Internet-Draft D. Barton (CRC for DST)
<draft-barton-gss-api-sec-party-00.txt> 22 February 1996

Multi-party Security Contexts Within the GSS-API

STATUS OF THIS MEMO

ABSTRACT

The Generic Security Service Application Program Interface (GSS-API), as defined in RFC-1508 and subsequent Internet Draft documents, provides security services to callers (applications) in a generic fashion, supportable with a range of underlying mechanisms and technologies, allowing source-level portability of applications to different environments.

Multi-party security contexts, where security context information is shared amongst more than two session peers, have received little consideration within the scope of the GSS-API. This document describes details of one approach for supporting multi-party security services within the GSS-API.

1. Introduction

Multi-party security contexts have been overlooked in the past, possibly due in part to the emphasis on the client-server paradigm where only two parties need be involved in the context. There is a growing need for the ability to seamlessly include multiple parties in a single security context. This draft defines a number of simple modifications and additions to the GSS-API enabling multi-party context support.

2. Multi-party security

In some situations it is desirable to share security context information between a group of parties (entities, individuals, peers, what have you). For example, multi-object bindings, secure conversations or even electronic contract negotiation may require multi-party security contexts for authentication, confidentiality, integrity, and other security services.

The existing specifications for the GSS-API provide no support for multi-party contexts. Separate contexts must be set up between each pair of parties, simulating multi-party contexts. This approach requires N(N-1)/2 contexts, where N is the number of parties. As N increases the management complexity and overheads increase to unmanagable levels. This is a limiting factor in scaling this "solution". The method proposed within this document provides a single context handle for each multi-party context, simplifying the management required by the application, and maintaining the paradigm of the GSS-API.

There are two methods of establishing a multi-party security context. Firstly, initiating a context directly with all of the parties via a single call, and secondly, establishing a two or more party context and subsequently adding parties to the context over time. These approaches are respectively called "static" and "dynamic". Definitions for calls supporting both approaches will be provided in the following sections, with discussion of each call's behaviour, input and output parameters.

Throughout the following discussion the term "party" will be used to refer to a participating, authenticated entity. The GSS-API terms these entities as "peers" or "principals", and other systems may use other names. The term "party" is intended to be interchangeable with any of these terms.

2.1 Static Multi-party Contexts

Static multi-party context establishment occurs via a single call interface, yielding a handle to context data which, upon successful completion of the initialisation process, is shared by all participating parties. This behaviour is a generalisation of the standard GSS-API context initialisation interface enables context initialisation between two parties. The interface for this call is based largely on the gss_init_sec_context() call.

Only one call is required for static multi-party context initiation. Without the additional extensions defined for dynamic security contexts, the group of parties involved in the context cannot change, hence the term "static".

A new call for the GSS-API is defined which enables multi-party context initialisation. This call is named gss_init_group_sec_context() and has the following interface characteristics:

Inputs:

- name_list NAME_LIST,
- other inputs as per gss_init_sec_context().

Outputs:

- all outputs correspond to gss_init_sec_context().

This call processes the input parameter, name_list, and attempts to initialise a security context shared with each nominated party. Network interactions (where necessary) may cause this call to block as is the case with gss_init_sec_context().

Return major_status codes:

- GSS_S_GROUP_UNSUPPORTED indicates that the underlying mechanism has no support for multi-party contexts.
- others as per gss_init_sec_context().

This call behaves similarly in all ways to the gss_init_sec_context() call. The initiator calls this function, supplying a list of names for inclusion in the context. A token is returned which must be passed to all named parties for context acceptance. Multi-phase token exchange (continuation) may occur between the initiator and acceptors if required.

2.2 Dynamic Multi-party Contexts

Dynamic multi-party contexts are somewhat more complicated than standard two party or static multi-party contexts due to extra management and maintenance requirements. Addition of new parties requires access control and a process by which a party external to the context may gain entry to the context. Context data must be maintained consistently across the group of parties in the context, and parties should have a simple way of determining the list of parties with whom they are sharing the context. Specifications are given below for the interface characteristics of three calls supporting some of the services mentioned previously; gss_add_party(), gss_vote() and gss_context_parties() respectively.

The gss_add_party() call effectively performs a context initiation with the nominated party using existing context data. This call will therefore behave similarly to the gss_init_sec_context() call in many ways. Continuation may occur in the intialisation phase, and multi-phase token exchange may be required for context data initialisation (sharing). The gss_add_party() call is intended to yield tokens which can be fed as input directly into the gss_accept_sec_context() call by the acceptor, rather than requiring a separate API call for accepting addition to a context.

The gss_add_party() call:

Inputs:

- new_member_name INTERNAL NAME specifies the name of the new member to add to the group.

- context_handle CONTEXT HANDLE group context data established using gss_init_group_sec_context() or gss_init_sec_context().

- output_token OCTET STRING context token generated by successful call, to be passed to the new peer for processing by gss_accept_sec_context() or used as input data for multi-phase initiation (continuation).

Outputs:

- output_token OCTET STRING as above.

Return major_status codes:

- GSS_S_SUCCESS Successful completion

- GSS_S_FAILURE Failure. See minor_status for more information.

- GSS_S_CONTEXT_EXPIRED The context has already expired

- GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired
- GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context
- GSS_S_NO_CRED The supplied credentials did not reference any credentials.
- GSS_S_BAD_NAME The target_name value provided in the input_token was ill-formed.
- GSS_S_VOTING_INCOMPLETE Indicates that voting is not yet finalised.
- GSS_S_CONTINUE_NEEDED as per gss_init_sec_context().

The initiator calls gss_add_party(), supplying the context handle and details of the party to be added. The initiator must then pass an application message to all other parties, voters, informing them, at the application level, that an add request has been made. The voters must then register their vote using gss_vote(). Meanwhile the initiator is polling the mechanism using gss_add_party(), and continues to do so until all votes are registered. Once all votes are registered, addition success can be determined. If addition is successful, gss_add_party() returns a token which must be passed to the party being added, for context acceptance using gss_accept_sec_context().

The process by which the application calling gss_vote() determines the value of the vote it casts can be determined in a number of ways. Firstly, the application could arbitrarily acknowledge all add requests. Secondly, an access control list or trust scheme may be used to determine whether or not to acknowledge an add request. Alternatively, the user (if the application is a user-level process) may be prompted for a response, particularly in the case of an on-line, multi-party conversation.

The gss_vote() call:

Inputs:

- context_handle CONTEXT HANDLE group context data established using gss_init_group_sec_context or gss_init_sec_context.
- vote_status BOOLEAN indicating either "yes"/TRUE or "no"/FALSE response to the addition of the specified party.

Outputs:

o none

Return major_status codes:

- GSS_S_SUCCESS Successful completion
- GSS_S_FAILURE Failure. See minor_status for more information.
- GSS_S_CONTEXT_EXPIRED The context has already expired
- GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired
- GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context
- GSS_S_NO_CRED The supplied credentials did not reference any credentials.

There are alternatives to the voting-based style of access control. Access control lists could be used, trust-based methods or even simply security policy. However, the design attempts to allow for the greatest flexibility, whilst providing a useful means of access control. The voting scheme was seen to satisfactorily address these requirements.

The third call defined for the dynamic multi-party contexts specification enables a party to determine the list of parties sharing the context. The gss_context_parties() call consults the mechanism for an authoritative reply as to the participating members of the context at the current time. On successful completion the call returns a list of the names of all context members.

The gss_context_parties() call:

Inputs:

- context_handle CONTEXT HANDLE group context data established using gss_init_group_sec_context() or gss_init_sec_context().

Outputs:

- party_list NAME LIST The list of internal names of all parties sharing the context data

Return major_status codes:

- GSS_S_SUCCESS Successful completion
- GSS_S_FAILURE Failure. See minor_status for more information.

- GSS_S_CONTEXT_EXPIRED The context has already expired

- GSS_S_CREDENTIALS_EXPIRED The context is recognized, but associated credentials have expired

- GSS_S_NO_CONTEXT The context_handle parameter did not identify a valid context

- GSS_S_NO_CRED The supplied credentials did not reference any credentials.

3 New Data Structures

A new data structure is necessary for multi-party context support to enable a list of names to be passed into the gss_init_group_sec_context() call and return from gss_context_parties(). This data structure is a compound of an existing GSS-API structure in a basic list data type. A C binding for this data structure might be defined as follows:

```
typedef struct _name_list {
                    gss_name_type     name;
                    struct _name_list *next;
                }
        name_list;
```

Only one other possible change to GSS-API data structures might be required. The context data structure is typically highly mechanism dependant, but generally contains the names of all parties involved in the context. This data structure may, in some cases, need to be generalised to enable the storage of more than two party names, typically in the form of a single local name, and a list of the names of all other parties in the context. This data type must also be dynamic and able to be updated throughout the lifetime of the context, to enable appropriate handling of addition of new parties to the context.

4 References

Linn, J., "Generic Security Service Application Program Interface", RFC 1508, September 1993.

Wray, J., "Generic Security Service API : C-Bindings", RFC 1509, September 1993.

5 Acknowledgements

The author would like to acknowledge the assistance in preparing this draft and designing the original forms of multi-party security for implementation. The thanks of the author are extended to: Dr. Luke O'Connor (DSTC), Gary Gaskell (DSTC), John Linn (OpenVision), David

Arnold (DSTC) and many others.

6 Appendix A : C-Bindings

```
OM_uint32 gss_init_group_sec_context(
            OM_uint32*,       /* minor_status */
            gss_cred_id_t,    /* claimant_cred_handle */
            gss_ctx_id_t*,    /* context_handle */
            gss_name_list_t,  /* target_name list */
            const_gss_OID,    /* mech_type */
            int,              /* req_flags */
            OM_uint32,        /* time_req */
            gss_channel_bindings_t,
                              /* input_chan_bindings */
            gss_buffer_t,     /* input_token */
            gss_OID*,         /* actual_mech_type */
            gss_buffer_t,     /* output_token */
            int*,             /* ret_flags */
            OM_uint32*        /* time_rec */
          );

OM_uint32 gss_add_party(
            OM_uint32*,       /* minor_status */
            gss_name_t,       /* add party's name */
            gss_ctx_id_t,     /* context_handle */
            gss_buffer_t*     /* return GSS token */
          );

OM_uint32 gss_vote(
            OM_uint32*,       /* minor_status */
            gss_ctx_id_t,     /* context_handle */
            OM_uint32,        /* operation ADD/DELETE/ETC */
            OM_uint32         /* vote preference */
          );

OM_uint32 gss_context_parties(
            OM_uint32*,       /* minor_status */
            gss_ctx_id_t,     /* context_handle */
          );
```