Abstract

XML signature standard as described in [RFC3275] and defined by IETF/W3C references or identifies signed elements by their unique identities in the given XML document. Hence, signed XML elements can be shifted from one location to another location in a XML document, and still, it does not have any effect on its ability to verify its signature. This flexibility paves the way for an attacker to tweek original XML message without getting noticed by the receiver. This document proposes to use absolute XPath as an "Positional Token" and modifies existing XML Digital Signature algorithm to overcome the XML Signature wrapping/rewriting attacks on XML signatures.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 13, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Introduction

Many researchers have shown that even a signed SOAP messages are vulnerable to interception and further manipulation of its content. McIntosh and Austel (described in wrapping_attack [wrapping_attack]) have illustrated that a SOAP message content, protected by an XML Digital Signature, as specified in WS-Security(refer, WS-Security [WS-Security]) can be forged without invalidating the signature. These attacks are termed as XML Signature wrapping attacks or XML rewriting attacks. These types of attacks are possible because the XML Digital Signature refers to a signed element in XML document in a way that does not take care of its location inside the XML document into consideration. Attackers inject additional nodes replacing signed nodes while still preserving the signed nodes inside the document but at different level in the hierarchy of the XML tree such that it results in successful signature verification thereby resulting in XML Re-Writing/Wrapping attack.
1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. XML Digital Signature structure

XML Signatures (described in RFC3275 [RFC3275]) are applied to arbitrary digital content (data objects). Data objects are digested, the resulting value is placed in an element (with other information) and that element is then digested and cryptographically signed. XML digital signatures are represented by the Signature element which has the following structure (where "?" denotes zero or one occurrence; "+" denotes one or more occurrences; and "*" denotes zero or more occurrences):

```xml
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
     (<Transforms>)?
     <DigestMethod>
     <DigestValue>
     </Reference>)+
    </Reference>
  </SignedInfo>
  <SignatureValue>
   (<KeyInfo>)?
   (<Object ID?>)*
  </Signature>
```

Signatures are related to data objects via URIs [URI]. Within an XML document, signatures are related to local data objects via fragment identifiers.

3. Suggested Modified Algorithm

As, SOAP requests are prone to XML wrapping attacks and this vulnerability stems mostly because of usage of ID (Identity) to identify the signed XML sub tree. There are many solutions proposed to mitigate such attacks but still such attacks can’t be fully eliminated because of inherent limitation present in XML Digital Signature standard. In this document, we have proposed an addition of "Positional Token" as a doping to the XML element getting signed to mitigate XML Signature wrapping attacks. We are also proposing a little modification of existing XML Signature standard as to use of
"Absolute XPath" instead of ID in <Reference> node’s "URI" attribute to refer the signed element. Use this absolute XPath as a "Positional Token", as this token exactly points to the position of element getting signed. Also, during signing process, add this "Positional Token" as an attribute (e.g. PosToken= "Absolute XPath") to the element subjected to be signed. This absolute XPath as a "Positional Token" would identify the signed element in XML Signature and addition of this "Positional Token" as an attribute to the element getting signed eliminate the chances of XML Wrapping attacks as in the case of forged SOAP requests, calculated digest of signed element will not match with the respective digest value in <DigestValue> node during signature validation process. We propose a modified XML signature algorithm which suggests usage of absolute XPath as a "Positional Token" and it will be used during signing as well as during signature validation process. The algorithms are as follows:

3.1. Algorithm for signing SOAP Request

1. KS=Load(Keystore.JKS) //Load certificates and keys
2. For each element subjected to be signed(represented by its "id" attribute value) {
3. ABSXpath= "Absolute XPath" of element to be signed as identified with its "Id" attribute value
4. ProtectTree=Node as identified by ABSXpath
5. MixedElement=AppendSyntacticToken(ProtectTree,  ABSXpath) /*Append a Positional Token as an attribute, "PosToken= ABSXpath" to the ProtectTree */
6. H=Hash(MixedElement)
7. Add ABSXpath to <Reference> node as "URI" attribute value
8. Enclose H to <DigestValue> node inside the <Reference> node, as defined in XML Signature standard.
9. }
10. SignedInfoHash=calculate hash of <SignedInfo> element /* Calculate the digest of the <SignedInfo> element */
11. SignedSOAP=Encrypt(SignedInfoHash , KS.PrivateKey) /*Signing that digest and enclosing the signature value in a <SignatureValue> element */

3.2. Algorithm for verification of Signature
1. SignInfoDigest=Calculate digest of the <SignedInfo> element
2. SignatureValueContent= content inside <SignatureValue> node
3. Flag=VerifySignature(Public Key, SignatureValueContent, SignInfoDigest)
4. If(Flag){
5.  Ids=All URI’s in <Reference> nodes inside the <SignedInfo> node
6.  For each Id from Ids{
7.    ABSXpath=Get the content of Id
8.    Subtree=Get the sub tree identified by ABSXpath
9.    MixedElement=AppendSyntacticTokenSubTree(Subtree, ABSXpath)
   /* Append a Positional Token as an attribute, PosToken= ABSXpath */
10.   H=Hash (MixedElement) /* generate hash value of signed elements. */
11.   Digest=Get digest value under the <Reference> node and inside <DigestValue> node, whose "URI" is equal to Id
12.   If(H!=Digest){
13.      return "Signature Validation Failed"
14.   }else{
15.      return "Signature Validation Successful"
16.   } //For loop
17. } else
18. return "Signature Validation Failed"
19. }
20. 

3.2.1. Verifying SignedInfo Element Digest with Decrypted Digest from SignatureValue element

1. VerifySignature(PublicKey, SignatureValueContent, SignInfoDigest){
2.  DecryptedDigest=Decrypt SignatureValueContent with PublicKey
3.  If(DecryptedDigest!=SignInfoDigest){
4.      return False
5.   } else{
6.      return True
7.   } //For loop
8. } else
9. }

4. Simple Example
The <Signature> Lets consider an XML document for the example purpose:

```xml
<?xml version="1.0"?>
<PatientRecord>
    <Visit date="10pm March 2018">
        <Account id="id1">1234</Account>
        <Name>ABC</Name>
        <Diagnosis>Kidney Test</Diagnosis>
    </Visit>
    <Visit date="12pm May 2018">
        <Account id="id2">1235</Account>
        <Name>DEF</Name>
        <Diagnosis>Liver Test</Diagnosis>
    </Visit>
</PatientRecord>
```

Figure 1
Existing XML Signature algorithm would produce a `<Signature>` element for the XML document mentioned in Figure 1, as follows:

```xml
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#WithComments" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="#id1">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>.................</DigestValue>
    </Reference>
    <Reference URI="#id2">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>................</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    ............
  </SignatureValue>
  <KeyInfo>
    <X509Data>
      <X509Certificate>
        ............................
      </X509Certificate>
    </X509Data>
  </KeyInfo>
</Signature>
```
The proposed XML Signature algorithm would produce a `<Signature>` element for the XML document mentioned in Figure 1, which is described in Figure 2. Also, during signing process, "Positional Token" as an attribute e.g. (PosToken= "Absolute XPath") has been used in it, as per proposed algorithm in Section 3.1. Now, `<DigestValue>` elements inside `<Signature>` element will also contain the trace of "Positional Token", hence the relative position of signed elements in the given XML document:

```xml
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#WithComments" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference URI="/PatientRecord/Visit[1]/Account[@id='id1']">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>.................</DigestValue>
    </Reference>
    <Reference URI="/PatientRecord/Visit[2]/Account[@id='id2']">
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>................</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    ............
  </SignatureValue>
  <KeyInfo>
    <X509Data>
      <X509Certificate>
        ..................................
      </X509Certificate>
    </X509Data>
  </KeyInfo>
</Signature>
```

Figure 2
5. Algorithm Validation

In this section we will discuss as how the suggested algorithm can mitigate the various scenarios of XML wrapping attacks.

5.1. Mitigation of XML Signature wrapping attacks

This kind of attacks are possible because signature verification algorithm identifies signed element using identity i.e. ID and identifying position of signed element using ID has inherent flaw as the signed element can easily be moved within the document and still the document retains the ability to verify its signature. So, in our algorithm, we have suggested the usage of absolute XPath in place of ID for identifying the position of signed elements. Absolute XPath has two fold advantages as it can easily identify the position of signed element within the XML document and it fixes both the vertical and horizontal axes of the signed element exactly. The absolute XPath expression to identify signed element will not be same as absolute XPath expression for signed element in forged document. The signature validation will fail at step-8, of algorithm in Section 3.2, as there is no such node. Further, if the attacker modifies the URI attribute and tries to perform XML wrapping attack, the digest of <SignedInfo> will not match and signature validation will fail at step-4 of algorithm in Section 3.2.

5.2. Mitigation of XML elements jumbling type of wrapping attacks

This kind of wrapping attacks are possible as the attacker jumbles the position of signed elements within the document as XML signature process defined by specification takes only ID into consideration for referencing the signed elements. The proposed algorithm suggests using "Absolute XPath" for referencing the signed elements as well as doping the elements subjected to be signed with it. Hence, the digest of the signed element inside <DigestValue> node has a trail of the position of element; refer step-6 of algorithm in Section 3.1. Hence, any changes in the position of signed elements by the attackers will invalidate the signature validation; refer step-12 of algorithm in Section 3.2, because calculated digest during signature validation will not match with the digest contained in <DigestValue> of the forged SOAP request.

6. Conclusion

XML Signature wrapping attacks try to inject forged elements into the XML document structure in such a way that the valid signature covers the unmodified elements, while forged elements are processed by the application logic. This results in a scenario, where an attacker can perform arbitrary web service requests, while authenticating as a
legitimate user. The proposed algorithm takes help of the absolute XPath as a "Positional Token" for identifying the signed elements and adding this to the elements to be signed as an attribute before the canonicalization process has a trace of both content of signed element and its position in the XML document as well. Hence, the proposed algorithm can solve the issue of XML wrapping attacks elegantly without much change in the current standard.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

This draft proposes a modification to the existing algorithm of XML signature to counter XML Signature wrapping attacks. However other forms of attack may be possible that could not be mitigated.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

Jitendra Kumar
C-DAC Bangalore
#68, Electronics City Hosur Road
Bangalore 560100
India

Email: jitendra@cdac.in

Balaji Rajendran
C-DAC Bangalore
#68, Electronics City Hosur Road
Bangalore 560100
India

Email: balaji@cdac.in

Bindhumadhava BS
C-DAC Bangalore
Old Madras Road, Opposite Hal Aero Engine Division
Bangalore 560038
India

Email: bindhu@cdac.in