NETWORK WORKING GROUP
Internet-Draft
Expires: December 28, 2006

N. Williams
Sun

June 26, 2006

Stackable Generic Security Service Pseudo-Mechanisms
draft-ietf-kitten-stackable-pseudo-mechs-02.txt

Status of this Memo

Copyright Notice

Abstract

This document defines and formalizes the concept of stackable pseudo-mechanisms, and associated concept of composite mechanisms, for the Generic Security Service Application Programming Interface (GSS-API), as well as several utility functions.

Stackable GSS-API pseudo-mechanisms allow for the composition of new mechanisms that combine features from multiple mechanisms. Stackable mechanisms that add support for Perfect Forward Security (PFS), data compression, additional authentication factors, etc... are

facilitated by this document.

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

Recent discussions within the IETF have shown the need for a refactoring of the features that GSS-API mechanisms may provide and a way to compose new mechanisms from smaller components.

One way to do this is to "stack" multiple mechanisms on top of each other such that the features of all of them are summed into a new, composite mechanism.

One existing GSS-API mechanism, LIPKEY [LIPKEY], is essentially stacked over another, SPKM-3 [LIPKEY] (although LIPKEY does not conform to the stackable pseduo-mechanism framework described herein).

The first truly stackable pseudo-mechanism proposed, CCM [CCM], is intended for signalling, during negotiation of mechanisms, the willingness of an initiator and/or acceptor to utilize channel bindings

Since then other similar mechanism compositing needs and ideas have come up, along with problems such as "what combinations are possible, useful, reasonable and secure?" This document addresses those problems. It introduces the concepts of stackable pseudo-mechanisms, composite mechanisms and mechanism features or attributes, as well as new inquiry and related interfaces to help in the mechanism compositing.

(Mechanism features are more formally referred to as "mechanism attributes" below. The terms "feature" and mechanism attribute" are sometimes used interchangeably.)

2.1. Glossary

Concrete GSS-API mechanism
: A mechanism which can be used standalone. Examples include: the Kerberos V mechanism [CFX], SPKM-1/2 [SPKM] and SPKM-3 [LIPKEY].

GSS-API Pseudo-mechanism
: A mechanism which uses other mechanisms in the construction of its context and/or per-message tokens and security contexts. SPNEGO

is an example of this.

Stackable GSS-API pseudo-mechanism
A mechanism which uses a single other mechanism in the construction of its tokens such that the OID of the composite result can be constructed by prepending the OID of the stackable pseudo-mechanism to the OID of the mechanism to be used by it.

Mechanism-negotiation GSS-API pseudo-mechanism
A GSS-API mechanism that negotiates the use of GSS-API mechanisms. SPNEGO [SPNEGO] is an example of this.

3. Mechanism Composition Issues

Interfacing with composite mechanisms through the existing GSS-API interfaces and the handling of composite mechanism tokens is straightforward enough and described in Section 4.

However, the concepts of stackable and composite mechanisms do give rise to several minor problems:

- How to determine allowable combinations of mechanisms;
- How to encode composite mechanism OIDs;
- How to decompose the OID of a composite mechanism and process its tokens properly;
- Application interfacing issues such as:
  - Whether and/or which composite mechanisms should be listed by GSS_Indicate_mechs();
  - Whether and/or which composite mechanisms not listed by GSS_Indicate_mechs() may nonetheless be available for use by applications and how applications can detect their availability;
  - What additional, if any, interfaces should be provided to help applications select appropriate mechanisms;
-

Mechanism negotiation issues (related to the application interface issues listed above), such as: vspace blankLines='1'/>
- Should applications advertise composite mechanisms in SPNEGO or other application-specific mechanism negotiation contexts?
- Or should applications implicitly advertise composite mechanisms by advertising concrete and stackable pseudo-mechanisms in SPNEGO or other application-specific mechanism negotiation contexts?

Section 4 addresses the OID composition, decomposition and encoding

issues, as well as basic interfacing and token handling issues.

Section 5 addresses interfacing issues more generally through the specification of additional, optional APIs.

Section 6 addresses mechanism negotiation issues.

4. Mechanism Composition

Mechanism composition by stacking pseudo-mechanisms on a concrete mechanism is conceptually simple: join the OIDs of the several mechanisms in question and process GSS-API tokens and routine calls through the top-most pseudo-mechanism in a stack, which can then, if necessary, recursively call the GSS-API to process any tokens for the remainder of the stack.

Some stackable pseudo-mechanisms may do nothing more than perform transformations on application data (e.g., compression); such pseudo-mechanisms will generally chain the processing of tokens and routine calls to the mechanisms below them in the stack.

Other stackable pseudo-mechanisms may utilize the mechanisms below them only during security context setup. For example, a stackable pseudo-mechanism could perform a Diffie-Hellman key exchange and authenticate it by binding a security context established with the mechanism stacked below it; such a mechanism would provide its own per-message tokens.

4.1. Construction of Composed Mechanism OIDs

Composition of mechanism OIDs is simple: prepend the OID of one pseudo-mechanism to the OID of another mechanism (composite or otherwise), but there MUST always be at least one final mechanism OID and it MUST be useful standalone (i.e., it MUST NOT be a pseudo-mechanism). A composite mechanism OID forms, essentially, a stack.

The encoding of composed mechanism OIDs is not quite the concatenation of the component OIDs' encodings, however. This is because the first two arcs of ASN.1 OIDs are encoded differently from subsequent arcs (the first two arcs have a limited namespace and are encoded as a single octet), so were composite mechanism OIDs to be encoded as the concatenation of the component OIDs the result would not decode as the concatenation of the component OIDs. To avoid this problem the first two arcs of each component of a composite mechanism OID, other than the leading component, will be encoded as other arcs would.

Decomposition of composite mechanism OIDs is similar, with each pseudo-mechanism in the stack being able to determine the OID suffix from knowledge of its own OID(s).

New pseudo-mechanisms MAY be allocated OIDs from the prefix given below as follows by assignment of a sub-string of OID arcs to be appended to this prefix. This prefix OID is:

<TBD> [1.3.6.1.5.5.11 appears to be available, registration w/ IANA TBD]

All OID allocations below this OID MUST be for stackable pseudo-mechanisms and MUST consist of a single arc. This will make it possible to decompose the OIDs of composite mechanisms without necessarily knowing a priori the OIDs of the component stackable pseudo-mechanisms.

4.2. Mechanism Composition Rules

All new stackable pseudo-mechanisms MUST specify the rules for determining whether they can stack above a given mechanism, composite or otherwise. Such rules may be based on specific mechanism attribute OID sets [EXTENDED-INQUIRY] and/or specific mechanism OIDs (composite and otherwise).

All stackable pseudo-mechanisms MUST have the following mechanism composition rule relating to unknown mechanism attributes:

- composition with mechanisms supporting unknown mechanism attributes MUST NOT be permitted.

This rule protects against compositions which cannot be considered today but which might nonetheless arise due to the introduction of new mechanisms and which might turn out to be insecure or otherwise undesirable.

Mechanism composition rules for stackable pseudo-mechanisms MAY and SHOULD be updated as new GSS-API mechanism attributes and mechanisms sporting them are introduced. The specifications of mechanisms that introduce new mechanism attributes or which otherwise should not be combined with others in ways which would be permitted under existing rules SHOULD also update the mechanism composition rules of affected pseudo-mechanisms.

A RECOMMENDED way to describe the stacking rules for stackable mechanisms is as an ordered sequence of "MAY stack above X mechanism," "REQUIRES Y mechanism feature(s)," "MUST NOT stack above Z mechanism," and/or "MUST NOT stack above a mechanism with Z

mechanism feature(s)."

For example a stackable mechanism that provides its own per-msg tokens and does not use the underlying mechnism's per-msg token facilities might require a rule such as "MUST NOT stack above a mechanism with the GSS_C_MA_COMPRESS mechanism feature."

4.3. Interfacing with Composite Mechanisms

The basic GSS-API [RFC2743] interfaces MUST NOT accept as input or provide as output the OID of any stackable pseudo-mechanism. Composite mechanisms MUST be treated as concrete mechanisms by the basic GSS-API interfaces [RFC2743].

Thus the way in which a composite mechanism is used by applications with the basic GSS-API (version 2, update 1) is straightforward: exactly as if composite mechanisms were normal GSS-API mechanisms.

This is facilitated by the fact that in all cases where the GSS-API implementation might need to know how to process or create a token it has the necessary contextual information, that is, the mechanism OID, available and can decompose composite mechanism OIDs as necessary.

For example, for initial GSS_Init_sec_context() calls the implementation knows the desired mechanism OID, and if it should be left unspecified, it can pick a default mechanism given the initiator credentials provided by the application (and if none are provided other default mechanism and credential selections can still be made). For subsequent calls to GSS_Init_sec_context() the implementation knows which mechanism to use from the given [partially established] security context. Similarly for GSS_Accept_sec_context, where on initial calls the mechanism OID can be determined from the given initial context token's framing.

The manner in which GSS-API implementations and the various mechanisms and pseudo-mechanisms interface with one another is left as an excercise to implementors.

4.4. Compatibility with the Basic GSS-APIv2u1 Interfaces

In order to preserve backwards compatibility with applications that use only the basic GSS-API interfaces (version 2, update 1), several restrictions are imposed on the use of composite and stackable pseduo-mechanisms with the basic GSS-API interfaces:

- GSS_Indicate_mechs() MUST NOT indicate support for any stackable pseduo-mechanisms under any circumstance.
- GSS_Indicate_mechs() MAY indicate support for some, all or none of

the available composite mechanisms.

- Which composite mechanisms, if any, are indicated through GSS_Indicate_mechs() SHOULD be configurable.
- Composite mechanisms which are not indicated by GSS_Indicate_mechs() MUST NOT be considered as the default mechanism (GSS_C_NULL_OID) or as part of the default mechanism set (GSS_C_NULL_OID_SET).
- The OIDs of *stackable* (not composite) pseudo-mechanisms MUST NOT be accepted as inputs or produced in the output of any of the basic GSS-APIv2, update 1 API functions, except for any OID set construction/iteration functions. And, if present in any OID SET input parameters of GSS-APIv2, update 1 functions, they MUST be ignored.
- The OIDs of *stackable* (not composite) pseudo-mechanisms MAY only be used as inputs or produced as outputs of functions whose specification explicitly allows for them or which are concerned with the creation/iteration of OID containters, such as OID SETs.

4.5. Processing of Tokens for Composite Mechanisms

The initial context token for any standard mechanism, including mechanisms composited from standard pseudo- and concrete mechanisms, MUST be encapsulated as described in section 3.1 of rfc2743 [RFC2743], and the OID used in that framing MUST be that of the mechanism, but in the case of composite mechanisms this OID MUST be the OID of the leading component of the composite mechanism.

Note that this has implications for pluggable multi-mechanism implementations of the GSS-API, namely that acceptors must route initial context tokens to the appropriate mechanism and they must allow that mechanism to determine the composite mechanism OID (such as by allowing that mechanism's GSS_Accept_sec_context() to output the actual mechanism to the application.

In all other cases the mechanism that produced or is to produce a given token can be determined internally through the given security context.

5. New GSS-API Interfaces

...

Utility functions for mechanism OID composition and decomposition are given in sections 5.1.1, 5.1.2 and 5.1.3.

Two utility functions, GSS_Indicate_negotiable_mechs() and GSS_Negotiate_mechs(), to aid applications in mechanism negotiation

are described in sections 5.1.4 and 5.1.5. These two interfaces may be implemented entirely in terms of the other interfaces described herein.

5.1. New GSS-API Function Interfaces

Several new interfaces are given by which, for example, GSS-API applications may determine what features are provided by a given mechanism, what mechanisms provide what features and what compositions are legal.

These new interfaces are all OPTIONAL.

In order to preserve backwards compatibility with applications that do not use the new interfaces GSS_Indicate_mechs() MUST NOT indicate support for any stackable pseduo-mechanisms. GSS_Indicate_mechs() MAY indicate support for some, all or none of the available composite mechanisms; which composite mechanisms, if any, are indicated through GSS_Indicate_mechs() SHOULD be configurable. GSS_Acquire_cred() and GSS_Add_cred() MUST NOT create credentials for composite mechanisms not explicitly requested or, if no desired mechanism or mechanisms are given, for composite mechanisms not indicated by GSS_Indicate_mechs().

Applications SHOULD use GSS_Indicate_mechs_by_mech_attrs() instead of GSS_Indicate_mechs() wherever possible.

Applications can use GSS_Indicate_mechs_by_mech_attrs() to determine what, if any, mechanisms provide a given set of features.

GSS_Indicate_mechs_by_mech_attrs() can also be used to indicate (as in GSS_Indicate_mechs()) the set of available mechanisms of each type (concrete, mechanism negotiation pseudo-mechanism, stackable pseudo-mechanism and composite mechanisms).

Applications may use GSS_Inquire_mech_attrs_for_mech() to test whether a given composite mechanism is available and the set of features that it offers.

GSS_Negotiate_mechs() may be used to negotiate the use of mechanisms such that composite mechanisms need not be advertised but instead be implied by offering stackable pseudo-mechanisms.

5.1.1. GSS_Compose_oid()

Inputs:
- mech1 OBJECT IDENTIFIER, -- mechanism OID
- mech2 OBJECT IDENTIFIER -- mechanism OID

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- composite OBJECT IDENTIFIER -- OID composition of mech1 with mech2 ({mech1 mech2})

Return major_status codes:
- GSS_S_COMPLETE indicates success.
- GSS_S_BAD_MECH indicates that mech1 is not supported.
- GSS_S_FAILURE indicates that the request failed for some other reason. The minor status will be specific to mech1 and may provide further information.

5.1.2. GSS_Decompose_oid()

Inputs:
- input_mech OBJECT IDENTIFIER, -- mechanism OID.
- mechs SET OF OBJECT IDENTIFIER -- mechanism OIDs (if GSS_C_NULL_OID_SET defaults to the set of stackable pseudo-mechanism OIDs indicated by GSS_Indicate_mechs_by_mech_attrs()).

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- lead_mech OBJECT IDENTIFIER, -- leading stackable pseudo-mechanism OID.
- trail_mech OBJECT IDENTIFIER -- input_mech with lead_mech removed from the front.

Return major_status codes:
- GSS_S_COMPLETE indicates success.
- GSS_S_BAD_MECH indicates that the input_mech could not be decomposed as no stackable pseudo-mechanism is available whose OID is a prefix of the input_mech.
- GSS_S_FAILURE indicates that the request failed for some other reason.

5.1.3. GSS_Release_oid()

The following text is adapted from the obsoleted rfc2078 [RFC2078].

Inputs:
- oid OBJECT IDENTIFIER

Outputs:
- major_status INTEGER,
- minor_status INTEGER

Return major_status codes:
- GSS_S_COMPLETE indicates successful completion
- GSS_S_FAILURE indicates that the operation failed

Allows the caller to release the storage associated with an OBJECT IDENTIFIER buffer allocated by another GSS-API call, specifically GSS_Compose_oid() and GSS_Decompose_oid(). This call's specific behavior depends on the language and programming environment within which a GSS-API implementation operates, and is therefore detailed within applicable bindings specifications; in particular, this call may be superfluous within bindings where memory management is automatic.

5.1.4. GSS_Indicate_negotiable_mechs()

Inputs:
- input_cred_handle CREDENTIAL HANDLE, -- credential handle to be used with GSS_Init_sec_context(); may be GSS_C_NO_CREDENTIAL.
- peer_type_known BOOLEAN, -- indicates whether the peer is known to support or not supprot the stackable pseudo-mechanism framework.
- peer_has_mech_stacking BOOLEAN -- indicates whether the peer supports the stackable pseudo-mechanism framework; ignore if peer_type_known is FALSE.

Outputs:
- major_status INTEGER,
- minor_status INTEGER,
- offer_mechs SET OF OBJECT IDENTIFIER, -- mechanisms to offer.

Return major_status codes:
- GSS_S_COMPLETE indicates success.
- GSS_S_NO_CREDENTIAL indicates that the caller's credentials are expired or, if input_cred_handle is GSS_C_NO_CREDENTIAL, that no credentials could be acquired for GSS_C_NO_NAME.
- GSS_S_FAILURE indicates that the request failed for some other reason.

This function produces a set of mechanism OIDs, optimized for space, that its caller should advertise to peers during mechanism negotiation.

The output offer_mechs parameter will include all of the mechanisms for which the input_cred_handle has elements (as indicated by GSS_Inquire_cred()), but composite mechanisms will be included either implicitly or implicitly as per the following rules:
- if peer_type_known is TRUE and peer_has_mech_stacking is FALSE then no composite mechanisms not indicated by GSS_Indicate_mechs() will be advertised, explictly or implicitly;

- o if peer_type_known is FALSE then all composite mechanisms indicated by GSS_Indicate_mechs() for which input_cred_handle has elements will be indicated in offer_mechs explicitly and all others may be indicated in offer_mechs implicitly, by including their component stackable pseduo-mechanism OIDs (see below);
- o if peer_type_known is TRUE and peer_has_mech_stacking is TRUE composite mechanisms will generally not be advertised explicitly, but will be advertised implicitly, by including their component stackable pseduo-mechanism OIDs (see below); no composite mechanisms will be advertised explicitly
- o if the input_cred_handle does not have elements for all of the possible composite mechanisms that could be constructed from the its elements' decomposed mechanisms, then all composite mechanisms for which the input_cred_handle does have elements will be advertised explicitly in offer_mechs.

5.1.5. GSS_Negotiate_mechs()

Inputs:
- o input_credential_handle CREDENTIAL HANDLE, -- mechanisms offered by the caller.
- o peer_mechs SET OF OBJECT IDENTIFIER -- mechanisms offered by the caller's peer.

Outputs:
- o major_status INTEGER,
- o minor_status INTEGER,
- o mechs SET OF OBJECT IDENTIFIER -- mechanisms common to the caller's credentials and the caller's peer.

Return major_status codes:
- o GSS_S_COMPLETE indicates success; the output mechs parameter MAY be the empty set (GSS_C_NO_OID_SET).
- o GSS_S_NO_CREDENTIAL indicates that the caller's credentials are expired or, if input_cred_handle is GSS_C_NO_CREDENTIAL, that no credentials could be acquired for GSS_C_NO_NAME.
- o GSS_S_FAILURE indicates that the request failed for some other reason.

This function matches the mechanisms for which the caller has credentials with the mechanisms offered by the caller's peer and returns the set of mechanisms in common to both, accounting for any composite mechanisms offered by the peer implicitly.

5.1.6. C-Bindings

```
OM_uint32 gss_compose_oid(
```

```
        OM_uint32         *minor_status,
        const gss_OID      mech1,
        const gss_OID      mech2,
        gss_OID           *composite);

    OM_uint32 gss_decompose_oid(
        OM_uint32         *minor_status,
        const gss_OID      input_mech,
        const gss_OID_set  mechs,
        gss_OID           *lead_mech,
        gss_OID           *trail_mech);

    OM_uint32 gss_release_oid(
        OM_uint32         *minor_status,
        gss_OID           *oid);

    OM_uint32 gss_indicate_negotiable_mechs(
        OM_uint32                 *minor_status,
        const gss_cred_id_t        input_cred_handle,
        OM_uint32                  peer_type_known,
        OM_uint32                  peer_has_mech_stacking,
        gss_OID_set               *offer_mechs);

    OM_uint32 gss_negotiate_mechs(
        OM_uint32                 *minor_status,
        const gss_cred_id_t        input_cred_handle,
        const gss_OID_set          peer_mechs,
        const gss_OID_set         *mechs);
```

Figure 1

6. Negotiation of Composite Mechanisms

Where GSS-API implementations do not support the stackable mechanism framework interfaces applications may only negotiate explicitly from a set of concrete and composite mechanism OIDs as indicated by GSS_Indicate_mechs() and for which suitable credentials are available. GSS_Indicate_mechs(), as described in Section 4.4, MUST NOT indicate support for individual stackable pseudo-mechanisms, so there will not be any composite mechanisms implied but not explicitly offered in the mechanism negotiation.

Applications that support the stackable mechanism framework SHOULD use GSS_Indicate_negotiable_mechs() to construct the set of mechanism OIDs to offer to their peers. GSS_Indicate_negotiable_mechs() optimizes for bandwidth consumption by using decomposed OIDs instead

of composed OIDs, where possible. See Section 5.1.4.

Peers that support the stackable mechanism framework interfaces SHOULD use GSS_Negotiate_mechs() to select a mechanism as that routine accounts for composite mechanisms implicit in the mechanism offers.

6.1. Negotiation of Composite Mechanisms Through SPNEGO

SPNEGO applications MUST advertise either the set of mechanism OIDs for which they have suitable credentials or the set of mechanism OIDs produced by calling GSS_Indicate_negotiable_mechs() with the available credentials and the peer_type_known parameter as FALSE.

7. Requirements for Mechanism Designers

Stackable pseudo-mechanisms specifications MUST:
- o list the set of GSS-API mechanism attributes associated with them
- o list their initial mechanism composition rules
- o specify a mechanism for updating their mechanism composition rules

All other mechanism specifications MUST:
- o list the set of GSS-API mechanism attributes associated with them

8. IANA Considerations

Allocation of arcs in the namespace of OIDs relative to the base stackable pseduo-mechanism OID specified in Section 4.1 is reserved to the IANA.

9. Security considerations

Some composite mechanisms may well not be secure. The mechanism composition rules of pseudo-mechanisms (including the default composition rule given in Section 4 for unknown mechanism attributes) should be used to prevent the use of unsafe composite mechanisms.

Designers of pseudo-mechanisms should study the possible combinations of their mechanisms with others and design mechanism composition rules accordingly.

Similarly, pseudo-mechanism designers MUST specify, and implementors MUST implement, composite mechanism attribute set determination rules appropriate to the subject pseduo-mechanism, as described in section 4.2. Failure to do so may lead to inappropriate composite mechanisms

being deemed permissible by programmatic application of flawed mechanism composition rules or to by their application with incorrect mechanism attribute sets.

10. Normative

[EXTENDED-INQUIRY] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", draft-ietf-kitten-extended-mech-inquiry-00.txt (work in progress).

[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.

Author's Address

Nicolas Williams
Sun Microsystems
5300 Riata Trace Ct
Austin, TX 78727
US

Email: Nicolas.Williams@sun.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.