INTERNET-DRAFT
CAT Working Group
Expires September 2000
draft-ietf-cat-gaa-cbind-03.txt

Tatyana Ryutov
Clifford Neuman
USC/Information Sciences Institute

March 9, 2000

Generic Authorization and Access control Application Program Interface C-bindings

0. Status Of this Document

1. Abstract

The Generic Authorization and Access control Application Programming
Interface (GAA API) provides access control services to calling
applications.
It facilitates access control decisions for applications and allows
applications to discover access control policies associated with a
targeted resource. The GAA API is usable by multiple applications
supporting different kinds of protected objects.
The GAA API design supports:

- a variety of security mechanisms based on public or secret key
  cryptosystems
- different authorization models
- heterogeneous security policies
- various access rights

This document specifies C language bindings for the GAA API, which
is described at a language-independent conceptual level in
draft-ietf-cat-acc-cntrl-frmw-03.txt

2. Approach

We propose an "object-oriented" approach inspired by the programming
style in [3]. It allows for better integration of application specific
modules with the GAA API.
We define three "classes": gaa, gaa_policy and gaa_sc.
The general structure of each class is depicted in Figure 1.

2.1. The class data structure

The class structure contains the following fields:

method
A pointer to the class method structure.

class_new(class_method*, class_ptr*)
creates a new class.

class_set(class_method*, class_ptr*)
sets appropriate fields of the class_method structure (attributes and methods).

class_free(class*)
frees the class structure. Depending on the configuration, this will free the
underlying data object.

2.2. "abstract" class_method data structure

The "abstract" class_method structure contains the following fields:

type
is the numeric type of the class_method.

name
is a textual representation of the method "type".

The class_method function pointers point to the respective
implementation specific function methods. Some of them can be NULL
if not implemented.

create()
creates a new class of type "type".

destroy()
frees class structure of type "type".

2.3. Implementation specific class_method data structure

This structure contains concrete values for the "abstract"
class_method attributes and methods which are set by the class_set
function.

```
-------------------------------      ------------------------------
|          class              |      |  "abstract"  class_method  |
| (gaa, gaa_policy or gaa_sc) |      |----------------------------|
|-----------------------------|      |         attributes         |
|        attributes           |      |                            |
|                             |      |  int   type                |
|  class_method  *method -----|----->|  char *name                |
|  class specific attributes  |      |  class specific attributes |
|-----------------------------|      |----------------------------|
|         methods             |      |         methods            |
|                             |      |                            |
| class_new(class_method*,    |      | create()                   |
|           class_ptr*)       |      | destroy()                  |
| class_set(class_method*,    |      | class specific methods     |
|           class*)           |      --------------^---------------
| class_free(class*)          |                    |
-------------------------------                    |
                                     --------------|---------------
                                     |   implementation specific  |
                                     |       class method         |
                                     |----------------------------|
                                     |         attributes         |
                                     |                            |
                                     | type=implem_specific_type  |
                                     | name=implem_specific_name  |
                                     | implementation specific    |
                                     | attributes                 |
                                     |----------------------------|
                                     |          methods           |
                                     |                            |
                                     | implem_specific_create()   |
                                     | implem_specific_destroy()  |
                                     ------------------------------
```

Figure 1.

3. The GAA API data types and calling conventions

The data types describe only fields that must be provided by all
GAA API implementations. Individual implementations may provide
additional fields for internal use within the GAA API routines.

3.1. Integer types

The GAA API defines the following integer data type:

```
uint32   32-bit unsigned integer
```

3.2. Opaque data types

Some data items are considered opaque to the GAA API, because their internal data structure has no significance to the GAA API, e.g. actual mechanism-specific credentials.
Opaque data is passed between the GAA API and the application using the gaa_buffer_ptr data type, which is a pointer to a gaa_buffer structure.

The gaa_buffer type is a structure containing the following fields:

length
Contains the total number of bytes in the datum

value
Contains a pointer to the actual datum

```
typedef struct gaa_buffer_struct gaa_buffer,
                                *gaa_buffer_ptr,
                                 gaa_options,
                                *gaa_options_ptr;

struct gaa_buffer_struct {
size_t     length;
void      *value;
};
```

3.3. Character data types

Certain data items used by the GAA API may be regarded as a character strings. The data of this kind is passed between the GAA API and application using the gaa_data data type, which is a pointer to void:

```
typedef char *gaa_data;
```

3.4. Ordered list types

Certain data items used by the GAA API may be regarded as an ordered list of objects. In this draft we refer to them as gaa_STACK data structure.
The implementation of the ordered list can be application specific.
A possible candidate for this type of data can be STACK structure in [3].

3.5. gaa_struct data structure

The gaa_struct structure is passed as an argument to the GAA API. It contains

information about behavior of the gaa evaluation routines.
See section 2 for explanation of the meaning of the fields
method, gaa_new, gaa_set and gaa_free.

```
typedef struct gaa_struct  gaa,
                          *gaa_ptr;

struct gaa_struct
{
  /* attributes */

  gaa_method_ptr  method;

  /* methods */

  gaa_error_code (*gaa_new)(gaa_ptr method,
                            gaa_ptr *gaa);
  gaa_error_code (*gaa_set)(gaa_method_ptr method,
                            gaa_ptr        gaa);
  gaa_error_code (*gaa_free)(gaa_ptr  gaa);
};
```

3.6. gaa_method_struct data structure

The gaa_method_struct structure contains the following fields:

condition_evaluation
Specific condition evaluation function called by GAA API if there are
application-specific conditions. Generic (understood by the GAA API) conditions
are evaluated by the GAA API internal functions.

See section 2 for explanation of the meaning of the fields type,
method, create and destroy.

```
typedef struct gaa_method_struct  gaa_method,
                                 *gaa_method_ptr;

struct gaa_method_struct
{
 /* attributes */

  int   type;
  char *name;

 /* methods */

  gaa_error_code (*condition_evaluation)(gaa_policy_ptr  policy,
                                         gaa_sc_ptr      sc,
                                         gaa_options_ptr options);
  gaa_error_code (*create)();
  gaa_error_code (*destroy)();
};
```

3.7. gaa_policy_struct data structure

The gaa_policy_struct structure contains the following fields:

policy
a pointer to memory which holds the application specific policy
structure

See section 2 for explanation of the meaning of the fields
method, gaa_policy_new, gaa_policy_set and gaa_policy_free.

```
typedef struct gaa_policy_struct  gaa_policy,
                                 *gaa_policy_ptr;

struct gaa_policy_struct
{
  /* attributes */

  gaa_policy_method_ptr  method;
  gaa_buffer_ptr         policy;
  /* methods */

  gaa_error_code (*gaa_policy_new)(gaa_policy_method_ptr method,
                                   gaa_policy_ptr        *policy);

  gaa_error_code (*gaa_policy_set)(gaa_policy_method_ptr method,
                                   gaa_policy_ptr        policy);

  gaa_error_code (*gaa_policy_free)(gaa_policy_ptr       policy);
};
```

3.8. gaa_policy_method_struct data structure

The gaa_policy_method_struct structure contains the following fields:

eval_method
defines a method for policy evaluation.
The default value is ordered policy evaluation.

get_matching_entries
implementation specific function for retrieval of the matching entries.
It returns an ordered list of objects of type gaa_policy_entry_ptr (see
section 3.9.), which are then evaluated by the gaa routines.

retrieve
application specific function for the retrieval of the object authorization
information. The application maintains authorization information in
a form understood by the application. It can be stored in a file,
database, directory service or in some other way. The upcall
function provided for the GAA API retrieves this information.

See section 2 for explanation of the meaning of the fields: type, method, create and destroy.

```
typedef struct gaa_policy_method_struct  gaa_policy_method,
                                        *gaa_policy_method_ptr;

struct gaa_policy_method_struct
{
  /* attributes */

  int   type;
  char *name;
  int   eval_method;   /* default ordered */

  /* methods */

  gaa_STACK_ptr /* gaa_policy_entry_ptr  */
              (*get_matching_entries) (gaa_buffer_ptr  policy,
                                        gaa_STACK_ptr /* gaa_right_ptr */
requested_rights);

  gaa_buffer_ptr(*retrieve)(uint32*      minor_status, /* OUT */
                            gaa_data object,  /* IN  */
                            gaa_data policy_db, ... );  /* IN  */

  gaa_error_code (*create)();
  gaa_error_code (*destroy)();
};
```

3.9. gaa_policy_entry_struct data structure

The gaa_policy_entrr_struct structure contains the following fields:

num
entry number in the policy

priority
specifies the priority of this entry

rights
A pointer to a linked list of structures of the type gaa_right_ptr.
Each structure indicates granted or denied access rights.

```
typedef struct gaa_policy_entry_struct  gaa_policy_entry,
                                       *gaa_policy_entry_ptr;

struct gaa_policy_entry_struct {
   int            num;
   int            priority;
   gaa_STACK_ptr /* gaa_right_ptr */ rights;
};
```

3.10. gaa_right_struct data structure

The gaa_right_struct structure contains the following fields:

type
An element of the type char*, which defines the type of the token.
Allowed token types are pos_access_rights and neg_access_rights.

authority
An element of the type char*, which indicates the authority
responsible for defining the value within the attribute type.

value
An element of the type char*, which indicates the value of the
right. The name space for the value is defined by
the "authority" field.

conditions
A pointer to an ordered list of objects of type gaa_condition_ptr.
It contains a list of pointers to conditions associated with the right.

```
typedef struct gaa_right_struct   gaa_right,
                                 *gaa_right_ptr;
struct gaa_right_struct {
   char*              type;
   char*              authority;
   char*              value;
   gaa_STACK_ptr /* gaa_condition_ptr */  conditions;
};
```

3.11. gaa_condition_struct

The gaa_condition_struct structure contains the following fields:

type
An element of the type char*, which defines the type of the condition.

authority
An element of the type char*, which indicates the authority
responsible for defining the value within the attribute type.

value
An element of the type char*, which indicates the value of the
security attribute. The name space for the value is defined by
the "authority" field.

status
Flags, indicating if the condition was evaluated or not evaluated,

if evaluated marked as met, not met or further evaluation or
enforcement is required.

```
typedef struct gaa_condition_struct  gaa_condition,
                                    *gaa_condition_ptr;

struct gaa_condition_struct {
   char*   type;
   char*   authority;
   char*   value;
   uint32  status;
};
```

3.12. gaa_sec_attrb_struct data structure

The gaa_sec_attrb_struct structure contains the following fields:

type
An element of the type char*, which defines the type of the security
attribute.

authority
An element of the type char*, which indicates the authority
responsible for defining the value within the attribute type.

value
An element of the type char*, which indicates the value of the
security attribute. The name space for the value is defined by
the "authority" field.

```
struct gaa_sec_attrb_struct {
   char*  type;
   char*  authority;
   char*  value;
};
```

3.13. GAA API Security Context data structures

The security context is a GAA API data structure, which is passed
as an argument to the GAA API. It stores information relevant to
access control.

3.13.1. gaa_sc_struct data structure

The gaa_sc_struct structure contains the following fields:

sc
a pointer to memory which holds the mechanism specific security
context structure

identity_cred

A pointer to an ordered list of structures of the type gaa_identity_cred

authr_cred
A pointer to an ordered list of structures of the type gaa_authr_cred

group_membership
A pointer to an ordered list of structures of the type gaa_identity_cred,
which specifies that the grantee is a member of only listed groups

group_non_membership
A pointer to an ordered list of structures of the type gaa_identity_cred,
which specifies that the grantee is NOT a member of the listed groups

attributes
A pointer to an ordered list of structures of the type gaa_attributes, which
contains miscellaneous attributes attached to the grantee, e.g. age
of the grantee, grantee's security clearance.

unevl_cred
A pointer to an ordered list of structures of type gaa_uneval_cred.

connection_state
Contains a mechanism-specific representation of per-connection
context, some of the data stored here include keyblocks, addresses.

pull_cred
This function is called when additional credentials are required.
It obtains the necessary credentials and then cred_evaluate
function is invoked. This process can be recursive.

cred_evaluate
This specific function is invoked to parse the contents of the
acquired credentials into the GAA API internal form and evaluate them.

See section 2 for explanation of the meaning of the fields
method, gaa_sc_new, gaa_sc_set and gaa_sc_free.

```
typedef struct gaa_sc_struct  gaa_sc,
                             *gaa_sc_ptr;

struct gaa_sc_struct
{
 /* attributes */

  gaa_sc_method_ptr  method;
  gaa_buffer_ptr     sc;

  gaa_STACK_ptr /* gaa_identity_cred_ptr */ identity_cred;
  gaa_STACK_ptr /* gaa_authr_cred_ptr    */ authr_cred;
  gaa_STACK_ptr /* gaa_identity_cred_ptr */ group_membership_cred;
  gaa_STACK_ptr /* gaa_identity_cred_ptr */ group_non_membership_cred;
  gaa_STACK_ptr /* gaa_attribute_ptr     */ attributes;
  gaa_STACK_ptr /* gaa_uneval_cred_ptr   */ unevl_cred;

  /* methods */

  gaa_error_code (*gaa_sc_new)(gaa_sc_method_ptr method,
                               gaa_sc_ptr        *sc);

  gaa_error_code (*gaa_sc_set)(gaa_sc_method_ptr method,
                               gaa_sc_ptr        sc);

  gaa_error_code (*gaa_sc_free)(gaa_sc_ptr sc);
};
```

3.13.2. gaa_sc_method_struct data structure

The gaa_sc_method_struct structure contains the following fields:

get_identity_cred
application specific function which translates mechanism specific
credentials to the gaa internal structure. It returns an ordered
list of objects of type gaa_identity_cred_ptr see section 3.13.3,
can be NULL if not implemented.

get_authr_cred
application specific function which translates mechanism
specific credentials to the gaa internal structure. It returns
an ordered list of objects of type gaa_authr_cred_ptr see
section 3.13.4,

get_group_membership_cred
application specific function which translates mechanism
specific credentials to the gaa internal structure. It returns an
ordered list of objects of type gaa_group_membership_cred_ptr see
section 3.13.5, can be NULL if not implemented.

get_group_non_membership_cred
application specific function which translates mechanism
specific credentials to the gaa internal structure. It returns
an ordered list of objects of type gaa_group_non_membership_cred_ptr
see section 3.13.6, can be NULL if not implemented.

get_attributes
application specific function which translates mechanism
specific credentials to the gaa internal structure. It returns
an ordered list of objects of type gaa_attribute_ptr see section 3.13.7,
can be NULL if not implemented.

get_uneval_cred
application specific function which translates mechanism
specific credentials to the gaa internal structure. It returns an
ordered list of objects of type gaa_uneval_cred_ptr see section 3.13.8,

can be NULL if not implemented.

See section 2 for explanation of the meaning of the fields type,
method, create and destroy.

```
typedef struct gaa_sc_method_struct  gaa_sc_method,
                                    *gaa_sc_method_ptr;

struct gaa_sc_method_struct
{
  /* attributes */

  int   type;
  char *name;

  /* methods */

    gaa_STACK_ptr /* gaa_identity_cred_ptr */
         (*get_identity_cred)(gaa_sc_ptr sc);
    gaa_STACK_ptr /* gaa_authr_cred_ptr    */
         (*get_authr_cred)(gaa_sc_ptr sc);
    gaa_STACK_ptr /* gaa_identity_cred_ptr */
         (*get_group_membership_cred)(gaa_sc_ptr sc);
    gaa_STACK_ptr /* gaa_identity_cred_ptr */
         (*get_group_non_membership_cred)(gaa_sc_ptr sc);
    gaa_STACK_ptr /* gaa_attribute_ptr     */
         (*get_attributes)(gaa_sc_ptr sc);
    gaa_STACK_ptr /* gaa_uneval_cred_ptr   */
         (*get_unevl_cred)(gaa_sc_ptr sc);

    gaa_error_code (*create)();
    gaa_error_code (*destroy)();
};
```

3.13.3. gaa_identity_cred_struct data structure

A gaa_identity_cred_struct structure is composed of a set of identity
credentials. Identity credentials describe a set of mechanism specific
principals, and give their holder the ability to act as any of those
principals. Each of the identity credentials contains information
needed to authenticate a single principal.

The gaa_identity_cred_struct structure contains the following fields:

principal
A pointer to a structure of the type gaa_sec_attrb_list

conditions
A pointer to an ordered list of objects of the type
gaa_sec_attrb_ptr,
which lists restrictions placed on the identity, e.g. validity time periods

mech_spec_cred
Contains a handle to the actual mechanism specific identity credential

```
typedef struct gaa_identity_cred_struct  gaa_identity_cred,
                                        *gaa_identity_cred_ptr;

struct gaa_identity_cred_struct {
   gaa_sec_attrb_ptr    principal;
   gaa_STACK_ptr /* gaa_condition_ptr */  conditions;
   gaa_buffer_ptr      mech_spec_cred;
};
```

3.13.4. gaa_authr_cred_struct data structure

The gaa_authr_cred_struct structure contains the following fields:

grantor
Specifies a principal who issued the credential

grantee
Specifies a principal for whom the credential was issued

objects
A pointer to a linked list of structures of the type gaa_data, which
contains a list of objects, which may be accessed by the grantee.
Object names are from the application-specific name space.

access_rights
A pointer to a linked list of structures of the type gaa_right_ptr.
Each structure indicates granted or denied access rights.

conditions
A pointer to an ordered list of objects of the type gaa_sec_attrb_ptr,
which lists restrictions placed on the authorized credentials

mech_spec_cred
Contains a handle to the actual mechanism-specific authorized
credential

```
typedef struct gaa_authr_cred_struct  gaa_authr_cred,
                                     *gaa_authr_cred_ptr;

struct gaa_authr_cred_struct{
   gaa_sec_attrb_ptr   grantor;
   gaa_sec_attrb_ptr   grantee;
   gaa_buffer          objects;
   gaa_STACK_ptr /* gaa_right_ptr */ access_rights;
   gaa_buffer_ptr      mech_spec_cred;
};
```

3.13.5. gaa_attribute_struct data structure

The gaa_attribute_struct structure contains the following fields:

mech_type
Security mechanism used to obtain the attribute

type
Type is used to define the type of attribute

value
Represents actual attribute contents

conditions
A pointer to an ordered list of objects of the type gaa_condition_ptr.
It contains pointers to conditions placed on the attribute credentials.

mech_spec_cred
Contains a handle to the actual mechanism specific attribute
credential

```
typedef struct gaa_attribute_struct  gaa_attribute,
                                     *gaa_attribute_ptr;

struct gaa_attribute_struct {
   char*               mech_type;
   char*               type;
   char*               value;
   gaa_STACK_ptr /* gaa_condition_ptr */  conditions;
   gaa_buffer_ptr      mech_spec_cred;
};
```

3.13.6. gaa_uneval_cred_struct data structure

Evaluation of the acquired credentials can be defferd till the
credential is actually needed. Unevaluated credentials are stored in
the gaa_uneval_cred_struct data structure.

The gaa_uneval_cred_struct structure contains the following
fields:

cred_type
Specifies credential type: GAA_IDENTITY, GAA_GROUP_MEMB,
GAA_GROUP_NON_MEMB, GAA_AUTHORIZED, and GAA_ATTRIBUTES.

grantor
Specifies a principal who issued the credential

grantee
Specifies a principal for whom the credential was issued

mech_type
Specifies security mechanism used to obtain the credential

mech_spec_cred
Contains a handle to the actual mechanism-specific authorization
credential

cred_verification
This pointer to the credential verification function for upcall is
added by the application or transport

```
typedef enum  {
      GAA_IDENTITY        ,
      GAA_GROUP_MEMB      ,
      GAA_GROUP_NON_MEMB  ,
      GAA_AUTHORIZED      ,
      GAA_ATTRIBUTES
 } gaa_cred_type;

typedef struct gaa_uneval_cred_struct   gaa_uneval_cred,
                                       *gaa_uneval_cred;

struct gaa_uneval_cred_struct {
   gaa_cred_type     cred_type;
   gaa_sec_attrb_ptr grantor;
   gaa_sec_attrb_ptr grantee;
   gaa_buffer_ptr    mech_spec_cred;
   void (*cred_verification )(gaa_ptr, va_list ap);
};
```

3.13.7. GAA API answer data structure

The gaa_check_authorization function returns various information to
the application for further evaluation in the gaa_answer data
structure.

The gaa_answer_struct structure contains the following fields:

valid_time
A pointer to a structure of type gaa_time_period. It specifies the
time period during which the authorization is granted and
is returned as a condition to be checked by the application.

rights
A pointer to an ordered list of structures of the type
gaa_right_ptr listing granted rights and corresponding conditions, if any.

```
typedef struct gaa_time_period_struct  gaa_time_period,
                                      *gaa_time_period_ptr;
struct gaa_time_period_struct{
   time_t    start_time; /* NULL for unconstrained start time */
   time_t    end_time;   /* NULL for unconstrained end time */
};

typedef struct gaa_answer_struct  gaa_answer,
                                 *gaa_answer_ptr;

struct gaa_answer_struct
{
   gaa_time_period_ptr  valid_time;
   gaa_STACK_ptr /* gaa_right_ptr */  rights;
};
```

4. Status codes

One or two status codes are returned by each GAA API routine. Two distinct sorts of status codes are returned. These are the GAA API status codes and mechanism specific status codes.

4.1. The GAA API status codes

GAA API routines return GAA API status codes as their gaa_error_code function value. These codes indicate errors that are independent of the underlying mechanisms. The errors that can be indicated via a GAA API status code are either generic API routine errors (errors that are defined in the GAA API specification) or calling errors (errors that are specific to these language bindings).

4.2. Mechanism specific status codes

GAA API routines return a minor_status parameter, which is used to indicate specialized errors from the underlying mechanisms or provide additional information about GAA API errors.
The GAA status code GAA_FAILURE is used to indicate that the underlying mechanism detected an error for which no specific GAA status code is defined. The mechanism status code will provide more details about the error.

5. GAA API routine descriptions

This section lists the functions performed by each of the GAA API routines and discusses their major parameters, describing how they are to be passed to the routines.

5.1. gaa_get_object_policy_info routine

Purpose:

The gaa_get_object_policy_info function is called to obtain security policy information associated with the object.

Parameters:

minor_status
mechanism specific status code

object
Reference to the object to be accessed. The identifier for the object
is from an application specific name space and is opaque to the
GAA API.

authr_db
Pointer to an application specific authorization database

policy_hadle
A pointer to a handle bound to the sequence of security attributes
which constitute the security policy associated with the targeted object
An unbound handle has the value GAA_UNBOUND.

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE   Failure, see minor_status for more information

```
gaa_error_code
gaa_get_object_policy_info(uint32*         minor_status,  /* OUT */
                           gaa_data        object,        /* IN  */
                           gaa_data        policy_db,     /* IN  */
                           gaa_policy_ptr  policy         /* OUT */);
```

5.2. gaa_check_authorization routine

Purpose:

The gaa_check_authorization function tells the application whether
the requested access rights are authorized, or if additional
application specific checks are required.

Parameters:

minor_status
Mechanism specific status code

policy_handle
A handle to the gaa_policy structure, returned by the
gaa_get_object_policy_info routine

gaa
A handle to the gaa structure

sc
A handle to the principal's security context

check_access_rights
Ordered list of access rights for authorization.

gaa_options
This argument contains parameters for parameterized operation.

detailed_answer
Contains various information for further evaluation by the application

Function value:

GAA status code:

GAA_FAILURE
Failure, see minor_status for more information

GAA_NO_CONTEXT
No valid security context was supplied

GAA_YES
(indicates authorization) is returned if all requested operations are
authorized.

GAA_NO
(indicates denial of authorization) is returned if at least one operation is
not authorized.

GAA_MAYBE
(indicates a need for additional checks) is returned if there are some
unevaluated
conditions and additional application specific checks are needed, or continuous
evaluation is required.

```
gaa_error_code
gaa_check_authorization
       (uint32            *minor_status,   /* OUT    */
        gaa_ptr            gaa,            /* IN&OUT */
        gaa_sc_ptr         sc,             /* IN&OUT */
        gaa_policy_ptr     policy_handle,  /* IN     */
        gaa_options_ptr    gaa_options,    /* IN, OPTIONAL */
        gaa_STACK_ptr /* gaa_right_ptr */  check_access_rights /* IN     */
        gaa_answer_ptr    *detailed_answer  /* OUT    */
       );
```

5.3. gaa_inquire_object_policy_info routine

Purpose:

The gaa_inquire_object_policy_info function allows application
to discover a particular user's rights on an object.

Parameters:

minor_status
Mechanism specific status code

gaa
A handle to the gaa structure

sc
A handle to the principal's security context

policy_handle
A handle to the gaa_policy structure, returned by the
gaa_get_object_policy_info routine

rights
A handle to the ordered list of objects of type gaa_right_ptr, which
contains list of rights that the principal is granted or denied.

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE
Failure, see minor_status for more information

GAA_NO_CONTEXT
No valid security context was supplied

gaa_error_code
gaa_inquire_policy_info

```
       (uint32         *minor_status,  /* OUT    */
        gaa_ptr         gaa,           /* IN&OUT */
        gaa_sc_ptr      sc,            /* IN&OUT */
        gaa_policy_ptr  policy_handle, /* IN     */
gaa_STACK_ptr /* gaa_policy_entry_ptr */  *rights   /* OUT    */
      );
```

5.4. gaa_allocate_buffer routine

Purpose:
Allocate a gaa_buffer data structure and assign default values

Parameters:

buffer
Pointer to allocated memory for gaa_buffer structure

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE
Failure

```
gaa_error_code
gaa_allocate_buffer
        (gaa_buffer_ptr*  buffer /* IN  */);
```

5.5. gaa_release_buffer routine

Purpose:

Free storage associated with a buffer format name. The storage must have been allocated by a GAA API routine. In addition to freeing the associated storage, the routine will zero the length field in the buffer parameter.

Parameters:

minor_status
Mechanism specific status code

buffer
The storage associated with the buffer will be deleted.
The gaa_buffer object will not be freed, but its length field will be zeroed.

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE
Failure, see minor_status for more information

GAA_NO_BUFFER
No valid buffer was supplied

```
gaa_error_code
```

```
gaa_release_buffer (uint32*          minor_status, /* OUT */
                    gaa_buffer_ptr   buffer        /* IN */);
```

5.6. gaa_allocate_answer routine

Purpose:
Allocate a gaa_answer data structure and assign default values

Parameters:

buffer
Pointer to allocated memory for gaa_buffer structure

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE
Failure

```
gaa_error_code
gaa_allocate_answer
        (gaa_answer_ptr*  buffer /* IN  */);
```

5.7. gaa_release_answer

Free storage associated with a buffer

Parameters:

minor_status
Mechanism specific status code

buffer
The storage associated with the buffer will be deleted

Function value:

GAA status code:

GAA_SUCCESS
Successful completion

GAA_FAILURE
Failure, see minor_status for more information

GAA_NO_BUFFER
No valid buffer was supplied

```
gaa_error_code
gaa_release_answer(uint32         *minor_status,
                   gaa_answer_ptr *buffer)
```

6. The GAA API constants

The following constants are used in GAA API calls and structures, this list is not complete:

```
#define GAA_NO_OPTIONS            ((gaa_options_ptr)0)
#define GAA_NO_BUFFER             ((gaa_buffer_ptr)0)
#define GAA_EMPTY_BUFFER          {0, NULL}
#define GAA_NO_DATA               ((gaa_data) 0)
#define GAA_NO_SEC_CONTEXT        ((gaa_sc_ptr)0)
#define GAA_NO_RIGHTS             ((gaa_right_ptr) 0)
#define GAA_NO_ANSWER             ((gaa_answer_ptr)0)

#define GAA_YES       0  (indicates authorization) is returned if all
                         requested operations are authorized.

#define GAA_NO        1  (indicates denial of authorization) is returned
                         if at least one operation is not authorized.

#define GAA_MAYBE    -1  (indicates a need for additional checks) is
                         returned if there are some unevaluated conditions
                         and additional application specific checks are needed,
                         or continuous evaluation is required.
```

7. The GAA API flags

Flags are 32 bits.

Condition flags:

```
#define COND_FLG_EVALUATED    0x01  condition has been evaluated
#define COND_FLG_MET          0x10  condition has been met
#define COND_FLG_ENFORCE      0x100 condition has to be enforced
```

8. References

[1] Linn, J., "Generic Security Service Application Program Interface", RFC 1508, Geer Zolot Associate, September 1993.

[2] Wray, "Generic Security Service Application Program Interface V2 - C bindings", Internet draft, May 1997.

[3] T J Hudson, E A Young SSLeay http://www.livjm.ac.uk/tools/ssleay/

9. Acknowledgments

Carl Kesselman and Douglas Engert have contributed to discussion

of ideas and material in this draft.

10. Authors' Addresses

Tatyana Ryutov
Clifford Neuman
USC/Information Sciences Institute
4676 Admiralty Way Suite 1001
Marina del Rey, CA 90292-6695
Phone: +1 310 822 1511
E-Mail: {tryutov, bcn}@isi.edu