Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 25, 2011

J. Schaad
Soaring Hawk Consulting
January 21, 2011

Email Policy Service Trust Processing
draft-schaad-eps-trust-00

Abstract

Write Me

Status of this Memo

This Internet-Draft will expire on July 25, 2011.

Copyright Notice

Table of Contents

1. Introduction

1.1. XML Nomenclature and Name Spaces

The following name spaces are used in this document:

Prefix	Namespace	Specification(s)
eps	http://ietf.org/2011/plasma/	This Specification
S11	http://schemas.xmlsoap.org/soap/envelope/	[SOAP11]
S12	http://www.w3.org/2003/05/soap-envelope	[SOAP12]
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512	[WS-TRUST]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-Security]
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-Security]
wsse11	http://docs.oasis-open.org/wss/oasis-wss-wsecurity-secext-1.1.xsd	[WS-Security]
ds	http://www.w3.org/2000/09/xmldsig#	[XML-Signature]
xenc	http://www.w3.org/2001/04/xmlenc#	[XML-Encrypt]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	[WS-Policy]
wsa	http://www.w3.org/2005/08/addressing	[WS-Addressing]
xs	http://www.w3.org/2001/XMLSchema	[XML-Schema1][XML-Schema2]

1.2. Requirements Terminology

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 [RFC2119].

2. Components

2.1. WS-Trust 1.3

We use WS-Trust as the basis for the protocol presented in this document. WS-Trust is a secure messaging protocol used for security token exchange to enable the issuance and dissemination of credentials within different trust domains. WS-Trust 1.3 is specified by OASIS in [WS-TRUST]. WS-Trust is built on SOAP (see [SOAP12]) to provide a messaging structure.

Implementers of this protocol MUST implement the HTTP binding.

Implementers of this protocol MUST implement SOAP 1.2. Support for SOAP 1.1 [SOAP11] is OPTIONAL.

3. Model

To be supplied from the problem statement document.

```
                     (1)(3)     +----------+
                   +----------->|Sending   |<------------+
                   |            |Agent     |             |
              (2)  v            +----------+             v
             +----------+           ^               +---------+
             |Email     |           |               |Mail     |
             |Policy    |<----------+               |Transfer |
             |Service   |                           |Agent    |
             +----------+                           +---------+
              ()   ^            +----------+             ^
                   |            |Receiving |             |
                   +----------->|Agent     |<------------+
                     ()()       +----------+
```

Figure 1: Message Access Control Actors

List the boxes above and give some info about them.

Email Policy Service is the gateway controller for accessing a message. Although it is represented as a single box in the diagram, there is no reason for it to be in practice. Each of the three protocols could be talking to different instances of a common system. This would allow for a server to operated by Company A, but be placed in Company B’s network thus reducing the traffic sent between the two networks.

Mail Transfer Agent is the entity or set of entities that is used to move the message from the sender to the receiver. Although this document describes the process in terms of mail, any method can be used to transfer the message.

Receiving Agent is the entity that consumes the message.

Sending Agent is the entity that originates the message.

3.1. Sender Processing

We layout the general steps that need to be taken by the sender of an EPS message. The numbers in the steps below refer to the numbers in the upper half of Figure 1. A more detailed description of the processing is found in Section 4 for obtaining the security policies that can be applied to a messages and Section 5 for sending a message.

1. The Sending Agent sends a message to one or more Email Policy Services in order to obtain the set of policies that it can apply to a message along with a security token to be used in proving the authorization. Details of the message send can be found in Section 4.1.

2. The Email Policy Service examines the set of policies that it understands and checks to see if the requester is authorized to send messages with the policy.

3. The Email Policy Service returns the set of policies and an security token to the Sending Agent. Details of the message sent can be found in Section 4.2.

4. The Sending Agent selects the Email Policy(s) to be applied to the message, along with the set of recipients for the message.

5. The Sending Agent relays the selected information to the Email Policy Service along with the security token. Details of this message can be found in Section 5.1.

6. The Email Policy Service creates the recipient info attribute as defined in [EPS-ASN].

7. The Email Policy Service returns the created attribute to the Sending Agent. Details of this message can be found in Section 5.2.

8. The Sending Agent composes the CMS EnvelopedData content type placing the returned attribute into a KEKRecipientInfo structure and then send the message to the Mail Transport Agent.

3.2. Recieving Agent Processing

We layout the general steps that need to be taken by the sender of an EPS message. The numbers in the steps below refer to the numbers in the lower half of Figure 1. A more detailed description of the processing is found in Section 6.

1. The Receiving Agent obtains the message from the Mail Transport Agent.

2. The Receiving Agent starts to decode the message and in that process locates an EvelopedData content type which has a KEKRecipientInfo structure with a XXXX attribute.

3. The Receiving Agent processes the SignedData content of the XXXX attribute to determine that communicating with it falls within accepted policy.

4. The Receiving Agent transmits the content of the XXXX attribute to the referenced Email Policy Service. The details of this message can be found in Section 6.1.

5. The Email Policy Service decrypts the content of the message and applies the policy to the credentials provided by the Receiving Agent.

6. If the policy passes, the Email Policy Service returns the appropriate key or RecipientInfo structure to the Receiving Agent. Details of this message can be found in Section 6.2.

7. The Receiving Agent proceeds to decrypt the message and perform normal processing.

4. Initial Token and Policy Acquisition

The first step in the process is for the sending agent to acquire the set of policies that it is permitted to use in labeling a message. This is done by a request and response. For this purpose we define two new uri values to be used in the wst:RequestType field:

urn:ietf:params:ns:eps-xml:RequestSendToken is used to identify a request to receive a set of security policies that can be used along with a security token to identify the sending agent when sending a message.

It is assumed that the Email Policy Server will do an exhaustive set of tests to check which security policies are usable by the sending agent in order to label messages. As this is going to be a computationally intensive operation, the process is expected to be done infrequently compared to sending messages. The data and security token returned is therefore expected to be good for a period of time. In situations where changes to privileges change and it is important that the system correctly enforce them, then a subsequence check on just the label presented at the time the mail message is sent.

4.1. Request Policy Information

Send a wst:RequestSecurityToken message to the Email Policy Service. The request will contain at least the following elements:

A wst:RequestType containing a urn:ietf:params:ns:eps-xml:#RequestSendToken URI MUST be included.

An example of a message requesting the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityToken>
      <wst:RequestType>
      urn:ietf:params:xml:ns:eps-xml:#RequestSendToken
      </wst:RequestType>
    </wst:RequestSecurityToken>
  </s12:Body>
</s12:Envelope>
```

In this example the identity information of the requester is implicit from the transport protocol used.

4.2. Request Policy Information Response

Receive a wst:RequestSecurityTokenResponse message with the following elements:

A wst:RequestedSecruityToken element containing the security token MUST be included. The format of the security token is not specified and is implementation specific, it is not expected that . Examples of items that could be used as security tokens are SAML statements, encrypted record numbers in a server database.

A eps:PolicySet containing the set of policies that the server has been ascertained are acceptable for the querier to use in labeling email messages MUST be included.

A wst:Lifetime giving the life time of the token SHOULD be included. It is not expected that this should be determinable from the token itself and thus must be independently provided. There is no guarantee that the token will be good during the lifetime as it make get revoked due to changes in credentials, however the client is permitted to act as if it where. The token provided may be used for duration. If this element is absent, it should be assumed that the token is either a one time token or of limited duration.

An example of a message returning the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityTokenResponse>
      <wst:RequestedSecurityToken>
        <me:CustomToken>ABCDEFGHIJKLMN
        </me:CustomToken>
      </wst:RequestedSecurityToken>
      <wst:RequestedProofToken>
        <wst:BinarySecret>PGRGFCDE</wst:BinarySecret>
      </wst:RequestedProofToken>
      <eps:PolicySet>
        <eps:Policy>
          <eps:Name>Policy Name #1</eps:Name>
          <eps:Identifier>
            http://this.is.a.com/policyX
          </eps:Identifier>
          <eps:ReferencePoint>
            http://Point.com/serverName
          </eps:ReferencePoint>
          <eps:ReferencePoint>
        http://ietf.org/email-policy-servers/ad-hoc/PolicyServer
```

```
            </eps:ReferencePoint>
            <eps:Options>
              <eps:Option name="Category">
                <eps:OptionValue value="1">Non-classified
                </eps:OptionValue>
                <eps:OptionValue value="2">Restricted
                </eps:OptionValue>
                <eps:OptionValue value="3">Classified
                </eps:OptionValue>
                <eps:OptionValue value="4">Don’t Read Me
                </eps:OptionValue>
              </eps:Option>
            </eps:Options>
          </eps:Policy>
          <eps:Policy>
            <eps:Name>Ad Hoc Corperate Policy</eps:Name>
            <eps:Identifier>
              http://ietf.org/email-policies/ad-hoc
            </eps:Identifier>
            <eps:ReferencePoint>
          http://ietf.org/email-policies/ad-hoc/PolicyServer
            </eps:ReferencePoint>
          </eps:Policy>
          <eps:Policy>
            <eps:Name>IETF Basic Policy 1</eps:Name>
            <eps:Identifier>
              http://ietf.org/email-policies/basic-1
            </eps:Identifier>
            <eps:ReferencePoint>
          http://ietf.org/email-policy-servers/ad-hoc/PolicyServer
            </eps:ReferencePoint>
            <eps:AllowNameList value="yes"/>
          </eps:Policy>
        </eps:PolicySet>
      </wst:RequestSecurityTokenResponse>
    </s12:Body>
  </s12:Envelope>
```

In this example, the Email Policy Service is returning three different policies that can be used along with a security token and a key to be used with the token when sending a message.

5. Sending A Message

When the sending agent is ready to build the list of recipient info structures, it builds a request message containing the label, the key encryption key and other information required for decryption to send to the Email Policy Service. It will then get back a response containing a CMS SignedData object to be included in a KEKRecipientInfo object.

To identify this operation we have defined a new uri urn:ietf:params:ns:eps-xml:RequestSendToken.

5.1. Send Message Request

The process we are looking at is: Send a wst:RequestSecurityToken to the Email Policy Service. The request MUST contain at least the following elements:

> A wst:RequestType containing a urn:ietf:params:ns:eps-xml:RequestSendToken URI.
>
> Put in the previously assigned tokens as if you were doing a token renewal.
>
> An eps:SendMessage as defined in this document.

An example of a message returning the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityToken>
      <wst:RequestType>
        urn:ietf:params:ns:eps-xml:RequestSendToken
      </wst:RequestType>
      <eps:SendMessageRequest>
        <eps:RecipientData>
          <eps:CompoundLabel action="or">
            <eps:Label
              name="http://ietf.org/policies/basic1">
              <eps:addressList>
                jimsch@example.com;
                patrick@example.com;
                paul@example.com
            </eps:addressList></eps:Label>
            <eps:CompoundLabel action="and">
              <eps:Label
        name="http://this.is.a.com/policyX?Category=4"/>
              <eps:Label name="http://ietf.org/policies"/>
            </eps:CompoundLabel>
          </eps:CompoundLabel>
          <eps:Recipient name="trevor@microsoft.com">
            <eps:Key>
              <eps:Identifier>....</eps:Identifier>
              <eps:RecipientInfo>....</eps:RecipientInfo>
            </eps:Key>
            <eps:Key>
              <eps:Identifier>....</eps:Identifier>
              <eps:RecipientInfo>....</eps:RecipientInfo>
            </eps:Key>
          </eps:Recipient>
          <eps:DefaultRecipient>
            <eps:Identifier>....</eps:Identifier>
            <wst:BinarySecret></wst:BinarySecret>
          </eps:DefaultRecipient>
        </eps:RecipientData>
      </eps:SendMessageRequest>
    </wst:RequestSecurityToken>
  </s12:Body>
</s12:Envelope>
```

5.2. Send Message Response

Receive a wst:RequestSecurityTokenResponse from the Email Policy Service. The response MUST contain at least the following elements:

An eps:SendMessageResponse as defined in this document.

An example of a message returning the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityTokenResponse>
      <eps:SignedDataBlob/>
    </wst:RequestSecurityTokenResponse>
  </s12:Body>
</s12:Envelope>
```

6. Decoding A Message

When the receiving agent is ready to decrypt the message, it identifies that there is a KEKRecipientInfo object which contains a key attribute identified by id-keyatt-eps-token. It validates that communicating with the Email Policy Service is within local policy and then sends a request to the service to obtain the encryption key for the message.

To identify this operation we have defined a new uri urn:ietf:params:ns:eps-xml:RequestReadToken.

In some cases the recipient of a message is not authorized to use the same set of labels for sending a message. For this purpose a token can be returned in the message along with the key so that recipient of the can reply to the message using the same set of security labels.

6.1. Requesting Message Key

Send a wst:RequestSecurityToken message to the EMail Policy Server. The request MUST contain at least the following elements:

A wst:RequestType containing a urn:ietf:params:ns:eps-xml:RequestReadToken URI.

A eps:ReadMessageRequest defined in this document.

An example of a message returning the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityToken>
      <wst:RequestType>
        urn:ietf:params:ns:eps-xml:RequestReadToken
      </wst:RequestType>
      <eps:SignedDataBlob/>
    </wst:RequestSecurityToken>
  </s12:Body>
</s12:Envelope>
```

6.2. Requesting Message Key Response

Receive a wst:RequestSecurityTokenResponse message from the Email Policy Server. The response contains the following elements:

An eps:ReadMessageResponse.

An example of a message returning the set of policy information is:

```
<s12:Envelope>
  <s12:Body>
    <wst:RequestSecurityTokenResponse>
      <wst:RequestedSecurityToken>
        <me:CustomToken>....</me:CustomToken>
      </wst:RequestedSecurityToken>
      <wst:RequestedProofToken>
        <wst:BinarySecret></wst:BinarySecret>
      </wst:RequestedProofToken>
      <eps:Key>...</eps:Key>
      <eps:Key>...</eps:Key>
      <eps:PolicySet>....</eps:PolicySet>
    </wst:RequestSecurityTokenResponse>
  </s12:Body>
</s12:Envelope>
```

7. Security Considerations

To be supplied after we have a better idea of what the document looks like.

8. IANA Considerations

We should have at least one name space to be registered.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[EPS-ASN] Schaad, J., "Email Policy Service ASN.1 Processing", Work In Pgoress draft-eps-smime-00, Jan 2011.

[SOAP11] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C NOTE NOTE-SOAP-20000508, May 2000.

[SOAP12] Lafon, Y., Gudgin, M., Hadley, M., Moreau, J., Mendelsohn, N., Karmarkar, A., and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)", World Wide Web Consortium Recommendation REC-soap12-part1-20070427, April 2007, <http://www.w3.org/TR/2007/REC-soap12-part1-20070427>.

[WS-TRUST] Lawrence, K., Kaler, C., Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and H. Granqvist, "WS-Trust 1.3", OASIS Standard ws-trust-200512, March 2007, <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>.

Appendix A. XML Schema

Author's Address

Jim Schaad
Soaring Hawk Consulting

Email: ietf@augustcellars.com