Network Working Group R. Housley
Internet-Draft Vigil Security, LLC
Intended status: Standards Track S. Ashmore
Expires: April 9, 2009 National Security Agency
C. Wallace
Cygnacom Solutions
October 6, 2008

Cryptographic Message Syntax (CMS) Content Constraints X.509 Certificate Extension

draft-ietf-pkix-cms-content-constraints-00

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 9, 2009.

Abstract

This document specifies the syntax and semantics for the Cryptographic Message Syntax (CMS) content constraints X.509 certificate extension. This extension is used to determine whether the public key in an X.509 public key certificate is appropriate to use in the processing of a protected content. In particular, the CMS content constraints certificate extension is one part of the authorization decision; it is used when validating a digital signature on a CMS SignedData content or validating a message authentication code (MAC) on a CMS AuthenticatedData content or CMS AuthEnvelopedData content. The signed or authenticated content type is identified by an ASN.1 object identifier, and this certificate extension indicates the content types that the certified public key is authorized to validate. If the authorization check is successful, the CMS content constraints certificate extension also provides default values for absent attributes.

Table of Contents

1. Introduction

The CMS SignedData [RFC3852] construct is used to sign many things, including cryptographic module firmware packages [RFC4108] and certificate management messages [RFC5272]. Similarly, the CMS AuthenticatedData and CMS AuthEnvelopedData constructs provide authentication, which can be affiliated with an originator's X.509 certificate.

This document assumes a particular authorization model, where each originator is associated with one or more authorized content types. A CMS SignedData, AuthenticatedData, or AuthEnvelopedData will be considered valid only if the signature or message authentication code (MAC) verification process is successful and the originator is authorized for the encapsulated content type. For example, one originator might be acceptable for verifying signatures on firmware packages, but that same originator may be unacceptable for verifying signatures on certificate management messages.

An originator's constraints are derived from the certification path used to validate the originator's certificate. Constraints are associated with trust anchors and constraints are optionally included in public key certificates. The trust anchor structure lists the content types for which it may be used, and the trust anchor may also include constraints associated with each of the content types. Certificates may include a CMS Content Constraints certificate extension that refines the privileges of the trust anchor for a particular certificate subject.

The entity that operates a trust anchor holds the corresponding private signature key, and that entity may delegate authority to another entity. Delegation is accomplished by issuing an X.509 certificate to that other entity. If the trust anchor issues a certification authority (CA) certificate, then that entity may perform further delegation. If the trust anchor issues an end entity certificate, then that entity is unable to perform further delegation.

The CMS Content Constraints certificate extension provides a mechanism to constrain the authorizations that are delegated when a certificate is issued by a trust anchor or a CA. The certificate containing the CMS Content Constraints certificate extension is limited to a subset of the content types associated with the certificate issuer, whether the issuer is a trust anchor or a CA. Also, the certificate issuer may add constraints to a content type that is not constrained for the certificate issuer. However, no amplification of authorization is possible through use of this certificate extension. When a content signature or MAC is validated,

checks must be performed to ensure that the encapsulated content type is within the permitted set and that the constraints associated with the specific content type, if any, are satisfied.

1.1. CMS Data Structures

CMS encapsulation can be used to compose structures of arbitrary breadth and depth. Four documents define the primary CMS content types:

RFC 3852 [RFC3852]: Cryptographic Message Syntax (CMS)

- SignedData
- EnvelopedData
- EncryptedData
- DigestedData
- AuthenticatedData

RFC 5083 [RFC5083]: The Cryptographic Message Syntax (CMS) AuthEnvelopedData Content Type

- AuthEnvelopedData

RFC 4073 [RFC4073]: Protecting Multiple Contents with the Cryptographic Message Syntax (CMS)

- ContentCollection
- ContentWithAttributes

RFC 3274 [RFC3274]: Compressed Data Content Type for Cryptographic Message Syntax (CMS)

- CompressedData

When using the CMS, the outermost structure is always ContentInfo. ContentInfo consists of an object identifier and an associated content. The object identifier describes the structure of the content. Object identifiers are used throughout the CMS family of specifications to identify structures.

Using the content types listed above, ignoring for the moment ContentCollection, encapsulation can be used to create structures of arbitrary depth. Two examples based on [RFC4108] are shown in Figure

1 and Figure 2.

When ContentCollection is used in conjunction with the other content types, tree-like structures can be defined, as shown in Figure 3.

The examples in Figures 1, 2, and 3 can each be represented as a tree: the root node is the outermost ContentInfo, and the leaf nodes are the encapsulated contents. The trees are shown in Figure 4.

```
+---------------------------------------------------------+
| ContentInfo                                             |
|                                                         |
| +-----------------------------------------------------+ |
| | SignedData                                          | |
| |                                                     | |
| | +-------------------------------------------------+ | |
| | | FirmwarePackage                                 | | |
| | |                                                 | | |
| | |                                                 | | |
| | +-------------------------------------------------+ | |
| +-----------------------------------------------------+ |
+---------------------------------------------------------+
```

Figure 1. Example of a Signed Firmware Package.

```
+---------------------------------------------------------+
| ContentInfo                                             |
|                                                         |
| +-----------------------------------------------------+ |
| | SignedData                                          | |
| |                                                     | |
| | +-------------------------------------------------+ | |
| | | EncryptedData                                   | | |
| | |                                                 | | |
| | | +---------------------------------------------+ | | |
| | | | FirmwarePackage                             | | | |
| | | |                                             | | | |
| | | |                                             | | | |
| | | +---------------------------------------------+ | | |
| | +-------------------------------------------------+ | |
| +-----------------------------------------------------+ |
+---------------------------------------------------------+
```

Figure 2. Example of a Signed and Encrypted Firmware Package.

These examples do not illustrate all of the details of the CMS structures; most CMS protecting content types, and some content types, contain attributes. These attributes can influence processing and handling of the CMS protecting content type or the encapsulated content type. Throughout this document, paths through the tree structure from a root node to a leaf node in a CMS-protected message are referred to as CMS paths.

```
         +---------------------------------------------------------+
         | ContentInfo                                             |
         |                                                         |
         | +-----------------------------------------------------+ |
         | | SignedData                                          | |
         | |                                                     | |
         | | +-------------------------------------------------+ | |
         | | | ContentCollection                               | | |
         | | |                                                 | | |
         | | | +----------------------+ +--------------------+ | | |
         | | | | SignedData           | | SignedData         | | | |
         | | | |                      | |                    | | | |
         | | | | +------------------+ | | +----------------+ | | | |
         | | | | | EncryptedData    | | | | Firmware       | | | | |
         | | | | |                  | | | | Package        | | | | |
         | | | | | +--------------+ | | | |                | | | | |
         | | | | | | Firmware     | | | | +----------------+ | | | |
         | | | | | | Package      | | | +--------------------+ | | |
         | | | | | |              | | |                        | | |
         | | | | | +--------------+ | |                        | | |
         | | | | +------------------+ |                        | | |
         | | | +----------------------+                        | | |
         | | +-------------------------------------------------+ | |
         | +-----------------------------------------------------+ |
         +---------------------------------------------------------+
```

Figure 3. Example of Two Firmware Packages in a Collection.

1.2. CMS Content Constraints Model

The CMS content constraints certificate extension is used to restrict the types of content for which a particular public key can be used to verify a signature or MAC. Trust in a public key is established by building and validating a certification path from a trust anchor to the subject public key. Section 6 of [RFC5280] describes the algorithm for certification path validation, and the basic path validation algorithm is augmented, as described in Section 3 of this document, to include processing required to determine the CMS content

constraints that have been delegated to the subject public key. If the subject public key is explicitly trusted (the public key belongs to a trust anchor), then any CMS content constraints associated with the trust anchor are used directly. If the subject public key is not explicitly trusted, then the CMS content constraints are determined by calculating the intersection of the CMS content constraints included in each certificate in a valid certification path from the trust anchor to the subject public key.

```
         +---------------------------------------------------------+
         |                                                         |
         |     CMS PATH RESULTING            CMS PATH RESULTING    |
         |       FROM FIGURE 1.                FROM FIGURE 2.      |
         |                                                         |
         |       ContentInfo                   ContentInfo         |
         |           |                             |               |
         |           V                             V               |
         |       SignedData                    SignedData          |
         |           |                             |               |
         |           V                             V               |
         |       FirmwarePackage               EncryptedData       |
         |                                         |               |
         |                                         V               |
         |                                     FirmwarePackage     |
         |                                                         |
         |                                                         |
         |            CMS PATHS RESULTING FROM FIGURE 3.           |
         |                                                         |
         |                       ContentInfo                       |
         |                           |                             |
         |                           V                             |
         |                       SignedData                        |
         |                           |                             |
         |                           V                             |
         |                       ContentCollection                 |
         |                           |                             |
         |                +----------+--------------+              |
         |                |                         |              |
         |                V                         V              |
         |            SignedData                SignedData         |
         |                |                         |              |
         |                V                         V              |
         |            EncryptedData             FirmwarePackage    |
         |                |                                        |
         |                V                                        |
         |            FirmwarePackage                              |
         |                                                         |
         +---------------------------------------------------------+
```

Figure 4. Example CMS Path Structures.

The CMS enables the use of multiple nested signatures or MACs. Each signature or MAC can protect and associate attributes with an encapsulated data object. The CMS content constraints certificate extension is associated with a public key, and that public key is used to verify a signature or a MAC.

The CMS content constraints mechanism can be used to permit the use of the subject public key to verify signatures on or authenticate one or more specific content types. Also, within a permitted content type, a permitted set of values can be expressed for one or more specific attribute types.

When multiple parties collaborate to produce a signed or authenticated CMS-protected content, each party must be authorized for the content types and attribute values appearing in the result. In all cases, the signer or originator closest to a leaf node must be authorized to serve as a source for the leaf node contents; outer signers or originators need not be authorized to serve as a source.

A signer or originator may be constrained to use a specific set of attribute values for some attribute types when producing a particular content type. If a signer or originator is constrained for a particular attribute that does not appear in a protected content of the type for which the constraint is defined, the constraint serves as a default attribute, i.e., the payload should be processed as if an attribute equal to the constraint appeared in the protected content. However, in some cases, the processing rules for a particular content type may disallow the usage of default values for some attribute types and require a signer to explicitly assert the attribute to satisfy the constraint.

1.3. Abstract Syntax Notation

All X.509 certificate [RFC5280] extensions are defined using ASN.1 [X.680][X.690].

CMS content types [RFC3852] are also defined using ASN.1.

CMS uses the Attribute type. The syntax of Attribute is compatible with X.501 [X.501].

1.4. 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 RFC 2119 [RFC2119].

2. CMS Content Constraints X.509 Certificate Extension

The CMS content constraints certificate extension MAY be critical, and it MUST appear at most one time in a certificate. The CMS content constraints certificate extension is identified by the id-pe-cmsContentConstraints object identifier:

```
      id-pe-cmsContentConstraints OBJECT IDENTIFIER ::=
          { iso(1) identified-organization(3) dod(6) internet(1)
            security(5) mechanisms(5) pkix(7) pe(1) 18 }
```

The CMS content constraints certificate extension provides a mechanism to constrain authorization during delegation. If the CMS content constraints certificate extension is not present, then the subject of the certificate has the same set of authorizations as the issuer of the certificate. A certificate containing the CMS content constraints certificate extension is limited to a subset of the content types for which the certificate issuer is authorized. Also, the certificate issuer may add constraints to a previously unconstrained content type, or add additional constraints to a previously constrained content type.

The syntax for the CMS content constraints certificate extension is:

```
  CMSContentConstraints ::= ContentTypeConstraintList

  ContentTypeConstraintList ::= SEQUENCE SIZE (1..MAX) OF
    ContentTypeConstraint

  ContentTypeConstraint ::= SEQUENCE {
    contentType           ContentType,
    canSource             BOOLEAN DEFAULT TRUE,
    attrConstraints       AttrConstraintList OPTIONAL }

  AttrConstraintList ::= SEQUENCE SIZE (1..MAX) OF AttrConstraint

  AttrConstraint ::= SEQUENCE {
    attrType               AttributeType,
    attrValues             SET SIZE (1..MAX) OF AttributeValue }

  ContentType ::= OBJECT IDENTIFIER

  id-ct-anyContentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
         us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
         ct(1) 0 }
```

The CMSContentConstraints is a list of permitted content types and associated constraints. A particular content type MUST NOT appear more than once in a ContentTypeConstraintList. When the extension is present, the certificate subject is being authorized by the certificate issuer to sign or authenticate the content types listed in the permitted list as long as the provided constraints, if any, are met. The certificate issuer MUST be authorized to delegate the privileges. When the extension is absent, the certificate issuer is authorizing the subject to sign or authenticate all of the content types for which the issuer is authorized.

The special id-ct-anyContentType value indicates the certificate subject is being authorized for any content type without any constraints. The id-ct-anyContentType object identifier can be used in trust anchor certificates when the trust anchor is unconstrained. Where id-ct-anyContentType is asserted in the contentType field, canSource and attrConstraints MUST BE absent, indicating the trust anchor can serve as a source for any content type without any constraints.

The fields of the ContentTypeConstraint type have the following meanings:

contentType contentType is an object identifier that specifies a permitted content type. When the extension appears in an end entity certificate, it indicates that a content of this type can be verified using the public key in the certificate. When the extension appears in a CA certificate, it indicates that a content of this type can be verified using the public key in the CA certificate or the public key in an appropriately authorized subordinate certificate. For example, this field contains id-ct-firmwarePackage when the certified public key can be used to verify digital signatures on firmware packages defined in [RFC4108]. A particular content type MUST NOT appear more than once in the list. The CMS-related content types need not be included in the list of permitted content types. These content types are always authorized to facilitate the use of CMS in the protection of content, and they MUST NOT appear in the contentType field. The always authorized content types are:

id-signedData,

id-envelopedData,

id-digestedData,

id-encryptedData,

id-ct-authEnvData,

id-ct-authData,

id-ct-compressedData,

id-ct-contentCollection

id-ct-contentWithAttrs.

canSource canSource is a Boolean flag, and it applies to direct signatures or direct authentication for the specified content type. If the canSource flag is FALSE, then the subject cannot directly sign or authenticate the specified content type. Regardless of the flag value, a subject can sign or authenticate a content that is already authenticated (when SignedData, AuthenticatedData, or AuthEnvelopedData is already present).

attrConstraints attrConstraints is an optional field that contains constraints that are specific to the content type. If the attrConstraints field is absent, the certified public key can be used to verify the specified content type without further checking. If the attrConstraints field is present, then the certified public key can only be used to verify the specified content type if all of the constraints are satisfied. A particular constraint type MUST NOT appear more than once in the attrConstraints. Constraints are checked by matching the values in the constraint against the corresponding attribute value in the content. Constraints processing fails if the attribute is present and the value is not one of the values provided in the constraint. Constraint checking is described fully in section 4.

The fields of the AttrConstraint type have the following meanings:

attrType attrType is an AttributeType, which is an object identifier that names an attribute. For a content encapsulated in a CMS SignedData, AuthenticatedData, or AuthEnvelopedData to satisfy the constraint, if the attributes that are covered by the signature or MAC include an attribute of the same type, then the attribute value must be equal to one of the values supplied in the attrValues field.

attrValues attrValues is a set of AttributeValue. The structure of each of the values in attrValues is determined by attrType. Constraint checking is described fully in section 4.

3. Certification Path Processing

When CMS content constraints are used for authorization, the processing described in this section MUST be included in the certification path validation. The processing is presented as additions to the certification path validation algorithm described in section 6 of [RFC5280]. Alternative implementations are possible but MUST yield the same results as described below.

Certification path processing validates the binding between the subject and subject public key. If a valid certification path cannot be found, then the corresponding CMS path MUST be rejected.

3.1. Inputs

If the trust anchor that terminates the path is authorized using CMS Content Constraints, then the trust anchor information includes a CMS Content Constraints structure. The trust anchor may be constrained or unconstrained, and if unconstrained it will include a CMS Content Constraints structure with a single permitted content type equal to the special id-ct-anyContentType value. In some cases, a particular CMS Content Constraints definition may be implied by the trust anchor information or application context. Otherwise, if the trust anchor does not contain a CMS Content Constraints structure, the CMS content constraints processing fails due to invalid input.

The content type of the protected content being verified is provided as input along with the set of attributes collected from the CMS path. Alternatively, the id-ct-anyContentType value can be provided as the content type input, along with an empty set of attributes, to determine the full set of constraints associated with a public key in the end entity certificate terminating the certification path being validated.

In some cases, a trust anchor may directly sign an object other than an X.509 certificate. In these cases, certification path validation as described in section 6 of [RFC5280] is not necessary but constraints processing must still be performed for the trust anchor. In such cases, the initialization and wrap-up steps described below can be performed to determine if the public key in the trust anchor is appropriate to use in the processing of a protected content.

3.2. Initialization

Create a constant input variable named cms_content_type and set it equal to the content type provided as input.

Create a constant input variable named cms_effective_attributes and

set it equal to the set of attributes provided as input.

Create a state variable named working_permitted_content_types. The initial value of working_permitted_content_types is the permitted content type list from the trust anchor, including any associated constraints.

Create an state variable of type SEQUENCE OF AttrConstaint named subject_default_attributes and initialize it to empty.

Create an state variable of type SEQUENCE OF ContentTypeConstraint named subject_constraints and initialize it to empty.

3.3. Basic Certificate Processing

If the CMS content constraints certificate extension is not present in the certificate or if the certificate is present and includes a single permitted content type equal to the special id-ct-anyContentType value, no action is taken and working_permitted_content_types is unchanged.

If the CMS content constraints certificate extension is present in the certificate, the extension contains a list of two or more permitted content types, one of which is the special id-ct-anyContentType value, constraints processing fails and certification path processing fails.

If the CMS content constraints certificate extension is present in the certificate, the extension contains a list of permitted content types, and working_permitted_content_types contains the id-ct-anyContentType special value, assign working_permitted_content_types the value of the CMS content constraints certificate extension.

If the CMS content constraints certificate extension is present in the certificate, the extension contains a list of permitted content types, and working_permitted_content_types does not contain the id-ct-anyContentType special value, then the processing actions to be performed for each entry in the permitted content type list sequence in the CMS content constraints certificate extension are as follows:

- If the CMS content constraints certificate extension includes a content type that is not present in working_permitted_content_types, no action is taken based on this entry. working_permitted_content_types is unchanged.

- If the CMS content constraints certificate extension includes a content type that is already present in working_permitted_content_types, then the constraints in the CMS

content constraints certificate extension can further reduce the authorization by adding constraints to previously unconstrained attributes or by removing attribute values from the attrValues set of a constrained attribute. Similarly, the canSource flag can be further constrained by setting the value to FALSE. The processing actions to be performed for each entry in the AttrConstraintList follow:

-- If the CMS content constraints certificate extension includes an attribute type that is not present in working_permitted_content_types for this content type, add the attribute type and the associated set of attribute values to working_permitted_content_types entry for the content type.

-- If the CMS content constraints certificate extension includes an attribute type that is already present in working_permitted_content_types for this content type, then compute the intersection of the set of attribute values from the working_permitted_content_types and the set of attribute values from the CMS content constraints certificate extension. If the intersection contains at least one attribute value, then the set of attribute values in working_permitted_content_types entry for this content type is assigned the intersection. If the intersection is empty, then the entry associated with the content type is removed from working_permitted_content_types.

Remove each entry in working_permitted_content_types that includes a content type that is not present in the CMS content constraints certificate extension.

3.4. Preparation for Certificate i+1

No additional action associated with the CMS content constraints certificate extension is taken during this phase of certification path validation as described in section 6 of [RFC5280].

3.5. Wrap-up procedure

If cms_content_type equals the special value anyContentType, the CCC processing portion of path validation succeeds. Set subject_constraints equal to working_permitted_content_types. If subject_constraints is empty, then the public key in the end entity certificate of the certification path is not authorized for any content type (though alternative certification paths may exist that yield different results). If cms_content_type is not equal to the special value anyContentType, perform the following steps:

- If working_permitted_content_types is equal to the special value anyContentType, set subject_constraints equal to working_permitted_content_types; the CCC processing portion of path validation succeeds.

- If cms_content_type does not equal the content type of an entry in working_permitted_content_types, constraints processing fails and path validation fails.

- If cms_content_type equals the content type of an entry in working_permitted_content_types, add the entry from working_permitted_content_types to subject_constraints.

- If the attrConstraints field of the corresponding entry in working_permitted_content_types is absent; the CCC processing portion of path validation succeeds.

- If the attrConstraints field of the corresponding entry in working_permitted_content_types is present, then constraints must be checked. For each attrType in the attrConstraints, the constraint is satisfied if either the attribute type is absent from cms_effective_attributes or each attribute value in the attrsValues field of the corresponding entry in cms_effective_attributes is equal to one of the values for this attribute type in the attrConstraints field. If cms_effective_attributes does not contain an attribute of that type, then the entry from attrConstraints is added to the subject_default_attributes for use in processing the payload.

3.6. Outputs

If certification path validation processing succeeds, return the value of the subject_constraints and subject_default_attributes variables.

4. CMS Content Constraints Processing

CMS content constraints processing consists of four primary activities:

- Collection of Signer or Originator Keys

- Collection of Attributes

- Leaf node classification

- Content Type and Constraint Checking

Processing is performed for each CMS path from the root node of a CMS-protected content to a leaf node, proceeding from the root node to the leaf node. Each path is processed independently of the other paths. Thus, it is possible that some leaf nodes in a content collection may be acceptable while other nodes are not acceptable. The processing described in this section applies to CMS paths that contain at least one SignedData, AuthEnvelopedData, or AuthenticatedData node.

Signer or originator public keys are collected when verifying signatures or message authentication codes (MACs). These keys will be used to determine the constraints of each signer or originator by building and validating a certification path to the public key. Public key values, public key certificates or public key identifiers are accumulated in a state variable named cms_public_keys, which is either initialized to empty or to an application provided set of keys when processing begins. The variable will be updated each time a SignedData, AuthEnvelopedData, or AuthenticatedData node is encountered in the CMS path.

Attributes are collected from each node after the first SignedData, AuthEnvelopedData, or AuthenticatedData in a CMS path, including the attributes protected by the first SignedData, AuthEnvelopedData, or AuthenticatedData. During processing, attributes collected from the nodes in the CMS path are maintained in a state variable named cms_effective_attributes and default attributes derived from message originator authorizations are collected in a state variable named cms_default_attributes. A default attribute value comes from a constraint that does not correspond to an attribute contained in the CMS path. When processing begins, cms_effective_attributes and cms_default_attributes are initialized to empty. Alternatively, cms_effective_attributes may be initialized to an application-provided sequence of attributes. The cms_effective_attributes value will be updated each time an attribute set is encountered in a SignedData, AuthEnvelopedData, or AuthenticatedData node while

processing a CMS path.

The output of content type and constraint checking always includes a set of attributes collected from the various nodes in a CMS path. When processing terminates at an encrypted node, the set of signer or originator public keys is also returned. When processing terminates at a payload node, a set of default attribute values is also returned along with a set of constraints that apply to the CMS-protected content.

When processing terminates at an encrypted node, the attributes and public keys are returned and may be used as inputs for CMS content constraints processing of the decrypted payload contents. An application may elect to discard some attributes before processing an encrypted payload. For example, attributes related to routing the encrypted content may be discarded or the MessageDigest and ContentType related to an outer signature layer may be discarded.

This section describes the processing of a CMS path. The output from CMS Content Constraints processing will depend on the type of the leaf node that terminates the CMS path. Four different output variables are possible. The conditions under which each is returned is described in the following sections. The variables are:

cms_public_keys cms_public_keys is a list of public key values, public key certificates or public key identifiers. Information maintained in cms_public_keys will be used to performed the certification path operations required to determine if a particular signer or originator is authorized to produce a specific object.

cms_effective_attributes cms_effective_attributes contains the attributes collected from the nodes in a CMS path. cms_effective_attributes is a SEQUENCE OF Attribute, which is the same as the AttrConstraintList structure except that it may have zero entries in the sequence.

cms_default_attributes cms_default_attributes contains default attributes derived from message signer or originator authorizations. A default attribute value is taken from a constraint that does not correspond to an attribute contained in the CMS path. cms_default_attributes is a SEQUENCE OF Attribute, which is the same as the AttrConstraintList structure except that it may have zero entries in the sequence.

cms_constraints cms_constraints contains the constraints associated with the message signer or originator for the content type of the protected content terminating a CMS path. cms_constraints is a SEQUENCE OF Attribute, which is the same as the AttrConstraintList structure except that it may have zero entries in the sequence.

4.1. Collection of signer or originator information

Signer or originator constraints are identified using the public keys to verify each SignedData, AuthEnvelopedData, or AuthenticatedData layer encountered in a CMS path. The public key value, public key certificate or public key identifier of each signer or originator are collected in a state variable named cms_public_keys. Constraints are determined by building and validating a certification path for each public key after the content type and attributes of the CMS-protected object have been identified.

4.1.1. Signature or MAC Verification

The signature or MAC generated by the originator MUST be verified. If signature or MAC verification fails, then the CMS path containing the signature or MAC MUST be rejected. Signature and MAC verification procedures are defined in [RFC3852][RFC5083]. The public key or public key certificate used to verify each signature or MAC in a CMS path is added to the cms_public_keys state variable for use in content type and constraint checking.

4.2. Collection of Attributes

Attributes are collected from all authenticated nodes in a CMS path. That is, attributes are not collected from content types that occur before the first SignedData, AuthEnvelopedData, or AuthenticatedData instance. Additionally, an application may specify a set of attributes that it has authenticated, perhaps from processing one or more content types that encapsulate a CMS-protected content. If the content is not a leaf node in a CMS path, and it contains attributes, then add the attributes to cms_effective_attributes. Leaf node attributes may be checked independent of the CMS content constraints certificate extension processing, but such processing is not addressed in this document.

4.3. Leaf node classification

The type of leaf node that terminates a CMS path determines the types of information that is returned and the type of processing that is performed. There are two types of leaf nodes: encrypted leaf nodes and payload leaf nodes.

A node in a CMS path is a leaf node if the content type of the node is not one of the following content types:

id-signedData (SignedData),

id-digestedData (DigestedData),

id-ct-authData (AuthenticatedData),

id-ct-compressedData (CompressedData),

id-ct-contentCollection (ContentCollection), and

id-ct-contentWithAttrs (ContentWithAttributes).

A leaf node is an encrypted leaf node if the content type of the node is one of the following content types:

id-encryptedData (EncryptedData),

id-envelopedData (EnvelopedData), and

id-ct-authEnvelopedData (AuthEnvelopedData).

All other leaf nodes are payload leaf nodes, since no further CMS encapsulation can occur beyond that node. However, specifications may define content types that provide protection similar to the CMS content types, may augment the lists of possible leaf nodes and encrypted leaf nodes or may define some encrypted types as payload leaf nodes.

When an encrypted leaf node is encountered, processing terminates and returns information that may be used as input when procesing the decrypted contents. Content type and constraints checking are only performed for payload leaf nodes. When an encrypted leaf node terminates a CMS path, the attributes collected in cms_effective_attributes are returned along with the public key information collected in cms_public_keys. When a payload leaf node terminates a CMS path, content type and constraint checking must be performed, as described in the next section.

4.4. Content Type and Constraint Checking

4.4.1. Inputs

The inputs to content type and constraint checking are the values collected in cms_public_keys and cms_effective_attributes from a CMS path along with the payload leaf node that terminates the CMS path.

4.4.2. Processing

When a payload leaf node is encountered in a CMS path and a signed or authenticated content type is present in the CMS path, content type and constraint checking MUST be performed. Content type and constraint checking need not be performed for CMS paths that do not contain at least one SignedData, AuthEnvelopedData, or AuthenticatedData content type. The cms_effective_attributes and cms_public_keys variables are used to perform constraint checking. Two additional state variables are used during the processing: cms_constraints and cms_default_attributes, both of which are initialized to empty. The steps required to perform content type and constraint checking are below.

For each public key in cms_public_keys, build and validate a certification path from a trust anchor to the public key, providing the content type of the payload leaf node and cms_effective_attributes as input.

> If path validation is successful, add the contents of subject_default_attributes to cms_default_attributes. The subject_constraints variable returned from certification path validation will contain a single entry. If the subject_constraints entry is equal to the special value anyContentType, content type and constraints checking succeeds. If the subject_constraints entry is not equal to the special value anyContentType, for each entry in the attrConstraints field of the entry in subject_constraints,
>
> > If there is an entry in cms_constraints with the same attrType value, add the value from the attrContraints entry to the entry in cms_constraints if that value does not already appear.
> >
> > If there is no entry in cms_constraints with the same attrType value, add a new entry to cms_constraints equal to the entry from the attrConstraints field.
>
> If the value of canSource field of the entry in the subject_constraints variable for the public key used to verify the signature or MAC closest to the payload leaf node is set to FALSE, constraints checking fails and the CMS path MUST be rejected.

If no valid certification path can be found, constraints checking fails and the CMS path MUST be rejected.

4.4.3. Outputs

When a payload leaf node is encountered and content type and constraint checking succeeds, return cms_constraints, cms_default_attributes and cms_effective_attributes for use in leaf node payload processing.

When an encrypted leaf node is encountered and constraint checking is not performed, return cms_public_keys and cms_effective_attributes for use in continued processing (as described in section 4.3.1).

The cms_effective_attributes list may contain multiple instances of the same attribute type or attributes with multiple values. Payload processing, which might take advantage of these effective attributes, needs to describe the proper handling of this situation. Payload processing is described in other documents, and it is expected to be specific to a particular content type.

The cms_default_attributes list may contain attributes with multiple values. Payload processing, which might take advantage of these default attributes, needs to describe the proper handling of this situation. Payload processing is described in other documents, and it is expected to be specific to a particular content type.

5. Security Considerations

The authorization model described in section 1 allows trust anchors with different privileges. Delegation is accomplished by issuing an X.509 certificate. If the trust anchor issues a certification authority (CA) certificate, then further delegation is permitted. If the trust anchor issues an end entity certificate, then further delegation is prohibited.

For any given certificate, multiple certification paths may exist, and each one can yield different results for CMS content constraints processing. To avoid creating unintended results, the impact of CMS content constraints included (or omitted) from cross-certificates must be evaluated.

CMS content constraints are not used with countersignatures.

Though not explicitly discussed in this document, CMS content constraints can be applied to CMS-protected contents featuring multiple parallel signers, for example where there is more than one SignerInfo, each carrying a signature from a different party, within a single SignedData content. In such cases, each SignerInfo must be processed as if it were the only SignerInfo, and the CMS content constraints must be met in order for that signature to be considered valid. Unlike signers represented in distinct SignedData contents, signers represented by multiple SignerInfos are not considered to be collaborating with regard to a particular content. Each parallel signer is evaluated independently; no relationship to the other signers in the set of SignerInfos implied. A content is considered valid only if there is at least one valid CMS path employing one SignerInfo within each SignedData content, even when more than one SignerInfo is present.

6. IANA Considerations

There are no IANA considerations. Please delete this section prior to RFC publication.

7. References

7.1. Normative References

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

[RFC3274] Gutmann, P., "Compressed Data Content Type for Cryptographic Message Syntax (CMS)", RFC 3274, June 2002.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC4073] Housley, R., "Protecting Multiple Contents with the Cryptographic Message Syntax (CMS)", RFC 4073, May 2005.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[X.680] "ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One", 1997.

[X.690] "ITU-T Recommendation X.690 Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", 1997.

7.2. Informative References

[PKIXASN1] Hoffman, P. and J. Schaad, "New ASN.1 Modules for PKIX", in progress.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, August 2005.

[RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, June 2008.

[X.208] "ITU-T Recommendation X.208 - Specification of Abstract Syntax Notation One (ASN.1)", 1988.

Appendix A. ASN.1 Modules

Appendix A.1 provides the normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680]. Appendix A.2 provides a module using ASN.1 as defined in [X.208]. The module in A.2 removes usage of newer ASN.1 features that provide support for limiting the types of elements that may appear in certain SEQUENCE and SET constructions. Otherwise, the modules are compatible in terms of encoded representation, i.e., the modules are bits-on-the-wire compatible aside from the limitations on SEQUENCE and SET constituents. A.2 is included as a courtesy to developers using ASN.1 compilers that do not support current ASN.1. A.1 references an ASN.1 module from [PKIXASN1].

A.1. ASN.1 Module Using 1993 Syntax

```
CMSContentConstraintsCertExtn-93
  { iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0) cmsContentConstr-93(42) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

IMPORTS
    ContentType
      FROM CryptographicMessageSyntax2004 -- from [RFC3852]
        { iso(1) member-body(2) us(840) rsadsi(113549)
          pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }
    AttributeType, AttributeValue
      FROM PKIX1Explicit88 -- from [RFC5280]
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-pkix1-explicit(18) }
    EXTENSION
      FROM PKIX-CommonTypes
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkixCommon(43) } ;

id-ct-anyContentType OBJECT IDENTIFIER ::=
    { iso(1) member-body(2)
      us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
      ct(1) 0 }

cmsContentConstraints EXTENSION ::= {
    SYNTAX         CMSContentConstraints
    IDENTIFIED BY  id-pe-cmsContentConstraints }
```

```
   id-pe-cmsContentConstraints OBJECT IDENTIFIER ::=
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) pe(1) 18 }

   CMSContentConstraints ::= ContentTypeConstraintList

   ContentTypeConstraintList ::= SEQUENCE SIZE (1..MAX) OF
                                 ContentTypeConstraint

   ContentTypeConstraint ::= SEQUENCE {
       contentType           ContentType,
       canSource             BOOLEAN DEFAULT TRUE,
       attrConstraints       AttrConstraintList OPTIONAL }

   AttrConstraintList ::= SEQUENCE SIZE (1..MAX) OF AttrConstraint

   AttrConstraint ::= SEQUENCE {
       attrType               AttributeType,
       attrValues             SET SIZE (1..MAX) OF AttributeValue }

   END
```

A.2. ASN.1 Module Using 1988 Syntax

```
    CMSContentConstraintsCertExtn-88
      { iso(1) identified-organization(3) dod(6) internet(1) security(5)
        mechanisms(5) pkix(7) id-mod(0) cmsContentConstr-88(41) }

    DEFINITIONS IMPLICIT TAGS ::=
    BEGIN

    IMPORTS
        ContentType
          FROM CryptographicMessageSyntax2004 -- from [RFC3852]
            { iso(1) member-body(2) us(840) rsadsi(113549)
              pkcs(1) pkcs-9(9) smime(16) modules(0) cms-2004(24) }
        AttributeType, AttributeValue
          FROM PKIX1Explicit88 -- from [RFC5280]
            { iso(1) identified-organization(3) dod(6) internet(1)
              security(5) mechanisms(5) pkix(7) id-mod(0)
              id-pkix1-explicit(18) } ;

    id-ct-anyContentType OBJECT IDENTIFIER ::=
        { iso(1) member-body(2)
          us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
          ct(1) 0}

    -- Extension object identifier

    id-pe-cmsContentConstraints OBJECT IDENTIFIER ::=
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) pe(1) 18 }

    -- CMS Content Constraints Certificate Extension

    CMSContentConstraints ::= ContentTypeConstraintList

    ContentTypeConstraintList ::= SEQUENCE SIZE (1..MAX) OF
                                ContentTypeConstraint

    ContentTypeConstraint ::= SEQUENCE {
        contentType           ContentType,
        canSource             BOOLEAN DEFAULT TRUE,
        attrConstraints       AttrConstraintList OPTIONAL }

    AttrConstraintList ::= SEQUENCE SIZE (1..MAX) OF AttrConstraint

    AttrConstraint ::= SEQUENCE {
        attrType               AttributeType,
        attrValues             SET SIZE (1..MAX) OF AttributeValue }

    END
```

Authors' Addresses

Russ Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170

Email: housley@vigilsec.com

Sam Ashmore
National Security Agency
Suite 6751
9800 Savage Road
Fort Meade, MD 20755

Email: srashmo@radium.ncsc.mil

Carl Wallace
Cygnacom Solutions
Suite 5200
7925 Jones Branch Drive
McLean, VA 22102

Email: cwallace@cygnacom.com

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.