KITTEN WORKING GROUP	N. Williams
Internet-Draft	Sun
Intended status: Standards Track	L. Johansson
Expires: August 13, 2011	SUNET
	February 09, 2011

GSS-API Naming Extensions
draft-ietf-kitten-gssapi-naming-exts-09

Abstract

The Generic Security Services API (GSS-API) provides a simple naming architecture that supports name-based authorization. This document introduces new APIs that extend the GSS-API naming model to support name attribute transfer between GSS-API peers.

Status of this Memo

Copyright Notice

Table of Contents

1. Conventions used in this document

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. Introduction

As described in [RFC4768] the GSS-API's naming architecture suffers from certain limitations. This document proposes concrete GSS-API extensions.

A number of extensions to the GSS-API [RFC2743] and its C Bindings [RFC2744] are described herein. The goal is to make information modeled as "name attributes" available to applications. Such information MAY for instance be used by applications to make authorization-decisions. For example, Kerberos V authorization data elements, both in their raw forms, as well as mapped to more useful value types, can be made available to GSS-API applications through these interfaces.

The model is that GSS names have attributes. The attributes of a name may be authenticated (eg an X509 attribute certificate or signed SAML attribute assertion), or may have been set on a GSS name for the purpose of locally "asserting" the attribute during credential acquisition or security context exchange. Name attributes' values are network representations thereof (e.g., the actual value octets of the contents of an X.509 certificate extension, for example) and are intended to be useful for constructing portable access control facilities. Applications may often require language- or platform-specific data types, rather than network representations of name attributes, so a function is provided to obtain objects of such types associated with names and name attributes.

3. Name Attribute Authenticity

An attribute is 'authenticated' iff there is a secure association between the attribute (and its values) and the trusted source of the peer credential. Examples of authenticated attributes are (any part of) the signed portion of an X.509 certificate or AD-KDCIssued authorization-data elements in Kerberos V Tickets provided of course that the authenticity of the respective security associations (eg signatures) have been verified.

Note that the fact that an attribute is authenticated does not imply anything about the semantics of the attribute nor that the trusted credential source was authorized to assert the attribute. Such interpretations SHOULD be the result of applying local policy to the attribute.

An un-authentciated attribute is called *asserted* in what follows.This is not to be confused with other uses of the word asserted or assertion eg "SAML attribute assertion", the attributes of which may be authenticated in the sense of this document for instance if the SAML attribute assertion was signed by a key trusted by the peer.

4. Name Attributes/Values as ACL Subjects

To facilitate the development of portable applications that make use of name attributes to construct and evaluate portable ACLs the GSS-API makes name attribute values available in canonical network encodings thereof.

5. Naming Contexts

Several factors influence the context in which a name attribute is interpreted. One is the trust context.
As discussed previously, applications apply local policy to determine whether a particular peer credential issuer is trusted to make a given statement. Different GSS-API mechanisms and deployments have different trust models surrounding attributes they provide about a name.
For example, Kerberos deployments in the enterprise typically trust a KDC to make any statement about principals in a realm. This includes attributes such as group membership.
In contrast, in a federated SAML environment, the identity provider typically exists in a different organization than the acceptor. In this case, the set of group memberships or entitlements that the IDP is permitted to make needs to be filtered by the policy of the acceptor and federation.
So even an attribute containing the same information such as e-mail address would need to be treated differently by the application in the context of an enterprise deployment from the context of a federation.
Another aspect related to trust is the role of the credential issuer in providing the attribute. Consider Kerberos PKINIT (RFC 4556). In this protocol, a public key and associated certificate are used to authenticate to a Kerberos KDC. Consider how attributes related to a pkinit certificate should be made available in GSS-API authentications based on the Kerberos ticket. In some deployments the certificate may be fully trusted; in including the certificate information in the ticket, the KDC permits the acceptor to trust the information in the certificate just as if the KDC itself had made these statements. In other deployments, the KDC may have authorized a hash of the certificate without evaluating the content of the certificate or generally trusting the issuing certificate authority. In this case, if the certificate were included in the issued ticket, the KDC would only be making the statement that the certificate was used in the authentication. This statement would be authenticated, but would not imply that the KDC stated particular attributes of the certificate described the initiator.

Another aspect of context is encoding of the attribute information. An attribute containing an ASCII or UTF-8 version of an e-mail address could not be interpreted the same as a ASN.1 Distinguished Encoding Rules e-mail address in a certificate.

All of these contextual aspects of a name attribute affect whether two attributes can be treated the same by an application and thus whether they should be considered the same name attribute. In the GSS-API naming extensions, attributes that have different contexts MUST have different names so they can be distinguished by applications. As an unfortunate consequence of this requirement, multiple attribute names will exist for the same basic information. That is, there is no single attribute name for the e-mail address of an initiator. Other aspects of how mechanisms describe information about subjects would already make this true. For example, some mechanisms use OIDs to name attributes; others use URIs.

Local implementations or platforms are likely to have sufficient policy and information to know when contexts can be treated as the same. For example the GSS-API implementation may know that a particular certificate authority can be trusted in the context of a pkinit authentication. The local implementation may have sufficient policy to know that a particular credential issuer is trusted to make a given statement. In order to take advantage of this local knowledge within the GSS-API implementation, naming extensions support the concept of local attributes in addition to standard attributes. For example, an implementation might provide a local attribute for e-mail address. The implementation would specify the encoding and representation of this attribute; mechanism-specific standards attributes would be re-encoded if necessary to meet this representation. Only e-mail addresses in contexts that meet the requirements of local policy would be mapped into this local attribute.

Such local attributes inherently expose a tradeoff between interoperability and usability. Using a local attribute in an application requires knowledge of the local implementation. However using a standardized attribute in an application requires more knowledge of policy and more validation logic in the application. Sharing this logic in the local platform provides more consistency across applications as well as reducing implementation costs. Both options are needed.

6. Representation of Attribute Names

Different underlying mechanisms provide different representations for the names of their attribute. In X.509 certificates, most objects are named by object identifiers (OIDs). The type of object (certificate extension, name constraint, keyPurposeID, etc) along with the OID is sufficient to identify the attribute. In contrast, according to Section 8.2 and 2.7.3.1 of [OASIS.saml-core-2.0-os], the name of an attribute has two parts. The first is a URI describing the format of the name. The second part, whose form depends on the format URI, is the actual name. In other cases an attribute might represent a certificate that plays some particular role in a GSS-API mechanism; such attributes might have a simple mechanism-defined name.

Attribute names MUST support multiple components. If there are more than one component in an attribute name, the more significant components define the semantics of the less significant components.

Attribute names are represented as STRING elements in the API described below. These attribute names have syntax and semantics that are understood by the application and by the lower-layer implementations (some of which are described below).

If an attribute name contains a space (ASCII 0x20), the first space separates the most significant or primary component of the name from the remainder. If there is no space, the primary component is the entire name, otherwise it defines the interpretation of the remainder of the name.s

If the primary component contains an ASCII : (0x3a), then the primary component is a URI. Otherwise, the attribute is a local attribute and the primary component has meaning to the implementation of GSS-API or to the specific configuration of the application. At this time, local attribute names are not standardized; there is debate about whether such standardization will be useful. Any future standardizations will need to balance potential problems resulting from attribute names used before standardization.

A sufficient prefix of attribute names needs to be dictated by a mechanism in order to describe the context. For example it would be problematic to represent SAML attribute names as the name format URI, a space, and the remainder of the name. A carefully crafted SAML assertion could appear to be a name from another mechanism or context. Typically a SAML attribute name would include a prefix describing the trust model and other context of the attribute name.

Local attribute names under the control of an administrator or a sufficiently trusted part of the platform need not have a prefix to describe context.

7. API

7.1. GSS_Display_name_ext()

Inputs:

- *name NAME,
- *display_as_name_type OBJECT IDENTIFIER

Outputs:

- *major_status INTEGER,

*minor_status INTEGER,

*display_name STRING

Return major_status codes:

*GSS_S_COMPLETE indicates no error.

*GSS_S_UNAVAILABLE indicates that the given name could not be displayed using the syntax of the given name type.

*GSS_S_FAILURE indicates a general error.

This function displays a given name using the given name syntax, if possible. This operation may require mapping MNs to generic name syntaxes or generic name syntaxes to mechanism-specific name syntaxes; such mappings may not always be feasible and MAY be inexact or lossy, therefore this function may fail.

7.1.1. C-Bindings

```
OM_uint32 GSS_Display_name_ext(
  OM_uint32                      *minor_status,
  gss_name_t                     name,
  gss_OID                        display_as_name_type,
  gss_buffer_t                   display_name
);
```

7.2. GSS_Inquire_name()

Inputs:

*name NAME

Outputs:

*major_status INTEGER,

*minor_status INTEGER,

*name_is_MN BOOLEAN,

*mn_mech OBJECT IDENTIFIER,

*attrs SET OF OCTET STRING

Return major_status codes:

*GSS_S_COMPLETE indicates no error.

*GSS_S_FAILURE indicates a general error.

This function outputs the set (represented as a NULL terminated array of gss_buffer_t) of attributes of a name. It also indicates if a given NAME is an MN or not and, if it is, what mechanism it's an MN of. The gss_buffer_set_t type and associated API is defined in [GFD.024]

7.2.1. C-Bindings

```
OM_uint32 gss_inquire_name(
  OM_uint32                     *minor_status,
  gss_name_t                    name,
  int                           name_is_MN,
  gss_OID                       *MN_mech,
  gss_buffer_set_t              *attrs
);
```

7.3. GSS_Get_name_attribute()

Inputs:

*name NAME,

*attr STRING

Outputs:

*major_status INTEGER,

*minor_status INTEGER,

*authenticated BOOLEAN, -- TRUE iff authenticated by the trusted peer credential source.

*complete BOOLEAN -- TRUE iff this represents a complete set of values for the name.

*values SET OF OCTET STRING,

*display_values SET OF STRING

Return major_status codes:

*GSS_S_COMPLETE indicates no error.

*GSS_S_UNAVAILABLE indicates that the given attribute OID is not known or set.

*GSS_S_FAILURE indicates a general error.

This function outputs the value(s) associated with a given GSS name object for a given name attribute.
The complete flag denotes that (if TRUE) the set of values represents a complete set of values for this name. The peer being an authoritative source of information for this attribute is a sufficient condition for the complete flag to be set by the peer.
In the federated case when several peers may hold some of the attributes about a name this flag may be highly dangerous and SHOULD NOT be used.
NOTE: This function relies on the GSS-API notion of "SET OF" allowing for order preservation; this has been discussed on the KITTEN WG mailing list and the consensus seems to be that, indeed, that was always the intention. It should be noted however that the order presented does not always reflect an underlying order of the mechanism specific source of the attribute values.

7.3.1. C-Bindings

The C-bindings of GSS_Get_name_attribute() requires one function call per-attribute value, for multi-valued name attributes. This is done by using a single gss_buffer_t for each value and an input/output integer parameter to distinguish initial and subsequent calls and to indicate when all values have been obtained.
The 'more' input/output parameter should point to an integer variable whose value, on first call to gss_name_attribute_get() MUST be -1, and whose value upon function call return will be non-zero to indicate that additional values remain, or zero to indicate that no values remain.
The caller should not modify this parameter after the initial call. The status of the complete and authenticated flags MUST NOT change between multiple calls to iterate over values for an attribute.

```
OM_uint32 gss_get_name_attribute(
  OM_uint32                     *minor_status,
  gss_name_t                    name,
  gss_buffer_t                  attr,
  int                           *authenticated,
  int                           *complete,
  gss_buffer_t                  value,
  gss_buffer_t                  display_value,
  int                           *more
);
```

7.4. GSS_Set_name_attribute()

Inputs:

- *name NAME,
- *complete BOOLEAN, -- TRUE iff this represents a complete set of values for the name.
- *attr STRING,
- *values SET OF OCTET STRING

Outputs:

- *major_status INTEGER,
- *minor_status INTEGER

Return major_status codes:

- *GSS_S_COMPLETE indicates no error.
- *GSS_S_UNAVAILABLE indicates that the given attribute OID is not known or could not be set.
- *GSS_S_FAILURE indicates a general error.

The complete flag denotes that (if TRUE) the set of values represents a complete set of values for this name. The peer being an authoritative source of information for this attribute is a sufficient condition for the complete flag to be set by the peer.
In the federated case when several peers may hold some of the attributes about a name this flag may be highly dangerous and SHOULD NOT be used.
NOTE: This function relies on the GSS-API notion of "SET OF" allowing for order preservation; this has been discussed on the KITTEN WG mailing list and the consensus seems to be that, indeed, that was always the intention. It should be noted that underlying mechanisms may not respect the given order.

7.4.1. C-Bindings

The C-bindings of GSS_Set_name_attribute() requires one function call per-attribute value, for multi-valued name attributes -- each call adds one value. To replace an attribute's every value delete the attribute's values first with GSS_Delete_name_attribute().

```
OM_uint32 gss_set_name_attribute(
  OM_uint32                     *minor_status,
  gss_name_t                    name,
  int                           complete,
  gss_buffer_t                  attr,
  gss_buffer_t                  value
);
```

7.5. **GSS_Delete_name_attribute()**

Inputs:

- *name NAME,
- *attr STRING,

Outputs:

- *major_status INTEGER,
- *minor_status INTEGER

Return major_status codes:

- *GSS_S_COMPLETE indicates no error.
- *GSS_S_UNAVAILABLE indicates that the given attribute OID is not known.
- *GSS_S_UNAUTHORIZED indicates that a forbidden delete operation was attempted eg deleting a negative attribute.
- *GSS_S_FAILURE indicates a general error.

Deletion of negative authenticated attributes from NAME objects MUST NOT be allowed and must result in a GSS_S_UNAUTHORIZED.

7.5.1. **C-Bindings**

```
OM_uint32 gss_delete_name_attribute(
  OM_uint32                     *minor_status,
  gss_name_t                    name,
  gss_buffer_t                  attr
);
```

7.6. GSS_Export_name_composite()

Inputs:

*name NAME

Outputs:

*major_status INTEGER,

*minor_status INTEGER,

*exp_composite_name OCTET STRING

Return major_status codes:

*GSS_S_COMPLETE indicates no error.

*GSS_S_FAILURE indicates a general error.

This function outputs a token which can be imported with GSS_Import_name(), using GSS_C_NT_COMPOSITE_EXPORT as the name type and which preserves any name attribute information associated with the input name (which GSS_Export_name() may well not). The token format is no specified here as this facility is intended for inter-process communication only; however, all such tokens MUST start with a two-octet token ID, hex 04 02, in network byte order.
The OID for GSS_C_NT_COMPOSITE_EXPORT is <TBD>.

7.6.1. C-Bindings

```
OM_uint32 gss_export_name_composite(
  OM_uint32                      *minor_status,
  gss_name_t                     name,
  gss_buffer_t                   exp_composite_name
);
```

8. IANA Considerations

This document creates a namespace of GSS-API name attributes. Attributes are named by URIs, so no single authority is technically needed for allocation. However future deployment experience may indicate the need for an IANA registry for URIs used to reference names specified by IETF standards. It is expected that this will be a registry of URNs but this document provides no further guidance on this registry.

9. Security Considerations

This document extends the GSS-API naming model to include support for name attributes. The intention is that name attributes are to be used as a basis for (among other things) authorization decisions or personalization for applications relying on GSS-API security contexts. The security of the application may be critically dependent on the security of the attributes. This document classifies attributes as asserted or authenticated. Asserted (non-authenticated) attributes MUST NOT be used if the attribute has security implications for the application (eg authorization decisions) since asserted attributes may easily be controlled by the peer directly.
It is important to understand the meaning of 'authenticated' in this setting. Authenticated does not imply that any semantic of the attribute is claimed to be true. The only implication is that a trusted third party has asserted the attribute as opposed to the attribute being asserte by the peer itself. Any additional semantics is always the result of applying policy. For instance in a given deployment the mail attribute of the subject may be authenticated and sourced from an email system where 'authoritive' values are kept. In another situations users may be allowed to modify their mail addresses freely. In both cases the 'mail' attribute may be authenticated by virtue of being included in signed SAML attribute assertions or by other means authenticated by the underlying mechanism.
When the underlying security mechanism does not provide a permanent unique identity (eg anonymous kerberos) the GSS-API naming extensions may be used to provide a replacement permanent unique identity attribute which in this case may be unique for each peer party. This is analogous to the SAML permanentIdentifier attribute and has comparable security and privacy properties and implications.

10. References

10.1. Normative References

[RFC2025]	Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2743]	Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
[RFC2744]	Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, January 2000.
[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.
[RFC4120]	

	Neuman, C., Yu, T., Hartman, S. and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
[GFD.024]	Argonne National LaboratoryNational Center for Supercomputing ApplicationsArgonne National LaboratoryArgonne National Laboratory, "GSS-API Extensions", GFD GFD.024, June 2004.

10.2. Informative References

[RFC4768]	Hartman, S., "Desired Enhancements to Generic Security Services Application Program Interface (GSS-API) Version 3 Naming", RFC 4768, December 2006.
[RFC3061]	Mealling, M., "A URN Namespace of Object Identifiers", RFC 3061, February 2001.
[OASIS.saml-core-2.0-os]	Cantor, S., Kemp, J., Philpott, R. and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

Authors' Addresses

Nicolas Williams Williams Sun Microsystems 5300 Riata Trace Ct Austin, TX 78727 US EMail: Nicolas.Williams@sun.com

Leif Johansson Johansson Swedish University Network Thulegatan 11 Stockholm, Sweden EMail: leifj@sunet.se URI: http://www.sunet.se