A .NET client cannot consume a SOAP message bound to a SAML 2.0 identity assertion signed by reference using the STR-TRANSFORM algorithm

Asked 10 years, 6 months ago Modified 10 years, 1 month ago Viewed 2k times

4

Post updated with a working example. Please see WORKING EXAMPLE below.

Problem - A .NET client cannot consume a SOAP message bound to a SAML 2.0 identity assertion signed by reference using the STR-TRANSFORM algorithm.

Java message producer: Spring and WSS4J

.NET client consumer: version 4.5.1

SAML: version 2.0, Sender-Vouches confirmation method; assertion itself signed; assertion also signed at the message level by reference using the STR-TRANSFORM algorithm.

The .NET client is failing on this:

```
<ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#STR-Transform">
```

A Microsoft hotfix (http://support.microsoft.com/kb/974842) for the .NET 3.5 framework allows a .NET message producer to a SOAP message bound to a SAML assertion signed by reference by defining a custom binding in web.config. But we have been unable to figure out how to get a .NET client to consume such a message.

EDIT:

Thank you for finding a better forum for this post. I understand the concern about multiple questions. Let me retry. The overall goal is to get the .NET client to consume a SOAP message bound to an identity in the form of a SAML assertion signed at the message level by reference using the STR-TRANSFORM algorithm, which resolves the reference to the actual assertion to allow signature verification. The .NET client is not handling the transform.

The .NET client throws this error:

```
An error occurred: 'System.Security.Cryptography.CryptographicException:
Unknown transform has been encountered.
   at System.Security.Cryptography.Xml.Reference.LoadXml(XmlElement value)
```

```
   at System.Security.Cryptography.Xml.SignedInfo.LoadXml(XmlElement value)
   at System.Security.Cryptography.Xml.Signature.LoadXml(XmlElement value)
   at System.Security.Cryptography.Xml.SignedXml.LoadXml(XmlElement value)
```

I think this error means that the program has encountered a URI in the message's element that is not recognized by the .NET framework. I have verified, by substituting a different transform in the message, that .NET does not recognize this transform:

```
<ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#STR-Transform">
```

Section 3.4.3 (SAML Assertion Referenced from SignedInfo) of the OASIS Web Services Security SAML Token Profile Version 1.1.1 states that "all conformant implementations MUST be able to process SAML assertions referenced by from elements within the element of a element in a header."

The .NET framework appears not to conform to specification. To customize WSS4J OpenSAML to use a different transform when producing the message would only worsen conformance.

Has anyone registered a custom .NET class in machine.config to handle the STR-TRANSFORM algorithm?

UPDATE, after troubleshooting the .NET side:

To produce the exception above ("Unknown transform has been encountered"), System.Security.Cryptography.Xml.Reference.LoadXML tries this:

```
Transform transform = CryptoConfig.CreateFromName(attribute) as Transform;
```

And fails here:

> if (transform == null) { throw new CryptographicException(SecurityResources.GetResourceString("Cryptography_Xml_UnknownTransform")); }

System.Security.Cryptography.CryptoConfig.CreateFromName goes to machine.config to determine what algorithms are available to the .NET framework.

Would it be appropriate to define a custom class to handle the STR-Transform algorithm then reference the class in machine.config, something like this?

```
<mscorlib>
    <cryptographySettings>
```

```
      <cryptoNameMapping>
        <cryptoClasses>
          <cryptoClass
strtransform="Custom.Class.StrTransformProvider,Custom.Class" />
        </cryptoClasses>
        <nameEntry name="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#STR-Transform" class="strtransform" />
      </cryptoNameMapping>
    </cryptographySettings>
</mscorlib>
```

WORKING EXAMPLE

I have updated this post with a working example. This took some time to achieve because I am on the side of the Java message producer, not the .NET message consumer, where the error occurs. The example payload is based on a WSS4J framework with OpenSAML libraries. The example program attempts to be a simple C# signature validator. Please note that cryptographic verification of signatures is not the point of the example and that successfully verifying signatures would actually be cryptographically impossible on the example payload.

To work the example, you will need the C# code and the example payload. (Due to length constraints, I will post the payload in a separate update.) If you save the payload as C:\Temp\Payload.xml, you will not need to modify the C# program. For my test I used Visual Studio 2010 with .NET 4.0 as the target framework. You might need to add some references to your Visual Studio project. I think that you should also be able to produce the same error using the latest .NET framework.

The C# code fails on this line:

```
signedXml.LoadXml((XmlElement)node);
```

With this message:

```
A first chance exception of type
'System.Security.Cryptography.CryptographicException' occurred in
System.Security.dll
********* ERROR: System.Security.Cryptography.CryptographicException: Unknown
transform has been encountered.
   at System.Security.Cryptography.Xml.Reference.LoadXml(XmlElement value)
   at System.Security.Cryptography.Xml.SignedInfo.LoadXml(XmlElement value)
   at System.Security.Cryptography.Xml.Signature.LoadXml(XmlElement value)
   at System.Security.Cryptography.Xml.SignedXml.LoadXml(XmlElement value)
   at TestSignatureVerification.Program.ValidateDocument(XmlDocument docToTest)
in ... Program.cs:line 59
   at TestSignatureVerification.Program.VerifyXMLSignature(String
xmlFileLocation) in ... Program.cs:line 26 *********
```

Please note that the signatures in this example are bogus. There would be no way to validate them even if you could get past the error. Cryptographic verification of signatures is downstream from the error and not relevant to this issue. Also note that the XOP-referenced binary attachment is not included in the example payload and is not relevant to it.

The test message's wsse:Security header has the following child elements relevant to this issue:

- wsse:BinarySecurityToken (x509 certificate of the SOAP producer)
- saml2:Assertion (identifies the creator of the transaction; the SOAP producer vouches for this identity using the SAML Sender-Vouches confirmation method)
- wsse:SecurityTokenReference (references wsse:BinarySecurityToken and is referenced by ds:Signature - see next)
- ds:Signature (signs message body and SAML assertion)

The SAML assertion is signed by reference, at the message level, using this algorithm:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform

In order to validate the signature, the SAML assertion first has to be resolved from the reference using the STR-Transform algorithm. I believe that this is where the .NET code fails with the "Unknown transform has been encountered" error. I base this conclusion on the fact that if you substitute "http://www.w3.org/2001/10/xml-exc-c14n#" for "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform" in the message you will bypass the "Unknown transform has been encountered" error and the program will simply (and as expected, given the bogus signatures and fictitious identities in the message) fail to validate the signatures (signature validation is not the point of the example; handling a transform algorithm is).

Note that the saml2:Assertion element itself has a ds:Signature element. The Security Token Service that provides the assertion signs the assertion. The SOAP producer takes the assertion as a message input from a Security Token Service. Before including the assertion in the message, however, The SOAP producer needs to verify the assertion signature to prove to itself that the assertion was not modified in the hand-off from the Security Token Service and to confirm its trust relationship with the Security Token Service. Having verified the integrity and producer of the SAML assertion, the SOAP producer signs the assertion at the message level, vouching for the assertion to the message consumer. This second signature, at the message level, is not a duplicate signature, not only because the signer is not the same, but also

because the security context is different (one context is the Security Token Service; the other is the SOAP producer).

In this example, Google is the fictitious Security Token Service. The value of the ds:X509Certificate element in the SAML assertion is the www.google.com digital certificate. Google (fictitiously) signs the SAML assertion, gives the signed assertion to the SOAP producer, which verifies the signature. The the SOAP producer, identified in this example by the digital certificate from https://www.example.com, fictitiously signs the message body as well as the SAML assertion.

The XPath query in this example picks up all ds:Signature elements in the message. The signature on the SAML assertion happens to source from the second ds:Signature element in the message. The program does not break on the cryptographic failure of the first signature verification (the signature at the SOAP message level) because "if (!status) break" is remarked. Then the program tries to validate the signature on the SAML assertion. It fails with the "Unknown transform has been encountered" error. This happens before the attempt to verify the signature; the program never gets to that step on the second signature. The staleness of all the conditions in the SAML assertion is irrelevant to the example.

Example C# Code

```
using System;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Security.Cryptography.Xml;
using System.Xml;
using System.Collections.Generic;
using System.Diagnostics;

namespace TestSignatureVerification
{
    class Program
    {
        static void Main(string[] args)
        {

Console.WriteLine(VerifyXMLSignature(@"C:\Temp\Payload.xml").ToString());
        }

        public static bool VerifyXMLSignature(string xmlFileLocation)
        {
            try
            {
                XmlDocument docToTest = new XmlDocument();
                docToTest.PreserveWhitespace = true;
                docToTest.XmlResolver = null;
                docToTest.Load(xmlFileLocation);
                return ValidateDocument(docToTest);

            }
            catch (Exception e)
            {
```

```
                // Console.WriteLine(e.Message);
                 Debug.WriteLine("********* ERROR: " + e.ToString() + "
*********");
                // Debug.WriteLine(e.StackTrace);
                 return false;
            }
        }

        public static bool ValidateDocument(XmlDocument docToTest)
        {
            bool status = true;

            XmlNamespaceManager manager = new
XmlNamespaceManager(docToTest.NameTable);
            manager.AddNamespace("wsse", "http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd");

            XmlNodeList securityList = docToTest.SelectNodes("//wsse:Security",
manager);
            X509Certificate2 cert = getCertificate(securityList[0]);

            // http://www.w3.org/2000/09/xmldsig#
            manager.AddNamespace("ds", SignedXml.XmlDsigNamespaceUrl);
            XmlNodeList nodeList = docToTest.SelectNodes("//ds:Signature",
manager);

            Debug.WriteLine("Count of Signature nodes: " + nodeList.Count);

            SignedXml signedXml = new SignedXml(docToTest);

                foreach (XmlNode node in nodeList)
                {
                Debug.WriteLine("InnerXML: " + node.InnerXml);
                signedXml.LoadXml((XmlElement)node);
//                Debug.WriteLine("Certificate: " + cert);
                status = signedXml.CheckSignature(cert, true);
//                Debug.WriteLine("Node Name: " + node.Name);
                Debug.WriteLine("CheckSignature status: " + status);
//                if (!status)
//                    break;
            }
            return status;
        }


        private static XmlElement retrieveHeader(XmlDocument xmlContent)
        {
            return xmlContent.ChildNodes.OfType<XmlElement>().First(e =>
e.Name.Contains("Envelope")).ChildNodes.OfType<XmlElement>().First(e=>
e.Name.Contains("Header"));

        }

        private static X509Certificate2 getCertificate(XmlNode securityNode)
        {
            XmlElement binarySecurityToken = (
               from element in securityNode.ChildNodes.OfType<XmlElement>()
               where element.Name.Contains("BinarySecurityToken")
               select element).First();
            string encodedCertificate = binarySecurityToken.InnerText;
            byte[] decodedContent =
Convert.FromBase64String(encodedCertificate);
```

```
            return new X509Certificate2(decodedContent);
        }


    }
}
```

Example Payload

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" SOAP-ENV:mustUnderstand="1">
            <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3" wsu:Id="CertId-
81591DAC97D1A4EF26139995608718319">MIIROTCCECGgAwIBAgIQD2AtUFHRJ/PkEI4BTHIMwDANBgk
            <saml2:Assertion
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
ID="_81591DAC97D1A4EF26139995608705916" IssueInstant="2014-05-13T04:41:27.065Z"
Version="2.0" xsi:type="saml2:AssertionType">
                <saml2:Issuer>
www.example.com</saml2:Issuer>
                <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:SignedInfo>
                        <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                        <ds:Reference
URI="#_81591DAC97D1A4EF26139995608705916">
                            <ds:Transforms>
                                <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                                <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                            </ds:Transforms>
                            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                            <ds:DigestValue>Q8nxma/rf1XRfxq46oR7vaj/1yA=
</ds:DigestValue>
                        </ds:Reference>
                    </ds:SignedInfo>

<ds:SignatureValue>CzkNUiZppovAIY/atOQzRQfirJ8yFcwbTnwSz8tKcJgx5nYMP23jRZ855lo20la
</ds:SignatureValue>
                    <ds:KeyInfo>
                        <ds:X509Data>

<ds:X509Certificate>MIIEdjCCA16gAwIBAgIILBjgSyHeH78wDQYJKoZIhvcNAQEFBQAwSTELMAkGA1
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
cm5ldCBBdXRob3JpdHkgRzIwHhcNMTQwNzE2MTIxNDExWhcNMTQxMDE0MDAwMDAw
WjBoMQswCQYDVQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEWMBQGA1UEBwwN
TW91bnRhaW4gVmlldzETMBEGA1UECgwKR29vZ2xlIEluYzEXMBUGA1UEAwwOd3d3
```

```
Lmdvb2dsZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDKMwph
144V2I3T0MfpUCYOgRASpo9gOTP/jVfkart6L2Y5OkJRWLPf2kjFHu9lneFMUkYU
SF/FM62prDo256Hw19/ec8GqfeeWgdn/I7F1GWeAWH96hah0hx0mKApoUN+S3jCD
8ZUEq3CG5WosyFaior0ms/M6lxVg+qFpZMr40jiM8wmBmNdUPaRpFa00EXKSvEN4
wkOcy3/chrZhvYvaPynbqESWslyIWjBbS1fo8HmE4tccf3hw3BzO75pmkjMJy/nG
G5NfB0sV8TvwKClF7UYj16gKMFOmGzYrKsMJJZdACbPcEHZJsvs49SGTLLTQECk6
MGWpB8mco2gxMHOXAgMBAAGjggFBMIIBPTAdBgNVHSUEFjAUBggrBgEFBQcDAQYI
KwYBBQUHAwIwGQYDVR0RBBIwEIIOd3d3Lmdvb2dsZS5jb20waAYIKwYBBQUHAQEE
XDBaMCsGCCsGAQUFBzAChh9odHRwOi8vcGtpLmdvb2dsZS5jb20vR0lBRzIuY3J0
MCsGCCsGAQUFBzABhh9odHRwOi8vY2xpZW50czEuZ29vZ2xlLmNvbS9vY3NwMB0G
A1UdDgQWBBSG63QhKSoMLrf/MwcMKcIpgpLsezAMBgNVHRMBAf8EAjAAMB8GA1Ud
IwQYMBaAFErdBhYbvPZotXb1gba7Yhq6WoEvMBcGA1UdIAQQMA4wDAYKKwYBBAHW
eQIFATAwBgNVHR8EKTAnMCWgI6Ahhh9odHRwOi8vcGtpLmdvb2dsZS5jb20vR0lB
RzIuY3JsMA0GCSqGSIb3DQEBBQUAA4IBAQCHHplYC1ORmY7GI7NuJrV23e0Tc8NZ
zIHwBr/MKHeu07h8tXw6empmo8Jpl72xCAlMdiaJ6gnx3T8euJdj387P60uIvYba
nSJt8hxwlOKxHWbK5WoIjlSERfD0Q8rJ7Sv77wsKv7HKxJgsjn1Eg8VeO0ruhmgO
6PT8pdRYazvxl82Mzs3rqXZKCslIa1OBt/nKaBDJ8Rl+J+LZ0idFXCj/oRUhSoaw
W0+zmPBMCJJkSom61LumjGXgU/TMLBCWI2NZp7JhUOoOkeb/lOMcZJIYQ7+zCtJH
P3gUMNJhX3uyZH1FzyAg9rpenMGSYMUPB2MXmKQi5b2Zu4qHiLjsJ0MK</ds:X509Certificate>
                        </ds:X509Data>
                    </ds:KeyInfo>
                </ds:Signature>
                <saml2:Subject>
                    <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified" NameQualifier="www.example.com">
Tester</saml2:NameID>
                    <saml2:SubjectConfirmation
Method="urn:oasis:names:tc:SAML:2.0:cm:sender-vouches"/>
                </saml2:Subject>
                <saml2:Conditions NotBefore="2014-05-13T04:41:27.117Z"
NotOnOrAfter="2014-05-13T04:46:27.117Z"/>
                <saml2:AuthnStatement AuthnInstant="2014-05-13T04:41:27.113Z">
                    <saml2:AuthnContext>
                        <saml2:AuthnContextClassRef>
urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml2:AuthnContextClassRef>
                    </saml2:AuthnContext>
                </saml2:AuthnStatement>
            </saml2:Assertion>
            <wsse:SecurityTokenReference xmlns:wsse11="http://docs.oasis-
open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd"
wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0" wsu:Id="STRSAMLId-81591DAC97D1A4EF26139995608718320">
                <wsse:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID">
_81591DAC97D1A4EF26139995608705916</wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
Id="SIG-81591DAC97D1A4EF26139995608718622">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                        <ec:InclusiveNamespaces
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList="SOAP-ENV"/>
                    </ds:CanonicalizationMethod>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#id-81591DAC97D1A4EF26139995608718421">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                                <ec:InclusiveNamespaces
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" PrefixList=""/>
```

```
                            </ds:Transform>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>
JLjybHqBnly5B2u2yhvTCTnn3os=</ds:DigestValue>
                    </ds:Reference>
                    <ds:Reference URI="#STRSAMLId-
81591DAC97D1A4EF26139995608718320">
                        <ds:Transforms>
                            <ds:Transform Algorithm="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform">
                                <wsse:TransformationParameters>
                                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                                </wsse:TransformationParameters>
                            </ds:Transform>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>
g3PCuPeWIcXW9HFYYuLJp2lrVwM=</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
gGZU5Fwzd86oNABwaX0kzlWU0XVR4HUAp/F04WwxgVI7TThTK/e4OdvyvFJ2tt3kaItoWXhS+YgVnv+4Mc
</ds:SignatureValue>
                <ds:KeyInfo Id="KeyId-81591DAC97D1A4EF26139995608718317">
                    <wsse:SecurityTokenReference wsu:Id="STRId-
81591DAC97D1A4EF26139995608718318">
                        <wsse:Reference URI="#CertId-
81591DAC97D1A4EF26139995608718319" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
        <WSHeader xmlns="http://www.example.com/WSHeader.xsd">
            <UsernameToken>
                <Username>
Tester</Username>
                <Nonce>
ODE1OTFEQUM5N0QxQTRFRjI2MTM5OTk1NjA4NTUyOTE1</Nonce>
                <Created>
2014-05-13T04:41:25.529Z</Created>
            </UsernameToken>
        </WSHeader>
        <ns1:attachmentHash
xmlns:ns1="http://www.example.com/schemas/attachmenthash" SOAP-
ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" SOAP-
ENV:mustUnderstand="0">
            <ns1:hashValue>
7WxA7WJauYkMVd7KzK369YFQKS8=</ns1:hashValue>
        </ns1:attachmentHash>
        <ns1:standardAttachment
xmlns:ns1="http://www.example.com/Attachment.xsd">
            <Attachment>
                <id>
1</id>
                <compressFlag>
yes</compressFlag>
                <compressMethod>
```

```
gzip</compressMethod>
            </Attachment>
        </ns1:standardAttachment>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="id-
81591DAC97D1A4EF26139995608718421">
        <submitTest xmlns="http://www.example.com/Test">
            <AttachmentInfo xmlns="http://www.example.com/Attachment.xsd">
                <attachmentData>
                    <Include xmlns="http://www.w3.org/2004/08/xop/include"
href="cid:2b380066-5b7e-4d5c-949d-f11d41d1cd1b"/>
                </attachmentData>
            </AttachmentInfo>
        </submitTest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This describes the same issue except the Java producer uses the Metro framework for WS-Security instead of Apache WSS4J. The result, however, is the same. I would like to avoid hacking up the Java side, for multiple reasons, not least breaking compliance with the WS-Security SAML Token Profile. From the post: "We modified the Metro source to eliminate STR-Transform and directly sign the SAML assertion from the main signature (instead of using a SecurityTokenReference and the STR-Transform)." I don't think this would be an acceptable solution.

.net **saml** **wcf**

Share

Improve this question

Follow

edited Aug 10, 2014 at 0:18

asked Jun 11, 2014 at 16:56

Thomas

63 ● 1 ● 7

This is off topic here - I'll try to migrate it somewhere useful, but they may close it because you have 3 questions. It is much better to ask one question at a time - so it can be answered. – Rory Alsop Jun 11, 2014 at 22:56

How are you creating the client? Can you create a small reproducing case, and include both the code and the configuration? – John Saunders Jun 11, 2014 at 23:02

There is no full SAML2 support in the .NET framework. Is the client using any third party library, or have they implemented everything themselves? – Anders Abel Jun 13, 2014 at 7:27

@AndersAbel.Thank you very much for your reply. MS-LWSSP certainly does seem to amount to a statement of limited SAML2 support. The client is not using a third-party library. Would you be able to recommend one for this purpose? – Thomas Jun 17, 2014 at 17:34

1 Your edit showing the ultimate solution should be removed from the question and added as a separate answer, which can then be upvoted. – John Saunders Jul 22, 2014 at 20:21

1 Answer

Sorted by: Highest score (default)

2

I am answering my own question after working directly with Microsoft for some time. The title of this issue remains true but can now be clarified:

Windows Identity Foundation (WIF) has support for the STR-Transform algorithm specified for validating tokens signed by reference, but Windows Communication Foundation (WCF) does not.

WCF source includes SendSecurityHeader.cs, which handles XML Signature and XML Encryption for sending service messages. The issue at hand is about receiving messages, but a remark in OnWriteHeaderContents in SendSecurityHeader seems to get to the bottom of the problem.

Starting at line 606 of Microsoft's published source reference for SendSecurityHeader.cs, note the remarks:

```
if (elementContainer.SourceSigningToken != null)
{
if (ShouldSerializeToken(this.signingTokenParameters, this.MessageDirection))
{
this.StandardsManager.SecurityTokenSerializer.WriteToken(writer,
elementContainer.SourceSigningToken);

// Implement Protect token
// NOTE: The spec says sign the primary token if it is not included in the
message. But we currently are not supporting it
// as we do not support STR-Transform for external references. Hence we can not
sign the token which is external ie not in the message.
// This only affects the messages from service to client where
// 1. allowSerializedSigningTokenOnReply is false.
// 2. SymmetricSecurityBindingElement with IssuedTokens binding where the
issued token has a symmetric key.

if (this.ShouldProtectTokens)
{
    this.WriteSecurityTokenReferencyEntry(writer,
elementContainer.SourceSigningToken, this.signingTokenParameters);
}
}
}
```

This means that WCF does not support XML security constructs on externally referenced tokens (for example, a SAML assertion, as in this issue). This also means that Microsoft does not fully support the standard that they chaired and co-edited, namely Web Services Security SAML Token Profile Version 1.1.1. I am working with Microsoft on an enhancement request and am also evaluating the possibility of overriding the framework. The implication here is that a .NET service consumer lacks interoperability with a Java producer of a service message secured with a SAML assertion signed by reference.

Share Improve this answer Follow

answered Oct 24, 2014 at 16:53

Thomas

63 ● 1 ● 7

Did you find out how to make it work with WCF 4.5? I'm having a similar issue here: stackoverflow.com/questions/30125288/… – Thuan May 8, 2015 at 16:32

As far as I know, there has not been any change: WCF does not support tokens signed by reference. – Thomas May 19, 2015 at 16:55