RC23691 (W0508-064) August 9, 2005
Computer Science

IBM Research Report

XML Signature Element Wrapping Attacks and Countermeasures

Michael McIntosh, Paula Austel
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

XML Signature Element Wrapping Attacks and Countermeasures

Michael McIntosh[1,2] and Paula Austel[1,3]

[1]IBM Research
19 Skyline Drive
Hawthorne, New York, 10532
[2]mikemci@us.ibm.com
[3]pka@us.ibm.com

Abstract. Naïve use of XML Signature may result in signed documents remaining vulnerable to undetected modification by an adversary. In the typical usage of XML Signature to protect SOAP messages, an adversary may be capable of modifying valid messages in order to gain unauthorized access to protected resources. This paper describes the general vulnerability and several related exploits, and proposes appropriate countermeasures. While the attacks described herein may seem obvious to security experts once they are explained, effective countermeasures require careful security policy specification and correct implementation by signed message providers and consumers. Since these implementers are not always security experts, this paper provides the guidance necessary to prevent these attacks.

Introduction

XML Signatures[1] are designed to facilitate integrity protection and origin authentication for a variety of document types. XML Signature offers several alternatives for protecting document content. Often these alternatives appear semantically equivalent. However, closer inspection reveals subtle differences that can lead to security vulnerabilities. Proper use of XML Signatures requires a thorough understanding of the semantics associated with the alternative mechanisms.

One important property of XML Signature is that signed XML elements along with the associated signature may be copied from one document into another while retaining the ability to verify the signature. This can be useful in scenarios where multiple actors process and potentially transform a document throughout a business process. However, this same property can be exploited by an adversary allowing the undetected modification of documents. In the general case of XML, simple obvious countermeasures may effectively prevent these attacks. However, SOAP[2,3,4,5] defines a message structure and associated processing rules which, in many cases, preclude the use of these simple countermeasures.

In the following sections we describe the general XML document case, followed by SOAP specific issues. We will describe specific exploits and propose countermeasures.

Background

As members of the Web Services Interoperability Organization (WS-I) Basic Security Profile (BSP) Working Group (WG) we are occasionally called upon to provide security guidance to other WS-I WGs. On one such occasion we were asked by the Sample Applications (SA) WG to help them choose the best way to sign a specific SOAP message element using Web Services Security (WSS)[6]. The process of providing them with an answer and its justification[7,8,9] led us to explore in detail the issues described herein.

Element Context and Semantics

XML facilitates the exchange of information in a tree structure[10]. An XML document contains a single root element. Each element has a name, a set of attributes, and a value consisting of character data, and a set of child elements. The interpretation of the information conveyed in an element is derived by evaluating its name, attributes, value and position in the document. As we demonstrate below, typical WSS usage of XML Signature protects an element's name, attributes, and value without protecting its position in the document.

Context Independent Semantics

In theory, an element may have semantics associated with it that do not vary based on its position in a document. In practice we can think of no realistic examples of purely context independent semantics.

Context Dependent Semantics

In the following sections we define some categories and subcategories of element context dependent semantics. For each of these we provide one or more examples of how an adversary can exploit the position independent semantics of XML Signatures and propose suitable countermeasures. Our examples are based on SOAP-specific constructs and message processing rules. However, the issues described are not necessarily SOAP-specific.

Simple Ancestry Context

In some cases, an element is required a document at a specific position and its semantics may be completely derived from its name, attributes, and value and the name of each of its ancestors. We refer to this as Simple Ancestry Context. An example of such an element is the SOAP Body.

Example 1 is a simple SOAP Message. Note that some details have been excluded, specifically the namespace definition.

```
001 <soap:Envelope ...>
002   <soap:Body>
003      <getQuote Symbol=”IBM”/>
004   </soap:Body>
005 </soap:Envelope>
```

Example 1

Lines 001-005 contain a document root element named soap:Envelope. The syntax and semantics for the soap:Envelope are defined by SOAP. Lines 002-004 contain an element named soap:Body which is a child of the soap:Envelope element. Line 003 contains a getQuote element which is a child of the soap:Body. The syntax and semantics for children of the soap:Body are application specific.

A stock quote application receiving this message would be expected to return a message containing the price for the stock identified by the getQuote/@Symbol attribute value. The application would charge consumers for this service and would therefore need to be able to authenticate the identity of the requestor and protect the relevant message content from intentional or unintentional modification during transmission. In this case the service provider would publish a security policy describing the requirements that:

a) the soap:Body element be signed using WSS with XML Signature, and

b) the associated signature verification key be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

Example 2 contains the message protected by the sender using WSS and XML Signature. Note that some details have been excluded or abbreviated, specifically namespace definitions, URIs, and digest and signature values.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
```

```
004          <wsse:BinarySecurityToken
005             ValueType="...#X509v3"
006             EncodingType="...#Base64Binary"
007             wsu:Id="X509Token">
008             MIabcdefg0123456789...
009          </wsse:BinarySecurityToken>
010          <ds:Signature>
011             <ds:SignedInfo>
012                <ds:CanonicalizationMethod
013                   Algorithm=".../xml-exc-c14n#"/>
014                <ds:SignatureMethod
015                   Algorithm="...#rsa-sha1"/>
016                <ds:Reference URI="#theBody">
017                   <ds:Transforms>
018                      <ds:Transform
019                         Algorithm=".../xml-exc-c14n#"/>
020                   </ds:Transforms>
021                   <ds:DigestMethod
022                      Algorithm=".../xmldsig#sha1"/>
023                   <ds:DigestValue>
024                      AbCdEfG0123456789...
025                   </ds:DigestValue>
026                </ds:Reference>
027             </ds:SignedInfo>
028             <ds:SignatureValue>
029                AbCdEfG0123456789...
030             </ds:SignatureValue>
031             <ds:KeyInfo>
032                <wsse:SecurityTokenReference>
033                   <wsse:Reference URI="#X509Token"/>
034                </wsse:SecurityTokenReference>
035             </ds:KeyInfo>
036          </ds:Signature>
037       </wsse:Security>
038    </soap:Header>
039    <soap:Body wsu:Id="theBody">
040      <getQuote Symbol="IBM"/>
041    </soap:Body>
042 </soap:Envelope>
```

Example 2

Lines 039-041 contain the same soap:Body element from Example 1, with the addition of the soap:Body/wsu:Id attribute. Lines 002-038 contain a soap:Header element which is a child of the soap:Envelope element. Lines 003-037 contain a wsse:Security element which is a child of the soap:Header element. The syntax and semantics for the wsse:Security element are defined by WSS. Lines 010-036 contain a ds:Signature which is a child of the wsse:Security element. The syntax and semantics for the ds:Signature element are define by XML Signature. Lines 011-027 contain a ds:SignedInfo element which is a child of the ds:Signature element. The ds:Reference/@URI attribute contains a shorthand pointer as defined by XPointer Framework[11]. The shorthand pointer identifies the soap:Body element by the value of its wsu:Id attribute. XML parsers typically perform efficient dereferencing of shorthand pointers.

A receiver of the message above processes the included signature to verify that the signed element, the soap:Body in this case, has not been altered after it was signed. The receiver also verifies that the requestor identified by the Subject of the X.509v3 certificate is authorized to make the request.

However, since the reference uses a shorthand pointer, which is position independent, the receiver of the message must enforce the expected security policy beyond merely verifying the signature and validating the certificate.

Example 3 contains a version of the message from Example 2, as altered by an adversary attempting to gain unauthorized access. Note that some details from the previous message have been replaced by ellipses.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
009                   ...
010                </ds:Reference>
011             </ds:SignedInfo>
012             ...
013          </ds:Signature>
014       </wsse:Security>
015       <Wrapper
016          soap:mustUnderstand=”0”
017          soap:role=”.../none”>
018    <soap:Body wsu:Id="theBody">
019      <getQuote Symbol=”IBM”/>
020    </soap:Body>
021       </Wrapper>
022    </soap:Header>
023    <soap:Body wsu:Id="newBody”>
024      <getQuote Symbol=”MBI”/>
025    </soap:Body>
026 </soap:Envelope>
```

Example 3

Lines 018-020 contain the soap:Body element from Example 2, unchanged except it is now a child of the Wrapper element contained on lines 015-021 instead of the soap:Envelope element. Lines 023-025 contain a new soap:Body element which is a child of the soap:Envelope element. The new soap:Body element specifies a different getQuote/@Symbol attribute value from that in the original soap:Body.

In this example the soap:mustUnderstand attribute is used to indicate to the SOAP processing layer that the Wrapper element can be safely ignored. Also, the soap:role attribute indicates to the SOAP processing layer that the Wrapper element is not targeted at the receiver. However, even without these attributes the expected behavior is the same since the receiver is not expected to understand how to process the invented Wrapper element and the default behavior is to ignore header elements that are not understood.

Since:

a) the message contains a signature provided by an authorized requestor, and

b) the value of the element referenced by the signature is unchanged, and

c) the reference uses a position independent mechanism,

then a naïve implementation of the service may mistakenly authorize this request. This attempted exploit can be easily prevented by a properly specified and enforced security policy. Care must be taken to verify that the signed element is the soap:Body element that the application logic will process and not just any element named soap:Body. A more specific refinement of the policy specified above would be:

a) the element specified by /soap:Evelope/soap:Body must be signed using WSS with XML Signature, and

b) the associated signature verification key must be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

Optional Element Context

Proper security policy specification and enforcement can prevent attempts to move a signed element within a document when the element has a Simple Ancestry Context. However, when an element is optional within the document, the specification of an enforceable security policy that prevents its relocation requires additional considerations.

Example 4 contains a message similar to Example 2, with the addition of an included and signed optional element.

```
<soap:Envelope ...>
   <soap:Header>
      <wsse:Security>
         ...
         <ds:Signature>
            <ds:SignedInfo>
               ...
               <ds:Reference URI="#theBody">
                  ...
               </ds:Reference>
               <ds:Reference URI="#theReplyTo">
                  ...
               </ds:Reference>
            </ds:SignedInfo>
            ...
         </ds:Signature>
      </wsse:Security>
      <wsa:ReplyTo wsu:Id=”theReplyTo>
         <wsa:Address>http://good.com/</wsa:Address>
      </wsa:ReplyTo>
   </soap:Header>
   <soap:Body wsu:Id="theBody”>
     <getQuote Symbol=”IBM”/>
   </soap:Body>
</soap:Envelope>
```

Example 4

Lines 018-020 contain a wsa:ReplyTo element which is a child of the soap:Header element. The wsa:ReplyTo element specifies where the response for this request should be sent. For the purposes of our example, assume that this element is optional. If it is not present in the message the response should be sent in the HTTP Response associated with the HTTP Request that contained the SOAP request message. Lines 011-013 contain an additional ds:Reference element which refers to the new wsa:ReplyTo element.

The location where the response should be sent must be authenticated and protected from alteration. Assume the receiver side security policy is:

a) the element specified by /soap:Evelope/soap:Body must be referenced from a signature “A” using WSS with XML Signature, and

b) if present, any element matching /soap:Envelope/soap:Header/wsa:ReplyTo must be referenced from a signature “A” using WSS with XML Signature, and

c) the signature “A” verification key must be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

At first this receiver side security policy specification appears adequate. But it is not.

Example 5 contains an altered version of Example 4.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
009                   ...
010                </ds:Reference>
011                <ds:Reference URI="#theReplyTo">
012                   ...
013                </ds:Reference>
014             </ds:SignedInfo>
015             ...
016          </ds:Signature>
017       </wsse:Security>
018       <Wrapper
019          soap:mustUnderstand=″0″
020          soap:role=″.../none″>
021       <wsa:ReplyTo wsu:Id=″theReplyTo>
022          <wsa:Address>http://good.com/</wsa:Address>
023       </wsa:ReplyTo>
024       </Wrapper>
025    </soap:Header>
026    <soap:Body wsu:Id="theBody″>
```

```
027       <getQuote Symbol=”IBM”/>
028    </soap:Body>
029 </soap:Envelope>
```

Example 5

Lines 018-023 contain a Wrapper element which contains the unchanged wsa:ReplyTo element. The SOAP processing layer dispatches only the soap:Body and entire children of the soap:Header for processing. Therefore, it will not dispatch the wrapped wsa:ReplyTo and the application will behave as if the message did not have a wsa:ReplyTo header element. Since:

a) the element specified by /soap:Evelope/soap:Body is referenced from a signature “A” using WSS with XML Signature, and

b) no element matching /soap:Envelope/soap:Header/wsa:ReplyTo is present,

c) the signature “A” verification key is provided by an X.509v3 certificate issued by one of the set of trusted Certificate Authorities (CAs),

then the service may mistakenly authorize this request and return the response in the HTTP Response associated with the HTTP Request that contained the SOAP request.

The above is an example where receiver side specification and enforcement of security policy does not provide the processing expected by the sender. Some additional specification by the sender is required.

Example 6 contains a message based on Example 4.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
009                   ...
010                </ds:Reference>
011                <ds:Reference URI="">
012                   <ds:Transforms>
013                      <ds:Transform
014                         Algorithm=".../REC-xpath-19991116">
015                         <ds:XPath ...>
016                         /soap:Envelope/soap:Header/wsa:ReplyTo
017                         </ds:XPath>
018                      </ds:Transform>
019                      <ds:Transform
020                         Algorithm=".../xml-exc-c14n#"/>
021                   </ds:Transforms>
022                   ...
023                </ds:Reference>
024             </ds:SignedInfo>
025             ...
026          </ds:Signature>
027       </wsse:Security>
028       <wsa:ReplyTo wsu:Id=”theReplyTo>
029          <wsa:Address>http://good.com/</wsa:Address>
```

```
030       </wsa:ReplyTo>
031    </soap:Header>
032    <soap:Body wsu:Id="theBody”>
033      <getQuote Symbol=”IBM”/>
034    </soap:Body>
035 </soap:Envelope>
```

Example 6

Lines 011-023 contain an altered ds:Reference from Example 4. The ds:Reference/@URI no longer specifies a shorthand pointer but is now empty. An empty URI identifies the document root. Lines 013-018 contain a new ds:Transform. The ds:Transform/@Algorithm specifies the XPath[12,13] algorithm. Lines 015-017 contain a ds:XPath element which is a child of the ds:Transform element. Line 016 contains an XPath expression which specifies any element named wsa:ReplyTo which is a child of any element named soap:Header which is a child of any root element named soap:Envelope.

This position dependent or absolute path XPath expression reference can be considered a sender side specification of security policy. If the wsa:ReplyTo element were moved from its intended position after the signature was generated, during signature verification the XPath expression would resolve to an empty nodeset and the digest value would not match.

A refined receiver side security policy would be:

a) the element specified by /soap:Evelope/soap:Body must be referenced from a signature “A” using WSS with XML Signature, and

b) if present, any element matching /soap:Envelope/soap:Header/wsa:ReplyTo must be referenced via an absolute path XPath expression from a signature “A” using WSS with XML Signature, and

c) the signature “A” verification key must be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

Sibling Value Context

The simple absolute path XPath expression described above may provide suitable countermeasures against wrapping of optional elements when it is not possible for any of the element’s ancestors to have a sibling element with the same name but with different semantics.

Example 7 contains a message which is similar to Example 6. This example no longer uses the optional wsa:ReplyTo element but instead it includes an optional signed wsu:Timestamp element.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
```

```
009                   ...
010                </ds:Reference>
011                <ds:Reference URI="">
012                   <ds:Transforms>
013                      <ds:Transform
014                         Algorithm=".../REC-xpath-19991116">
015                         <ds:XPath ...>
016 /soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp
017                         </ds:XPath>
018                      </ds:Transform>
019                      ...
020                   </ds:Transforms>
021                   ...
022                </ds:Reference>
023             </ds:SignedInfo>
024             ...
025          </ds:Signature>
026          <wsu:Timestamp wsu:Id="theTimestamp">
027             <wsu:Created>2005-05-29T08:45:00Z</wsu:Created>
028             <wsu:Expires>2005-05-29T09:00:00Z</wsu:Expires>
029          </wsu:Timestamp>
030       </wsse:Security>
031    </soap:Header>
032    <soap:Body wsu:Id="theBody”>
033      <getQuote Symbol=”IBM”/>
034    </soap:Body>
035 </soap:Envelope>
```

Example 7

Lines 026-029 contain a wsu:Timestamp element which is a child of the wsse:Security element. The syntax and semantics associated with the wsu:Timestamp element are specified by WSS. Line 027 contains a wsu:Created element which is a child of the wsu:Timestamp element. The value of the wsu:Created element specifies the time the wsse:Security element was generated. Line 028 contains a wsu:Expires element which is a child of the wsu:Timestamp element. The value of the wsu:Expires element specifies the time after which the semantics associated with the other child elements contained in the wsse:Security element no longer apply. Line 016 contains an XPath expression which specifies any element named wsu:Timestamp which is a child of any element named wsse:Security which is a child of any element named soap:Header which is a child of any root element named soap:Envelope.

A appropriate receiver side security policy would be:

a) the element specified by /soap:Evelope/soap:Body must be referenced from a signature “A” using WSS with XML Signature, and

b) if present, any element matching /soap:Envelope/soap:Header/wsa:ReplyTo must be referenced via an absolute path XPath expression from a signature “A” using WSS with XML Signature, and

c) if present, any element matching /soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp must be referenced via an absolute path XPath expression from a signature “A” using WSS with XML Signature, and

d) the signature “A” verification key must be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

However, this might not provide the desired level of protection.

Example 8 contains an altered message based on Example 7.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
009                ...
010                </ds:Reference>
011                <ds:Reference URI="">
012                   <ds:Transforms>
013                      <ds:Transform
014                         Algorithm=".../REC-xpath-19991116">
015                         <ds:XPath ...>
016 /soap:Envelope/soap:Header/wsse:Security/wsu:Timestamp
017                         </ds:XPath>
018                      </ds:Transform>
019                      ...
020                   </ds:Transforms>
021                   ...
022                </ds:Reference>
023             </ds:SignedInfo>
024             ...
025          </ds:Signature>
026       </wsse:Security>
027       <wsse:Security
028          soap:mustUnderstand="0"
029          soap:role="…/none">
030          <wsu:Timestamp wsu:Id="theTimestamp">
031             <wsu:Created>2005-05-29T08:45:00Z</wsu:Created>
032             <wsu:Expires>2005-05-29T09:00:00Z</wsu:Expires>
033          </wsu:Timestamp>
034       </wsse:Security>
035    </soap:Header>
036    <soap:Body wsu:Id="theBody">
037      <getQuote Symbol="IBM"/>
038    </soap:Body>
039 </soap:Envelope>
```

Example 8

Lines 027-034 contain a wsse:Security element which is a child of the soap:Header element. Line 029 contains a wsse:Security/@soap:role attribute with a value of "none" which indicates that no SOAP node should process this header element. Lines 030-033 contain the wsu:Timestamp element from Example 7, unchanged except it is a child of the new wsse:Security element instead of the original. WSS allows for the presence of more than one wsse:Security header element in a message. WSS does have a restriction that each wsse:Security header elements must have a unique value for the wsse:Security/@soap:role attribute.

Protecting against this exploit proves very difficult. The wsse:Security header cannot itself be completely signed since intermediary SOAP nodes may need to add elements to it. Obviously some value that uniquely identifies the ambiguous ancestor is required to be used in the XPath expression, but that also must be protected from alteration.

Example 9 contains a modified form of the message contained in example 7.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security soap:role=”.../ultimateReceiver”>
004          ...
005          <ds:Signature>
006             <ds:SignedInfo>
007                ...
008                <ds:Reference URI="#theBody">
009                   ...
010                </ds:Reference>
011                <ds:Reference URI="">
012                   <ds:Transforms>
013                      <ds:Transform
014                         Algorithm=".../REC-xpath-19991116">
015                         <ds:XPath ...>
016
/soap:Envelope/soap:Header/wsse:Security[@soap:role=”.../ultimateRec
eiver”]/wsu:Timestamp
017                         </ds:XPath>
018                      </ds:Transform>
019                      ...
020                   </ds:Transforms>
021                   ...
022                </ds:Reference>
023             </ds:SignedInfo>
024             ...
025          </ds:Signature>
026          <wsu:Timestamp wsu:Id="theTimestamp">
027             <wsu:Created>2005-05-29T08:45:00Z</wsu:Created>
028             <wsu:Expires>2005-05-29T09:00:00Z</wsu:Expires>
029          </wsu:Timestamp>
030       </wsse:Security>
031    </soap:Header>
032    <soap:Body wsu:Id="theBody”>
033      <getQuote Symbol=”IBM”/>
034    </soap:Body>
035 </soap:Envelope>
```

Example 9

Line 003 contains the updated wsse:Security element which now explicitly specifies a soap:role attribute with a value indicating that the element should be processed by the ultimate receiver. Line 016 contains the updated XPath expression which specifies that the wsse:Security element must have a soap:role attribute with a value that indicates that it should be processed by the ultimate receiver.

Unfortunately this is still inadequate. WSS allows presence of one wsse:Security header element with an explicit soap:role indicating that the element should be processed by the ultimate receiver, and another wsse:Security header element without a soap:role which implicitly indicates that the element should also be processed by the ultimate receiver.

Example 10 contains a modified form of the message contained in example 9.

```
001 <soap:Envelope ...>
002    <soap:Header>
003       <wsse:Security soap:role=”.../ultimateReceiver”>
004          <wsu:Timestamp wsu:Id="theTimestamp">
005             <wsu:Created>2005-05-29T08:45:00Z</wsu:Created>
006             <wsu:Expires>2005-05-29T09:00:00Z</wsu:Expires>
007          </wsu:Timestamp>
008       </wsse:Security>
009       <wsse:Security>
010          ...
011          <ds:Signature>
012             <ds:SignedInfo>
013                ...
014                <ds:Reference URI="#theBody">
015                   ...
016                </ds:Reference>
017                <ds:Reference URI="">
018                   <ds:Transforms>
019                      <ds:Transform
020                         Algorithm=".../REC-xpath-19991116">
021                         <ds:XPath ...>
022
/soap:Envelope/soap:Header/wsse:Security[@soap:role=”.../ultimateRec
eiver”]/wsu:Timestamp
023                         </ds:XPath>
024                      </ds:Transform>
025                      ...
026                   </ds:Transforms>
027                   ...
028                </ds:Reference>
029             </ds:SignedInfo>
030             ...
031          </ds:Signature>
032       </wsse:Security>
033    </soap:Header>
034    <soap:Body wsu:Id="theBody”>
035      <getQuote Symbol=”IBM”/>
036    </soap:Body>
037 </soap:Envelope>
```

Example 10

Lines 003-008 contain the original wsse:Security header element with all of its contents removed except for the wsu:Timestamp. Lines 009-032 contain a new wsse:Security header containing all of the contents of the original except for the wsu:Timestamp. The semantics of the new header are the same as the original without the wsu:Timestamp.

Prevention of that exploit requires more specific receiver side security policy such as:

a) a signature “A” XML Signature must be present in a wsse:Security header element with an explicit soap:role attribute with the value “…/ultimateReceiver”.

b) the element specified by /soap:Evelope/soap:Body must be referenced from a signature “A”, and

c) if present, any element matching /soap:Envelope/soap:Header/wsa:ReplyTo must be referenced via an absolute path XPath expression from a signature “A”, and

d) if present, any element matching /soap:Envelope/soap:Header/wsse:Security[@role=”…/ultimateReceiver”]/wsu:Timestamp must be referenced via an absolute path XPath expression from a signature “A”, and

e) the signature “A” verification key must be provided by an X.509v3 certificate issued by one of a set of trusted Certificate Authorities (CAs).

Unfortunately, this solution does not protect against the general form of this exploit, since it depends on the semantics of the wsse:Security header. More work is required to define appropriate countermeasures that do not rely on element specific semantics.

Sibling Order Context

Another difficult problem involves the protection of individually signed sibling elements, whose semantics are related to their order relative to one another, from reordering by an adversary. More work is required to define appropriate countermeasures that do not prevent the addition and removal of siblings that do not impact the ordering semantics.

Conclusion

XML Signatures can be used to effectively protect SOAP messages only when appropriate security policies are specified and correctly enforced. Typical Web Service developers may not be aware of some subtle properties of XML Signature that can create unintended vulnerabilities. In certain circumstances shorthand pointer references do not provide adequate protection against element wrapping attacks. XPath expression references may be used effectively in many cases where shorthand pointer references are inadequate. A simple subset of XPath should be profiled in order to foster efficient, interoperable, and secure Web Services.

Acknowledgements

Portions of this work were originally described in email messages sent to Web Services Interoperability Organization (WS-I) mailing lists. A form of the attack similar to that described in Example 3 was subsequently presented by Fournet[14].

We would like to thank our colleagues at IBM especially Anthony Nadalin, Kelvin Lawrence, Hyen V. Chung, Barbara McKee, Rick Allen, Birgit Pfitzmann, Hiroshi Maruyama, Naohiko Uramoto, Suzanne McIntosh, Larry Koved, Josyula Rao, and Charles Palmer; and the members of the WS-I

Basic Security Profile (BSP) and Sample Applications Working Groups; especially Paul Cotton, Thomas DeMartini, Martin Gudgin, and Hal Lockhart; and members of the Organization for the Advancement of Structured Information Systems (OASIS) Web Services Security Technical Committee (TC), especially Chris Kaler.

References

1. Eastlake, D., Reagle, J., Solo, D. (editors): XML-Signature Syntax and Processing: W3C Recommendation: 12 February 2002 (See: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/)
2. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1: W3C Note: 08 May 2000: (See: http://www.w3.org/TR/2000/NOTE-SOAP-20000508)
3. Mitra, N. (editor): SOAP Version 1.2 Part 0: Primer: W3C Recommendation: 24 June 2003: (See: http://www.w3.org/TR/2003/REC-soap12-part0-20030624/)
4. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F (editors): SOAP Version 1.2 Part 1: Messaging Framework: W3C Recommendation 24 June 2003: (See: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/)
5. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F. (editors): SOAP Version 1.2 Part 2: Adjuncts: W3C Recommendation: 24 June 2003: (See: http://www.w3.org/TR/2003/REC-soap12-part2-20030624/)
6. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R. (editors) Web Services Security: SOAP Message Security 1.0 (WS-Security 2004): OASIS Standard 200401, March 2004 (See: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
7. McIntosh, M. "New Issue (w/Proposal): Signing the Signer's Security Token": Online Posting: 23 September 2004: WS-I Basic Security Profile Working Group (See: http://members.ws-i.org/Resource.phx/lyris/newmessage.htx?id=64333)
8. McIntosh, M. "Re: [wsi_wsbasic_apps] question from sample apps group on C5440": Online Posting: 23 September 2004: WS-I Basic Security Profile Working Group (See: http://members.ws-i.org/Resource.phx/lyris/newmessage.htx?id=64445)
9. McIntosh, M. "Re: [wsi_wsbasic_apps] question from sample apps group on C5440": Online Posting: 24 September 2004: WS-I Basic Security Profile Working Group (See: http://members.ws-i.org/Resource.phx/lyris/newmessage.htx?id=64673)
10. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F. (editors): Extensible Markup Language (XML) 1.0 (Third Edition): W3C Recommendation: 04 February 2004 (See: http://www.w3.org/TR/2004/REC-xml-20040204)
11. Grosso, P., Maler, E., Marsh, J., Walsh, N. (editors): XPointer Framework: W3C Recommendation: 25 March 2003: (See: http://www.w3.org/TR/2003/REC-xptr-framework-20030325/)
12. Boyer, J., Hughes, M., Reagle, J. (editors): XML-Signature XPath Filter 2.0: W3C Recommendation: 08 November 2002: (See: http://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/)
13. Clark, J., DeRose, S. (editors): XML Path Language (XPath) Version 1.0: W3C Recommendation 16 November 1999: (See: http://www.w3.org/TR/1999/REC-xpath-19991116)
14. Fournet, C. "Formal Tools for Web Services Security": 5-6 May 2005: DIMACS Workshop (See: http://dimacs.rutgers.edu/Workshops/Commerce/slides/fournet.ppt)