truly on the message that was originally signed. Even the example Amazon EC2 attack would not succeed as the Web Service would not execute the new modified (first) body but the original (second) body. One problem of this approach – as already remarked by the authors – is that the Web Service can not 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 forrest 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 can not operate as a filter.

4. SOAP Message Security Validation

The previous sections have shown, that signature wrapping attacks are a serious threat to XML based communications, especially for Web Services. A very careful validation of incoming SOAP message is vital to provide an effective protection against the depicted attacks. Additionally, current solutions for detecting wrapping attacks are inadequate and a lot of proposed solutions are still not technically mature (e.g. not standardized) or focus on parts of the problem only.

In this section we propose a *best practice guide* for validating SOAP messages in order to fend signature wrapping attacks. The constraint for the measures is considering current Web Service specifications as well as typical workflows in Web Service frameworks. This guide picks up ideas from the above presented proposals and combines them with some new validation processes. The two main validation steps are: *XML Schema Validation* and *Security Policy Validation*.

4.1. XML Schema Validation

XML schema validation [11] is an important measure for checking the syntactical correctness of incoming message. It can help, amongst others, to reduce the risk of Denial-of-Service attacks [15] as well as discovering invalid messages in very early stages. Nevertheless, current Web Service frameworks by default do not perform a schema validation step. This is mainly because of the performance implications. Another issue in this context is the often low strictness of most schemas, which reduces the effect of schema validation. Most schemas attached to specifications include a number of extension points, i.e. types allowing arbitrary additional elements or attributes.

As an example the SOAP 1.1 specification [9] permits any element as following sibling of the SOAP body (including a second SOAP body). Thus, the first step for using XML schema validation is tightening the schema. Picking up the Amazon example, the schema for the SOAP envelope should only allow two children: the SOAP header and the SOAP body. This is also in conformance with the WS-I recommendation [5] and the SOAP 1.2 specification [17]. The attack message against the Amazon EC2 service is not valid according to such a schema. Thus, schema validation would have detected the depicted Amazon vulernability.

The schema for the SOAP body itself is well-defined by the service's description. This schema should also be as strict as possible to hamper modification inside the body

(which of course is only possible if not the whole body is signed). Instructions on how to modify the schema can be found in [14]. As this schema is controlled by the service itself, it can be easily tightened by the service developer and is automatically announced to the service client by the Web Service description.

Additionally, the schema for the SOAP header (which by default can include any element) can also be tightened to for example just include WS-Addressing and WS-Security header elements. This way, moving signed message parts to new bogus header blocks can be prevented. Of course, care must be taken when restricting the header as this reduces the flexibility of the SOAP message.

Finally, schema validation would detect the above presented attack regardless of the used schema. The claimed XML schema type for the `wsu:Id` attributes is `xsd:ID`, which does not allow two occurrences with the same value. Thus the message above is not schema valid. This property is not only useful for this special case, but also helps determining which elements are actually signed when using XPointer IDs [13]. See below for details on this topic.

4.2. Security Policy Validation

An important prerequisite for policy validation is using a security policy, that has been verified to be safe against wrapping attacks. For example, the methods and tools proposed in [8] can be used to ensure this property. As shown before, the Amazon security policy fullfills this requirement. However, obviously the process of checking the policy can still be weak.

For the security policy validation we assume the following scenario, which is very common in current Web Service development. The security component processes incoming messages for all security related tasks. The subsequent business logic component of the Web Service does not need to cope with any security issues. The security component uses a signature verification function returning the ID of the signed elements together with the certificate used to verify the signature. This is for example true for the widespread Axis WSS4J library [25].

Let P the signature assertion from the security policy given as absolute XPath expression (i.e. starting with the root node and not containing any wildcards). Other XPath expressions are not safe against wrapping attacks and thus would not have passed the policy verification claimed above. Let $P^* \subseteq P$ those assertions that refer to elements evaluated by the business logic. In simple scenarios, these are just elements inside the SOAP body. All assertions in P^* must refer to elements which are according to the message schema unique inside the message. Otherwise, either the schema or the policy must be modified. For the recommended security policy [8] the following holds:

$$P^* = \{\texttt{/soap:Envelope/soap:Body},\ \texttt{/soap:Envelope/soap:Header/wsa:To},\ \texttt{/soap:Envelope/soap:Header/wsa:Action}\}$$

Thus with the SOAP envelope schema claimed before, this requirement is fullfilled.

Let I the set IDs of signed references in the message (other referencing mechanisms then IDs are theoretically possible, but not recommended [22]). Then, the security policy check is simply: For every $p \in P$: get the ID i of the element matching p and check if $i \in I$. If this check is successfull, the certificate used for signature verification can be used for access control purposes.

The security policy validation checks that all assertions are fullfilled. Additionally, as the schema validation ensures the uniqueness of IDs, the mapping from signed elements to policy assertions is safe. And finally, as the signed elements which are evaluated by the business logic part are uniquely defined by the schema, the business logic will process exactly those elements, checked by the signature verifier.

5. Conclusion

Signature wrapping attacks are a serious threat to Web Service and SOA security. This paper has shown the—to our knowledge—first real-world occurence of such an attack in an operational service-based application. The prominent character of the vulnerable service illustrates the relevance of this topic. We investigated current solution proposals on their effect in the light of this example and concluded that they are only partly able to fend these kinds of attacks. The paper also presents a guide for detecting signature wrapping attacks, focussing on simple realization in current Web Service systems. The solution is based on validating incoming messages for conformance to the claimed schema and security policy, which have been beforehand checked for robustness regarding wrapping attacks.

Although the security considerations have shown that the proposed solution mitigates signature wrapping attacks, a formal proof of the safeness is missing. Thus, upcoming work will focus on this issue. Finally, the impact on the performance properties of the SOAP message verification components need to be investigated further.

Acknowledgement

The authors would like to thank Eric Martin, Amazon's Director of Global Security Engineering and responsible for Amazon's Global Security and Risk Management Program

for the many fruitful discussions as well as the professional and responsible handling of the disclosure process. Additionally, we want to thank Amazon Web Services engineer Mark Cavage, and Amazon Application Security Engineer, Cyrus Durgin for content review.

References

[1] Amazon Elastic Compute Cloud (EC2).

[2] G. Alonso, F. Casati, H. Konu, and V. Machiraju. *Web Services*. Springer, 2004.

[3] S. Anderson et al. Web Services Trust Language (WS-Trust). 2005.

[4] S. Anderson and others. Web Services Secure Conversation Language (WS-SecureConversation). 2005.

[5] 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.

[6] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing. *W3C Recommendation*, 2002.

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

[8] 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*, pages 1–9, New York, NY, USA, 2005. ACM Press.

[9] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. *W3C Note*, 2000.

[10] T. Erl. *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall, 2005.

[11] D. C. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. *W3C Recommendation*, 2004.

[12] S. Gajek, L. Liao, and J. Schwenk. 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, November 2007. Association for Computing Machinery.

[13] P. Grosso, E. Male, J. Marsh, and N. Walsh. Xpointer framework. *W3C Recommendation*, 2003.

[14] N. Gruschka. *Protection Web Service by extended and efficient message validation*. PhD thesis, University of Kiel, 2008.

[15] 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)*, pages 171–182, 2006.

[16] 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*, pages 80–86, 2006.

[17] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. *W3C Recommendation*, 2003.

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

[19] M. Jensen, N. Gruschka, R. Herkenhöner, and N. Luttenberger. Soa and web services: New technologies, new standards – new attacks. In *Proceeding of the 5th IEEE European Conference on Web Services (ECOWS '07)*, pages 35–44, 2007.

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

[21] M. McIntosh and P. Austel. 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, 2005. ACM Press.

[22] M. McIntosh, M. Gudgin, K. S. Morrison, and A. Barbir. Basic security profile version 1.0. *WS-I Organisation*, 2007.

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

[24] M. A. Rahaman, A. Schaad, and M. Rits. 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, 2006. ACM Press.

[25] The Apache Software Foundation. Apache wss4j.