X/Open Snapshot

Generic Security Service API (GSS-API)

Security Attribute and Delegation Extensions

X/Open Company Ltd.

X/Open Snapshot

Generic Security Service API (GSS-API) Security Attribute and Delegation Extensions

ISBN: 1-85912-026-1
X/Open Document Number: S307

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

List of Examples

List of Tables

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and allows users to move between systems with a minimum of retraining.

The components of the Common Applications Environment are defined in X/Open CAE Specifications. These contain, among other things, an evolving portfolio of practical application programming interfaces (APIs), which significantly enhance portability of application programs at the source code level, and definitions of, and references to, protocols and protocol profiles, which significantly enhance the interoperability of applications.

The X/Open CAE Specifications are supported by an extensive set of conformance tests and a distinct X/Open trade mark - the XPG brand - that is licensed by X/Open and may be carried only on products that comply with the X/Open CAE Specifications.

The XPG brand, when associated with a vendor's product, communicates clearly and unambiguously to a procurer that the software bearing the brand correctly implements the corresponding X/Open CAE Specifications. Users specifying XPG conformance in their procurements are therefore certain that the branded products they buy conform to the CAE Specifications.

X/Open is primarily concerned with the selection and adoption of standards. The policy is to use formal approved *de jure* standards, where they exist, and to adopt widely supported *de facto* standards in other cases.

Where formal standards do not exist, it is X/Open policy to work closely with standards development organisations to assist in the creation of formal standards covering the needed functions, and to make its own work freely available to such organisations. Additionally, X/Open has a commitment to align its definitions with formal approved standards.

X/Open Specifications

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the long-life specifications that form the basis for conformant and branded X/Open systems. They are intended to be used widely within the industry for product development and procurement purposes.

Developers who base their products on a current CAE Specification can be sure that either the current specification or an upwards-compatible version of it will be referenced by a future XPG brand (if not referenced already), and that a variety of compatible, XPG-branded systems capable of hosting their products will be available, either immediately or in the near future.

CAE Specifications are not published to coincide with the launch of a particular XPG brand, but are published as soon as they are developed. By providing access to its specifications in this way, X/Open makes it possible for products that conform to the CAE (and hence are eligible for a future XPG brand) to be developed as soon as practicable, enhancing the value of the XPG brand as a procurement aid to users.

- *Preliminary Specifications*

  These are specifications, usually addressing an emerging area of technology, and consequently not yet supported by a base of conformant product implementations, that are released in a controlled manner for the purpose of validation through practical implementation or prototyping. A Preliminary Specification is not a ‘‘draft’’ specification. Indeed, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE Specification.

  Preliminary Specifications are analogous with the ‘‘trial-use’’ standards issued by formal standards organisations, and product development teams are intended to develop products on the basis of them. However, because of the nature of the technology that a Preliminary Specification is addressing, it is untried in practice and may therefore change before being published as a CAE Specification. In such a case the CAE Specification will be made as upwards-compatible as possible with the corresponding Preliminary Specification, but complete upwards-compatibility in all cases is not guaranteed.

In addition, X/Open periodically publishes:

- *Snapshots*

  Snapshots are ‘‘draft’’ documents, which provide a mechanism for X/Open to disseminate information on its current direction and thinking to an interested audience, in advance of formal publication, with a view to soliciting feedback and comment.

  A Snapshot represents the interim results of an X/Open technical activity. Although at the time of publication X/Open intends to progress the activity towards publication of an X/Open Preliminary or CAE Specification, X/Open is a consensus organisation, and makes no commitment regarding publication.

  Similarly, a Snapshot does not represent any commitment by any X/Open member to make any specific products available.

X/Open Guides

X/Open Guides provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant.

X/Open Guides are not normative, and should not be referenced for purposes of specifying or claiming X/Open-conformance.

This Document

This document is a Snapshot (see above). It consists of material from a SESAME proposal for extensions to the Generic Security Service API (GSS-API) and an IETF proposal for Privilege Attribute Certificates and authorisation services, which was originally in RFC 1508.

- Chapter 1 is an introduction.
- Chapter 2 introduces the SESAME proposals for additional interfaces.
- Chapter 3 defines the data types required by the C-language extensions.
- Chapter 4 contains C-language function specifications for the additional interfaces.
- Chapter 5 discusses access control.
- Chapter 6 discusses a proposal to support Privilege Attribute Certificates and authorisation services.
- A glossary and index are also provided.

Intended Audience

This document is intended for system and application programmers. Readers are expected to have read the **Base GSS-API** specification (see **Referenced Documents** on page x).

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, and C-language keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — C-language variable names, for example, substitutable argument prototypes
  - — C-language functions; these are shown as follows: *name*( ).
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- The notation [EABCD] is used to identify a C-language return code EABCD.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width font`*.

Trade Marks

Kerberos™ is a trade mark of the Massachusetts Institute of Technology.

NetWare® is a registered trade mark of Novell, Inc.

Novell® is a registered trade mark of Novell, Inc.

OSF™ is a trade mark of Open Software Foundation, Inc.

UNIX® is a registered trade mark of UNIX System Laboratories, Inc. in the U.S.A. and other countries.

X/Open™ and the ''X'' device are trade marks of X/Open Company Ltd.

Acknowledgements

X/Open gratefully acknowledges SESAME (Bull, ICL and SNI) for permitting use of their proposal **An Extended Generic Security Service API**. The author of this document is Piers McMahon of ICL. Acknowledgements are due to Denis Pinkas and Eric Baize for reviewing and providing many useful comments on this paper.

X/Open also gratefully acknowledges the work of the X/Open Security Working Group in the development of this document. Many members of the X/Open Security Working Group have contributed to this specification, by reviewing drafts. In particular, thanks are due to:

Dave Bauer, Bellcore
Denis Pinkas, Groupe Bull
Peter Callaway, IBM
Piers McMahon, ICL
John Linn, OpenVision Technologies
Craig Heath, SCO
Ingo Hoffmann, Siemens-Nixdorf
Joe Brame, Unisys

X/Open gratefully acknowledges the Open Software Foundation for its agreement to allow two of their internal documents that are referenced in this document to be made available to readers of this document by X/Open.

Referenced Documents

The following standards and papers are referenced in this specification:

ABLP-AC

M.Abadi, M.Burrows, B.Lampson, G.Plotkin, Digital Equipment Corporation Systems Research Center, A Calculus for Access Control in Distributed Systems, report No. 70, February 1991.

ACM

ECMA Draft Standard, Association Context Management — including Security Context Management, Draft 8, May 1993.

Base GSS-API

X/Open Preliminary Specification, January 1994, Generic Security Service API (GSS-API) Base (ISBN: 1-85912-025-3, P308).

Extended GSS-API

Piers McMahon, SESAME (Bull-ICL-SNI), An Extended Generic Security Service API, Issue 4, 23 July 1993.

GSAI

N.Pope, Generic Security Abstract Interface, ISO/IEC JTC1/SC21/WG6 N1158, March 1992.

GSS-DCE

Internal OSF publication by J.Wray, GSS-API Extensions for DCE, DCE-RFC 5.1, January 1994. To obtain this document, please send electronic mail to:

XoSpecs@xopen.co.uk

ISO/IEC 7498-2

ISO/IEC 7498-2: 1989, Information Processing Systems — Open Systems Interconnection — Basic Reference Model — Security Architecture.

ISO 8859-1

ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.

ISO/IEC 9594-2

ISO/IEC 9594-2: 1990, DAM 1, Information Technology — Open Systems Interconnection — The Directory — Part 2: Models, together with:

Draft Amendment 1: Access control.

ISO/IEC 10181-1 CD

ISO/IEC CD 10181-1, 13AUG92, ISO/IEC JTC1/SC21 N7083, Information Technology — Open Systems Interconnection — Security Frameworks in Open Systems — Part 1: Security Frameworks Overview.

ISO/IEC 10181-2 DIS

ISO/IEC DIS 10181-2.2, 29JUL93, Information Technology — Open Systems Interconnection — Security Frameworks in Open Systems — Part 2: Authentication Framework.

ISO/IEC 10181-3 CD

ISO/IEC CD 10181-3, 26JUN91 ISO/IEC JTC1/SC21 N6168, Information Technology — Open Systems Interconnection — Security Frameworks in Open Systems — Part 3: Access Control.

ISO / IEC 10745 DIS

ISO / IEC DIS 10745: 19??, Information Technology — Open Systems Interconnection — Upper Layers Security Model.

ISO / IEC 10745 DIS comments

ISO / IEC JTC1 / SC21 N7621, table of replies on ISO / IEC DIS 10745: 19??, Information Technology — Open Systems Interconnection — Upper Layers Security Model, 22 February 1993.

LABW-AC

Butler Lampson, Martin Abadi, Michael Burrows, Edward Wobber, Authentication in Distributed Systems: Theory and Practice, Proceedings of the 13th ACM Symposium on Operating System Principles, October 1991.

N-RP

B.Clifford Neuman, Proxy-Based Authorisation and Accounting for Distributed Systems, ISI / USC, Proceedings of the 13th International Conference on Distributed Computing Systems, Pittsburgh, May 1993.

POSIX.6

Draft Standard for Information Technology — Portable Operating Sytem Interface (POSIX) — Part 1: System Application Program Interface (API) — Amendment #: Protection, Audit, and Control Interfaces [C Language], P1003.6.1 / D13 November 1992, Institute of Electrical and Electronics Engineers, Inc.

RFC 3

Internal OSF publication by J.Pato, Extending the DCE Authorisation Model to Support Practical Delegation, DCE-RFC 3, June 1992. To obtain this document, please send electronic mail to:

XoSpecs@xopen.co.uk

RFC 3+

M.Erdos and J.Pato, Extending the OSF DCE Authorisation Model to Support Practical Delegation, Proceedings of Privacy and Security Research Group Workshop on Network and Distributed System Security, February 1993.

RFC 19

B.Fairthorne, Proposed Security Enhancements for DCE1.1, DCE-RFC 19, December 1992.

RFC 1508

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

RFC 1510

Internet Proposed Standard, The Kerberos Network Authentication System, John Kohl, B.Clifford Neuman, issue 5.2, 21 April 1993.

TMACH

NIST Document Number: TMACH / 93-013, Trusted Mach, System Architecture & Philosophy of Protection, Trusted Information Systems, Inc, Draft May 1993.

TR / 46

ECMA TR / 46 — ECMA Security in Open Systems — A Security Framework, July 1988.

X.509

ISO / IEC 9594-8: 1990 Information Technology — Open Systems Interconnection — The Directory — Part 8: Authentication Framework, together with:

Technical Corrigendum 1: 1991 to ISO / IEC 9594-8: 1990.

The following X / Open documents are referenced in this specification.

DCE Security

Future X / Open Preliminary Specification, X / Open DCE: Security.

Procurement Guide

X / Open Guide, September 1992, Defining and Buying Secure Open Systems (ISBN: 1-872630-61-8, G206).

Security Guide

X / Open Guide, February 1991, Security Guide, Second Edition (ISBN: 1-872630-07-3, G010 or XO / GUIDE / 90 / 010).

XOM

X / Open CAE Specification, November 1991, OSI-Abstract-Data Manipulation API (XOM) (ISBN: 1-872630-17-0, C180 or XO / CAE / 91 / 080).

Chapter 1

Introduction

The base Generic Security Service API (GSS-API) defined in the **Base GSS-API** specification meets many of the needs of applications using distributed security services. However, some requirements cannot be met by the base GSS-API. This chapter explains the base GSS-API design goals and why there is a need for extensions. This chapter also gives an overview of the base GSS-API and its use.

1.1 GSS-API Design Goals

The GSS-API design assumes and addresses several basic goals, including:

Mechanism independence

The GSS-API defines an interface to cryptographically-implemented strong authentication and other security services at a generic level that is independent of particular underlying mechanisms. For example, services provided by GSS-API can be implemented by secret-key technologies (for example, Kerberos) or public-key approaches (for example, X.509).

Protocol environment independence

The GSS-API is independent of the communication protocol suites with which it is employed, permitting use in a broad range of protocol environments. In appropriate environments, GSS-API need not be directly invoked by applications, but may form an intermediate layer that is indirectly invoked. For example, in an RPC environment, GSS-API may be layered beneath RPC (or above it).

Protocol association independence

The GSS-API's security context construct is independent of communication protocol association constructs. This characteristic allows a single GSS-API implementation to be utilised by a variety of invoking protocol modules on behalf of those modules' calling applications. GSS-API services can also be invoked directly by applications, wholly independent of protocol associations.

Suitability to a range of implementation placements

GSS-API clients are not constrained to reside within any Trusted Computing Base (TCB) perimeter defined on a system where the GSS-API is implemented; security services are specified in a manner suitable to both intra-TCB and extra-TCB callers.

1.2 GSS-API Security Services and the Need for Extensions

The base GSS-API supports services such as mutual authentication and data confidentiality and integrity. As currently specified the base GSS-API can:

- enable a client to form a secure association with a server
- permit a client and server to protect their dialogue for integrity with or without confidentiality
- allow a server to act as a client in calling another application server
- allow application or system selection of security mechanism (this allows an application to participate in a variety of security contexts, using the security mechanism appropriate for each).

The base GSS-API is supportive of a limited form of authorisation: identity-based access control lists. As GSS-API is currently specified, an access control security service is not fully supported, because in the ISO / IEC 7498-2 standard a wider view of distributed authorisation is described than identity-based access control lists. Additionally, controls on delegation are primitive in the base GSS-API and do not permit selective delegation of access rights to specific targets. Without changes to the GSS-API interface to support access control adequately, a wide class of distributed applications are prevented from being mechanism independent. Hence, a set of complementary APIs are required to provide these essential security facilities not accessible through GSS-API as currently defined. These include support for authorisation based on subject privileges and controlling the targets that can use credentials for access control or for delegation by forwarding them to other targets for use by proxy. Such extensions to GSS-API are defined in the RFC 19 paper and are restated and discussed here to show how they:

- allow a server to obtain attributes associated with a client for use in its own access control decisions
- permit a client to specify required access control attributes, and the constraints on their use.

When used in conjunction with the GSS-DCE paper extensions, the RFC 19 paper APIs enable portable access to DCE security services. In addition to the APIs defined in the RFC 19 paper, some complementary generic APIs that are supportive of delegation are considered.

At some stage, other application security services not available through the base GSS-API should be defined, including principal authentication, non-repudiation, data labelling, and making authorisation decisions. Hence, for complete coverage of security services, portable APIs should be specified in these areas. These API areas are not considered further; note that some are the subject of current work in the SESAME project, IETF and recent initiatives in POSIX.

1.3 Overview of the Base GSS-API and its Use

Where security is integrated into the communication infrastructure, most security details can be applied without the application having to take any action. The administrator can specify default values for security parameters associated with either the user's role or particular applications, and these are used automatically unless the application specifies otherwise. Such applications or components should be able to use standard library interfaces (such as XTI or RPC) without recourse to security APIs (such as GSS-API), if the communication library provides the necessary security services. Some applications are more security aware and want more explicit control of the security services they call. This also applies to any secure communications infrastructure. Such security-aware software entities may therefore ask for any non-default security services to be used, for example, to reduce the time for which credentials can be used and with which targets they can be used. They may also want explicit control of delegation. A mechanism-independent security API is required for such software. The GSS-API provides a number of interfaces, the most important of which are as follows:

- Credential handling:

  GSS_Acquire_cred( )
  GSS_Inquire_cred( )
  GSS_Release_cred( ).

- Security Context handling:

  GSS_Init_sec_context( )
  GSS_Accept_sec_context( )
  GSS_Delete_sec_context( )
  GSS_Process_context_token( )
  GSS_Context_time( ).

- Protection of application dialogue:

  GSS_Sign( )
  GSS_Verify( )
  GSS_Seal( )
  GSS_Unseal( ).

- Support APIs:

  GSS_Display_status( )
  GSS_Indicate_mechs( )
  GSS_Compare_name( )
  GSS_Display_name( )
  GSS_Import_name( )
  GSS_Release_name( )
  GSS_Release_buffer( )
  GSS_Release_oid_set( ).

Chapter 2

Extended GSS-API

This chapter outlines how the extended GSS-API can be used, and includes consideration of how it may be used to support environment-specific security interfaces such as those being developed for DCE.

2.1 Candidate Extensions to GSS-API

There are two main, not (necessarily) mutually exclusive, published proposals for GSS-API extensions:

DCE-specific GSS-API extensions	the GSS-DCE paper
Generic GSS-API extensions	the RFC 19 paper

The extensions proposed are summarised below.

2.1.1 DCE-specific GSS-API Extensions

DCE uses privilege attribute certificates (PACs) and login contexts. The following APIs are specific to the DCE environment and provide facilities for GSS-API applications to use DCE security interfaces (*sec_*( )* and *sec_acl_*( )*).

GSSDCE_Extract_PAC_From_SecContext( )
Gets a reference to a DCE PAC from a GSS-API security context.

GSSDCE_Extract_PAC_From_Cred( )
Gets a reference to a DCE PAC from a GSS-API credentials handle.

GSSDCE_LoginContext_To_Cred( )
Maps DCE login context to GSS-API credentials.

GSSDCE_Cred_To_LoginContext( )
Maps GSS-API credentials to DCE login contexts.

Early versions of the DCE APIs are described in the overview in the GSS-DCE paper and will be published by OSF in due course.

2.1.2 Proposed Security Attribute and Delegation Extensions

This document specifies the following extensions, which are outlined in the RFC 19 paper:

gss_get_attributes( )
Allows callers to retrieve attributes associated with credentials or a security context.

gss_modify_cred( )
Permits a caller to change privilege and control attributes in an existing set of credentials or to create a variant set of credentials — this includes selective controls over delegation.

Note that an SC21/WG8 input document, the GSAI paper, proposes a generalisation of the existing GSS-API and also new authorisation interfaces. It suggests similar interfaces for modification of credentials and security attribute retrieval as proposed in the above generic extensions.

Additional generic GSS-API extensions for sophisticated security requirements including support of *delegate credentials*, and other multiple credential servers are as follows:

gss_set_default_cred( )
Specifies alternative default credentials.

gss_get_delegate_creds( )
Retrieves delegated credentials associated with credentials or a security context.

gss_compound_cred( )
Links two credentials to produce one which is a compound of the two.

The generic extensions outlined above are detailed in Chapter 3, and are discussed further in the next section.

2.2 Overview of the Extended GSS-API

The main features of the basic *gss_get_attributes*( ) and *gss_modify_cred*( ) functions are outlined in Section 2.2.1 and Section 2.2.2. Subsequent sections cover the slightly more complex cases of delegation.

2.2.1 Security Attribute Querying

A user of the Extended GSS-API can use the *gss_get_attributes*( ) function to inspect security attributes if the mechanism makes these available. This function can be used by initiating clients, delegates and servers to query attributes in credentials or security contexts. Security attributes that can be queried include:

- identifiers — such as the audit identity
- privileges — such as access identity, group and role (which may be as delegate or direct accessor or both)
- controls — time limitations, legitimate target and delegation constraints
- restrictions — optional or required application-specific access controls
- user-defined attributes, which may qualify other attributes, or be used in peer-to-peer negotiations.

2.2.2 Security Attribute Request

A user of the Extended GSS-API can use the *gss_modify_cred*( ) function to request security attributes (such as discussed above for *gss_get_attributes*( )) if the mechanism permits this. This function enables the security-aware user to create and then cumulatively annotate credentials with requests for particular access rights, and constraints on how these access rights are propagated through the network.

It is not specified at the API whether calls to *gss_modify_cred*( ) result in calls to security services required by particular mechanisms. The only requirement is that any annotated credential requirements are applied when a *gss_init_sec_context*( ) operation is invoked. However, for optional user control over when credentials may be modified by processes or threads that inherit or have access to these credentials, the *commit_cred_req* boolean option is provided at the API. By asserting this option, the API user prevents further annotation of the credentials. This causes failure of subsequent *gss_modify_cred*( ) operations that do not specify *duplicate_cred_req*==TRUE. Hence the credentials are effectively frozen. The **gss_target_control_set** structure passed to *gss_modify_cred*( ) is a compound structure that is used to describe which constraints on the use of access rights apply to which targets. It specifies whether access rights can be forwarded to certain targets, whether they can be delegated to certain targets, and the time period for which the access rights are valid. For each group of targets a set of delegation options is available by setting a **delegation_flags** field, which includes settings for forbidding delegation, simple delegation or traced delegation.

2.2.3 Simple Delegation

A distributed application can consist of many clients servicing users, some intermediate servers which perform a routing or relay function, and a number of *target* servers which access resources on the application users' behalf.

Such an application requires simple delegation if each target is required to make access control decisions based only on the initiating user's privilege attributes, and thus the delegate's privilege attributes are of no interest. Simple delegation is also required if such targets must further delegate the user's access rights within a group of servers in order to achieve the required processing.

If a user of this distributed application needs to delegate its access rights solely to a specific group of targets (for example, a specific intermediary and a group of targets), then the user calls *gss_modify_cred*( ) to create a new set of credentials specifying in the **gss_target_control_set** that the **delegation_flags** field for these targets is set to can_be_delegate_and_target.

After the normal *gss_init_sec_context*( ) or *gss_accept_sec_context*( ) operations described in Section 1.3 on page 3, the intermediate server possesses a set of delegated credentials emitted from *gss_accept_sec_context*( ) (the *delegated_cred_handle*).

These delegated credentials can be used in the normal way to initiate further security contexts, but such attempts only succeed for targets included in the group originally specified by the initiating user. The mechanism constrains whether *gss_modify_cred*( ) can be used by the intermediate server with delegated credentials, but some mechanisms permit the intermediate server to reduce (but normally not increase) the ability to delegate an initiator's access rights. When used in conjunction with DCE RPC interfaces, the GSS-DCE paper interfaces are required to convert between DCE login contexts and GSS-API credentials. For example, if an RPC client is used to access a non-RPC target by means of a gateway intermediate server, the annotated credentials are converted to DCE login contexts by means of *GSSDCE_Cred_To_Login_Context*( ); then the client login context is set by means of *rpc_binding_set_auth_info*( ). At the server, the *rpc_binding_inq_auth_client*( ) function is used to retrieve a login context, which is converted by means of *GSSDCE_LoginContext_To_Cred*( ) into a GSS-API credential.

2.2.4 Traced Delegation

If the targets of a distributed application need to inform their access control decisions of the access rights or other security attributes of the delegating intermediate servers, in addition to the security attributes of the initiating user, then traced delegation must be used.

An example of such a requirement could be if particular delegation routes are specifically mandated or, conversely, prohibited. Where traced delegation is required, the **delegation_flags** field specified by the initiating user must include trace_required.

The initiating user, and each subsequent delegate specifies the next link in the delegation chain by the target specified in *gss_init_sec_context*( ).

When the intermediate server obtains the delegated credentials (whether by means of RPC or not), it has two options:

- In the first (simpler) case, the security attributes of the intermediate server are the same for all targets; any security contexts initiated by the intermediate server combine its default delegate credentials with the initiating user's credentials when these user's credentials are used as an argument to *gss_init_sec_context*( ). This obviates the need for a separate operation to compound the user's credentials with the server's default delegate credentials.

  The application target server is able to retrieve any delegate credentials from the credentials emitted by *gss_accept_sec_context*( ) (the *delegated_cred_handle*) by means of *gss_get_delegate_creds*( ). Each such credential's security attributes can be inspected using *gss_get_attributes*( ).

- Alternatively, the security attributes of the intermediate server may be required to be different for different target groups (as some groups may be more trusted than others).

  Such different server delegate credentials are requested by applying *gss_modify_cred*( ) to the server's credentials in the normal way, but by specifying is_delegate rather than is_accessor in the **access_flags** field of the *gss_priv_attribute_set* input argument to *gss_modify_cred*( ). In this case, security contexts initiated by such an intermediate server must explicitly specify which delegate credentials are to be combined with the initiating user's credentials using the *gss_compound_cred*( ) operation. The *gss_compound_cred*( ) operation emits a credential handle which can be used in the same way as any credential in the subsequent *gss_init_sec_context*( ) operations.

2.3 Relationship of Extended GSS-API to ISO Work

Upper Layers Security

The extended GSS-API described here defines a set of operations that provide an implementation of the *system security function* concept as defined in the ISO / IEC 10745 DIS. Such system security functions are general in nature, and are not unique to the OSI Upper Layers, nor are they dependent on any specific communications protocols.

OSI Security Architecture

In general, system security functions are supportive of protocol-specific security communication functions and the implementation of OSI security services. The base GSS-API addresses many of the services defined in the ISO / IEC 7498-2 standard, but unlike the ISO / IEC 7498-2 standard, is limited to an identity-based authorisation model. The extended GSS-API includes the base GSS-API but also specifies extensions that can support alternative authorisation models (such as role-based), and richer forms of access controls (such as selective delegation of access rights).

Security Frameworks

The extended GSS-API is consistent with the ISO Security Framework by supporting the operational generic security services for generation, verification and acquisition of security information as defined in the ISO / IEC 10181-1 CD. Through these generic services, the extended GSS-API provides support for authentication exchanges, transfer of initiator access control information (ACI), and transfer of action access control decision information as described in the ISO / IEC 10181-2 DIS and the ISO / IEC 10181-3 CD without imposing constraints on the mechanism.

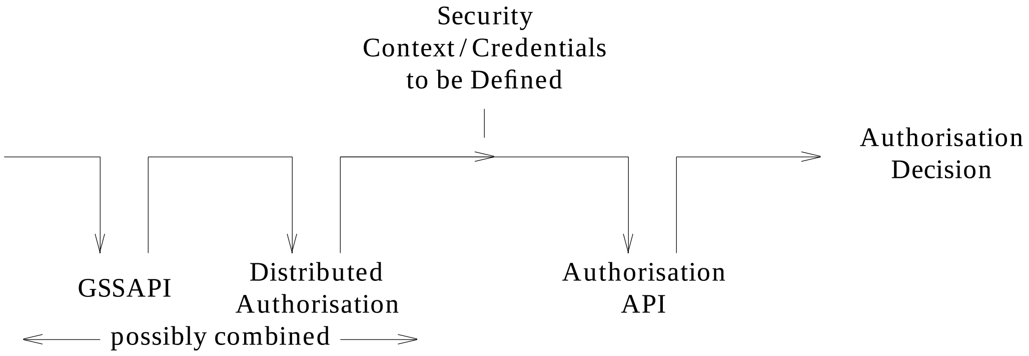
Generic Upper Layers Security

The data generated and processed by the extended GSS-API primitives is equivalent to the *Security Exchange Item Set* (SEIS) concept identified in the ISO / IEC 10745 DIS comments as part of AFNOR's comments on the ISO / IEC 10745 DIS, where each SEIS comprises a number of different and distinct instances of Security Exchange Information (SEI). The contents of an SEIS are specific to a security mechanism and are opaque to the communication infrastructure.

2.4 Relationship of GSS-API Extensions to IETF Work

One of the aims of the IETF Authorisation and Access Control WG (AAC WG) is to define an API that can support access control decisions for applications where authorisation decisions are local, but can later fit with distributed authorisation services.

The current expectation is that the output of the combined GSS-API and distributed authorisation services will be a security context (or a set of credentials) which is fed into the authorisation API. The diagram below shows the flow of control.



Note: The authorisation API provides mechanisms to query a local authorisation database to check authorisation.

Therefore, in order for a single combined interface to be provided that integrates authorisation support services into the GSS-API, the GSS-API security context and credentials need to be extended so that they can be used directly as input to the authorisation API. This enables the privileges and restrictions associated with the user's security context and credentials to be used as input to the access control decision.

Work is currently underway in the AAC WG to define the mechanism for extending the GSS-API security context and the authorisation API.

2.5 Relationship of DCE-specific APIs to GSS-API Extensions

Section 6 of the RFC 3 paper, **CHANGES TO THE PROGRAMMING MODEL** presents a set of APIs that enable application programmers to access the delegation functions expected to be present in DCE1.1.

2.5.1 RFC 3 API Summary

The APIs to query PACs and restrictions are:

sec_cred_get_initiator( )
sec_cred_get_opt_restrictions( )
sec_cred_get_req_restrictions( ).

The APIs to support delegation are:

sec_login_disable_delegation ( )
sec_login_become_delegate( )
sec_cred_get_delegate( ).

The RFC 3 paper APIs for the client, are partially superseded by the RFC 3+ paper:

set_login_become_initiator( ) (supersedes *sec_login_enable_delegation* ( ))
set_login_become_delegate( )
sec_login_become_impersonator.( )

2.5.2 Extended GSS-API and RFC 3

Basic Differences

The basic differences between the extended GSS-API and RFC 3 are as follows:

- The extended GSS-API operates on GSS-API credentials and security contexts, whereas RFC 3 operates on DCE login contexts.
- The extended GSS-API provides support for requesting privilege attributes by means of *gss_modify_cred*( ), whereas DCE-RFC 3 APIs only provide support for requesting groups by means of *sec_login_new_groups*( ).
- Extended GSS-API supports typed privilege attributes to facilitate portable server retrieval of audit and accounting identity and privileges, whereas DCE-RFC 3 APIs support DCE PACs.
- Credential restrictions can be applied on a per-target or per-target-group basis in the Extended GSS-API, whereas DCE-RFC 3 APIs specify the same login context restrictions for all targets.

Delegation Differences

The delegation differences between extended GSS-API and RFC 3 are as follows:

- To allow reuse of standard delegation controls, the Extended GSS-API splits the operation performed by a delegate to request delegation controls (*gss_modify_cred*( )) from the operation to combine the delegate and incoming credentials (*gss_compound_cred*( )); DCE-RFC 3 requires, in a way that reflects the DCE mechanism, these two operations to be combined (*become_delegate*( )).
- Extended GSS-API supports querying the target controls that apply to delegated credentials to check if a call to a security service will succeed; DCE-RFC 3 APIs provide no interface to query target controls.

- *gss_get_delegate_creds*( ) enables the returned list of delegate credentials to be passed easily to an authorisation service; the equivalent *sec_cred_get_delegate*( ) returns the delegate PACs one at a time.
- For simplicity, one API (*gss_modify_cred*( )) is used instead of three almost identical APIs (*become_initiator*( ), *become_delegate*( ) and *become_impersonator*( )).

2.5.3 Conclusions

In general, the RFC 3 APIs are environment-specific (through their use of DCE-specific types), so applications written to these DCE APIs could not function in (say) the Sun ONC environment. This dependence may be appropriate for applications that rely on other DCE services, but is not always desirable for applications written for portability across a wide range of distributed environments.

In addition, the above DCE-RFC 3 APIs can be constructed using the Extended Generic Security Service API together with parts of the API in the GSS-DCE paper. This is because an implementation that supports the generic API can also readily support the less generic one (although the inverse is not true).

Chapter 3

Extended GSS-API Data Types

The chapter specifies the data types required by the extended GSS-API. In a future version of this document, this chapter will be more descriptive.

```
typedef struct{
    gss_type_en     id_type;
    gss_value       id_value;}
    gss_id;

typedef enum{
    gss_oid_t,
    gss_integer,
    gss_string,
    gss_uuid,
    gss_buffer}
    gss_type_en;

struct union{
    gss_OID          OID;
    OM_uint32       *integer;
    char            *string;
    uuid_t          *uuid;
    gss_buffer_t     buffer;}
    gss_value;

typedef struct {
    OM_uint32        cred_count;
    gss_cred_id_t   *cred_list;}
    gss_cred_list;

typedef struct {
    OM_uint32        value_count;
    gss_value       *value_list;}
    gss_value_list;

typedef struct gss_attribute_desc {
    OM_uint32        validity_indicator;
    gss_id           attribute_type;
    gss_id           policy_authority;
    gss_type_en      value_syntax;
    gss_value_list *values_list;}
    gss_attribute;

typedef struct gss_id_set_desc{
    OM_uint32        id_count;
    gss_id          *ids;}
    gss_id_set;
```

```
typedef struct gss_attribute_set_desc{
    OM_uint32       attribute_count;
    gss_attribute  *attributes;}
    gss_attribute_set;

/* access_flags is used to specify whether the privileges
 * in the associated credentials apply to direct access or
 * delegation.  A server may own multiple accessor and delegate
 * credentials. A server’s delegate credentials are required when
 * a server initiates an outgoing security context on behalf of a
 * client whose credentials specify traced delegation.
 */
typedef struct gss_priv_attribute_set_desc{
    OM_uint32           access_flags;
       /* combination of one or more of the following: */
       /* - is accessor */
       /* - is delegate */
    gss_attribute_set*  priv_attributes;}
    gss_priv_attribute_set;

typedef struct {
    OM_uint32           time_count;
    time_t             *time_list;}
    gss_time_list;

typedef struct  {
    OM_uint32           control_count;
    gss_controls       *control_list;}
    gss_control_set;

typedef struct gss_controls_desc{
    OM_uint32          *check_back;
    OM_uint32          *usage_count;}
    gss_controls;

typedef struct target_control_desc{
    gss_attribute_set  *target_attributes;
    /* attributes characterising a group of targets
     * Two special values are defined for convenience:
     * - If GSS_C_NULL_ATTRIBUTE_SET is
     *   specified then all targets are included
     * - If GSS_C_SELF is specified or
     *   target_attributes->attribute_count == 0
     *   then the controls, attributes or restrictions
     *   specific to the caller are indicated.
     */
```

```
        gss_control_set     *controls_required;
        /* controls to be applied
         * - If GSS_C_NULL_CONTROLS is specified then default
         *   controls apply for all targets
         */

        gss_attribute_set   *optional_restrictions;
        gss_attribute_set   *required_restrictions;
        /* restrictions to be applied for the specified target(s) */
         * - If GSS_C_NULL_ATTRIBUTES is specified then default
         *   restrictions apply for all targets
         */

        gss_attribute_set   *requested_attributes;
        /* attributes requested (which may qualify the privileges
         * for the specified  target(s))
         * - If GSS_C_NULL_ATTRIBUTES is specified then default
         *   request attributes apply for all targets
         */

        OM_uint32            delegation_flags;
        /* sensible combinations of the following flags describe
         * target_attributes. The following are supported:
         * - none                          no delegation permitted
         * - can_be_delegate_and_target   may act as delegate and/or target
         * - trace_required                delegation must be traced
         * - trace_degrade_to_initiator   if trace unacceptable to target
         *                                 then use simple delegation
         * - trace_degrade_to_requester   if trace unacceptable to target
         *                                 then use delegate’s access rights
         */
        }gss_target_control;

    typedef struct {
        gss_time_list        valid_time_periods;
        OM_uint32            target_control_count;
        gss_target_control  *target_controls;}
        gss_target_control_set;
```

Chapter 4

Extended GSS-API C Reference Manual Pages

This chapter specifies the GSS-API C-language extensions in alphabetical order.

NAME

gss_compound_cred — combine specified credentials

SYNOPSIS

```
OM_uint32 gss_compound_cred(
    OM_uint32    *minor_status,
    gss_cred_id_t cred_handle_1,
    gss_cred_id_t cred_handle_2,
    gss_cred_id_t cred_handle_new
    );
```

DESCRIPTION

This function is used to compound credentials. It is useful when traced delegation is supported as it permits a server to combine a specific set of delegate credentials with a client's credentials.

The arguments for *gss_compound_cred*( ) are:

minor_status (out)
Mechanism-specific status code.

cred_handle_1 (in)
Handle to claimed credentials. The argument *cred_handle_1* refers to an authenticated principal.

cred_handle_2 (in)
Handle to claimed credentials. The argument *cred_handle_2* refers to an authenticated principal.

cred_handle_new (out)
Handle to a compound set of credentials.

RETURN VALUE

The following GSS status codes can be returned:

[GSS_S_COMPLETE]
The credentials have been compounded.

ERRORS

Failure results in a mechanism-specific error.

NAME

gss_get_attributes — get attributes associated with credentials or security context

SYNOPSIS

```
OM_uint32 gss_get_attributes(
    OM_uint32              *minor_status,
    gss_cred_id_t           cred_handle,
    gss_ctx_id_t            context_handle
    gss_id_set              attribute_types_required,
    gss_priv_attribute_set *attributes,
    gss_target_control_set *actual_target_controls
    );
```

DESCRIPTION

If the underlying mechanism supports it, this function allows a caller to enquire the current value of a nominated access control attribute type in the specified credentials and (optionally) context. The requested attribute value is returned together with the policy authority (if any). This function can be used by initiating clients, delegates and servers to query attributes in credentials or security contexts. If the *attribute_types_required* argument is not supplied, then all attributes are returned. This option could allow clients of this interface to obtain all attributes and pass these to a separate authorisation service to make a decision. If retrieval of access control attributes is not supported, or some other failure occurs, the *major_status* returned is mechanism-specific; otherwise *major_status* is [GSS_S_COMPLETE]. This interface permits mechanism-independent retrieval of trusted authorisation information, which, depending on the environment might be, for example:

- DCE — the user's, identity, group and in the future, other security attributes
- Kerberos — the user's access identity
- ECMA — the user's access identity, role and group membership; the audit identity; controls; restrictions.

Two important usages of this API in a DCE environment would be:

- audit-id retrieval
- use by ACL manager (instead of passing PAC by means of *rpc_binding_inq_auth_client*( )) to get privileges.

The arguments for *gss_get_attributes*( ) are:

minor_status (out)
Mechanism-specific status code.

cred_handle (in)
Handle to credentials. The argument *cred_handle* refers to an authenticated principal. This argument must be supplied if *context_handle* is not supplied. Supply GSS_C_NO_CREDENTIAL to use default credentials.

context_handle (in)
GSS-API security context handle. The argument *context_handle* refers to an established association. This argument must be supplied if *cred_handle* is not supplied, that is if:

```
cred_handle == GSS_C_NO_CREDENTIAL
```

Otherwise, that is if:

```
cred_handle != GSS_C_NO_CREDENTIAL
```

context_handle is ignored.

Note: Typically it is only necessary to use a *context_handle* argument rather than *cred_handle* for the case when a security context is emitted by *gss_accept_sec_context*( ), but not with an accompanying set of delegated credentials.

attribute_types_required (in)
A set of attribute types. If the default (GSS_C_NULL_OID_SET) is specified, all attributes are returned.

attributes (out)
A set of subject attributes. Response is conditional on the *attribute_types_required* input.

actual_target_controls (out)
The target controls. This argument is always returned.

RETURN VALUE

The following GSS status codes can be returned:

[GSS_S_COMPLETE]
The retrieval of attributes is supported and all, some or none of the requested attribute types have been returned.

ERRORS

Failure results in a mechanism-specific error.

NAME

gss_get_delegate_creds — get delegates (if any) associated with credentials or security context

SYNOPSIS

```
OM_uint32 gss_get_delegate_creds(
    OM_uint32     *minor_status,
    gss_cred_id_t  cred_handle,
    gss_ctx_id_t   context_handle,
    gss_cred_list *delegates
    );
```

DESCRIPTION

This function is for use when traced delegation is supported. It permits the retrieval of delegate credentials for use in authorisation decisions. The arguments for *gss_get_delegate_creds*( ) are:

minor_status (out)
Mechanism-specific status code.

cred_handle (in)
Handle to claimed credentials. The argument *cred_handle* refers to an authenticated principal. This argument must be supplied if *context_handle* is not supplied. Supply GSS_C_NO_CREDENTIAL to use default credentials.

context_handle (in)
GSS-API security context handle. The argument *context_handle* refers to an established association. This argument must be supplied if *cred_handle* is not supplied, that is if:

```
cred_handle == GSS_C_NO_CREDENTIAL
```

Otherwise, that is if:

```
cred_handle != GSS_C_NO_CREDENTIAL
```

context_handle is ignored.

Note: As for *gss_get_attribute*( ), typically it is only necessary to use a *context_handle* argument rather than *cred_handle* for the case when a security context is emitted by *gss_accept_sec_context*( ), but not with an accompanying set of delegated credentials.

delegates (out)
The argument *delegates* contains an ordered list of delegate credentials for each of the intermediate delegates (if any) between the initiating user and this target. It is expected that the normal use for such delegate credentials would merely be inspection by means of *gss_get_attributes*( ) as most known mechanisms would not permit such delegate credentials to be used for initiating further security contexts. It is the caller's responsibility to free any delegate credentials returned from *gss_get_delegate_creds*( ) by means of *gss_release_cred*( ).

RETURN VALUE

The following GSS status codes can be returned:

[GSS_S_COMPLETE]
Delegate credentials have been retrieved.

ERRORS

Failure results in a mechanism-specific error.

NAME

gss_modify_cred — request a set of privileges and controls

SYNOPSIS

```
OM_uint32 gss_modify_cred(
    OM_uint32              *minor_status,
    gss_cred_id_t           cred_handle,
    OM_uint32               duplicate_cred_req,
    OM_uint32               commit_cred_req,
    gss_priv_attribute_set  required_privilege_attributes,
    gss_target_control_set  required_target_controls,
    gss_cred_id_t          *output_cred_handle,
    gss_priv_attribute_set *actual_privilege_attributes,
    gss_target_control_set *actual_target_controls
    );
```

DESCRIPTION

This function requests a set of privileges and controls, optionally replacing existing credentials or creating a new set. The effect of this interface is cumulative on a set of credentials.

The form of the call is as follows: a set of privilege attributes is

requested, together with controls which qualify the unrestricted use of those attributes. The attributes that must be possessed by targets that can use the PAC may be specified; the default is all targets.

This service creates or modifies variant credentials if a mechanism supports it. This could permit access control restrictions to be applied by an authenticated entity to any security contexts created under these modified credentials. For example, this would allow a user to lower the clearance associated with a set of credentials, or to change the role. So, if a user's privilege attributes give a clearance of TOP SECRET, then when contacting a particular server they may wish to use this interface to lower their clearance to be UNCLASSIFIED. Normally such services would, respectively, constrain existing privileges, or result in new privileges being requested (if possible). However, whether or not this interface triggers a call on a privilege server is unspecified at the interface level.

Controls (usage constraints) on these privilege attributes may be requested by using the *required_target_control* argument. This argument can specify permitted security context targets, together with constraints on whether delegation to those targets is permitted. Additionally, this argument can be used to specify further controls such as the validity period for credentials, or that credentials must be revalidated at every target.

If setting or getting attributes is not possible, or not meaningful for this mechanism, the *major_status* returned by the function indicates failure. Otherwise, *major_status* is [GSS_S_COMPLETE], and *actual_privilege_attributes* is the set of privilege attributes associated with the *output_cred_handle* credentials. The current controls that apply to the specified credentials and context are returned in *actual_target_controls* and *actual_privilege_attributes*.

The arguments for *gss_modify_cred*( ) are:

minor_status (out)
Mechanism-specific status code.

cred_handle (in)
Handle for credentials claimed. The argument *cred_handle* refers to an authenticated principal. Supply GSS_C_NO_CREDENTIAL to use default credentials.

duplicate_cred_req (in)
TRUE for a variant credential set; FALSE means modify the original.

commit_cred_req (in)
TRUE for immediate attribute acquisition; FALSE means deferred attribute acquisition.

required_privilege_attributes (in)
A set of (typed) privilege attribute values. One of these may be a role name that is an attribute set reference used to select a set of attributes, which can include privilege and target ones and also PAC controls. To request default privilege attributes, specify GSS_C_NULL_ATTRIBUTE_SET.

required_target_control (in)
A set of attribute values for controlling which target applications can use the privileges in the PAC either directly or can forward the PAC to others for use by proxy. This can also restrict which initiators can call the target application. Controls on credential validity time can also be requested. For default target controls specify GSS_C_NULL_TARGET_CONTROLS.

output_cred_handle (out)
The function *gss_modify_cred*( ) produces a modified version of the input credentials (*cred_handle*). The original credentials are directly changed if *duplicate_cred_req* is FALSE; otherwise the *output_cred_handle* references a new, and potentially different, copy of the original input credentials (which remain untouched). The function *gss_release_cred*( ) can be used when the caller is finished with any new credentials created by this call.

actual_privilege_attributes (out)
A set of (typed) privilege attribute values referred to by *output_cred_handle*.

actual_target_controls (out)
A set of attribute values for controlling which target applications can use the access rights in the credentials referred to by *output_cred_handle*, or can forward them to others for use by proxy.

RETURN VALUE

The following GSS status codes can be returned:

[GSS_S_COMPLETE]
Attributes can be set, and all, some or none of the requested privileges and controls have been returned.

[GSS_S_NO_CRED]
Attributes cannot be set because there is no credential.

[GSS_S_DEFECTIVE_CREDENTIAL]
Attributes cannot be set because the credential is defective.

[GSS_S_CREDENTIALS_EXPIRED]
Attributes cannot be set because credentials have expired.

[GSS_S_BAD_MECH]
Attributes cannot be set because the mechanism does not support the requested functionality.

ERRORS

No other errors are defined.

NAME

gss_set_default_cred — set default for credentials

SYNOPSIS

```
OM_uint32 gss_set_default_cred(
    OM_uint32    *minor_status,
    gss_cred_id_t cred_handle
    );
```

DESCRIPTION

This function permits a user to indicate which credential is to be interpreted as the default. The arguments for *gss_set_default_cred*( ) are:

minor_status (out)
Mechanism-specific status code.

cred_handle (in)
Conditional handle for credentials claimed. The argument *cred_handle* refers to an authenticated principal. Supply GSS_C_NO_CREDENTIAL to use default credentials (which would mean that this call would be non-functional).

RETURN VALUE

The following GSS status codes can be returned:

[GSS_S_COMPLETE]
The default credential can be set, and has been set.

[GSS_S_NO_CRED]
The default credential cannot be set because there is no credential.

[GSS_S_DEFECTIVE_CREDENTIAL]
The default credential cannot be set because the credential is defective.

[GSS_S_CREDENTIALS_EXPIRED]
The default credential cannot be set because credentials have expired.

ERRORS

No other errors are defined.

Chapter 5

Access Control

This chapter discusses access control in distributed systems:

- Some of the concepts and models for access control as defined by X/Open publications and elsewhere are presented.
- An outline of existing implementations of access control in distributed systems is given.

Then portable access control software is considered:

- Requirements are stated.
- Examples are provided of the use of Extended GSS-API.
- Standard attribute types are defined.

5.1 Access Control in Distributed Systems

5.1.1 Previous X/Open Work on Access Control

Access control is one of the basic facilities required to implement a secure system. In recent years X/Open has published guides based on conventions in use by system administrators, which included consideration of access control and access control policy implementation, in particular:

- A general definition of access control is given in the **Security Guide** as ‘‘ensuring that authenticated users can gain access to the resources they need, and that they cannot gain access to other resources’’.
- Access control is discussed in Section 7.2 of the **Procurement Guide**, but this is confined to consideration of stand-alone systems.

5.1.2 Selected Distributed Access Control Concepts and Models

Access control in distributed systems is a more general problem than for stand-alone, centralised, multi-user systems. The framework defined by the XPG/POSIX.1 model with machine-specific security attributes associated with users (uid, gid), and limited operations permitted on objects (read, write, execute) is no longer sufficient in a large heterogeneous network.

A number of perspectives on access control have been developed by work in the standards arena, industry and research which basically generalise from the stand-alone system.

The terminology, concepts and basic models differ. Some examples are given below.

Access Control Definitions

Access control is defined in the ABLP-AC paper as ‘‘deciding whether the agent that makes a statement is trusted on this statement; for example, a user may be trusted (hence obeyed) when he says that his files should be deleted’’.

Access control is defined in the ISO/IEC 7498-2 standard as ‘‘the prevention of unauthorised use of a resource, including the prevention of use of a resource in an unauthorised manner’’.

In the TR/46 standard, access control is treated generally. It is stated that ‘‘there is a distinction to be drawn between the two complementary components of an access control policy: authentication and authorisation. The former describes the process of providing claims of identity, the latter describes the process of controlling access by already identified subjects to already identified protected objects in a system.’’.

ISO Access Control Framework

The primary goal of access control is defined in the ISO/IEC 10181-3 CD as ‘‘to counter the threat of unauthorised operations involving elements of a computer communications system; these threats are frequently subdivided into classes known as unauthorised use, disclosure, modification, destruction, and denial of service’’. A distinction is made by the ISO/IEC 10181-3 CD between the Access Decision Function (ADF), which implements access mediation, and the Access Enforcement Function (AEF), which enforces the outputs of the ADF. The ADF is informed by access control decision information (ADI) associated with the initiator, the action and the target, as well as by relevant contextual information.

ECMA Security Framework

In the ECMA Security Framework, authorisation attributes associated with the subject are called privilege attributes, and authorisation attributes associated with the object are called control attributes. A subject's request to access an object is either granted or denied based on the privilege attributes, control attributes, and the kind of access being requested.

Delegation

The ABLP-AC paper notes the requirement for delegation (that is, enabling one principal to act on another principal's behalf), and also discusses the motivation for enabling access decisions to be informed by the attributes of a compound principal constructed from the attributes of initiator and delegates. For example, where a workstation is treated as a delegate for a user, a file server may not let a known untrusted workstation read a user's files even if the user logged in on that workstation. The file server may also let a trusted node read files over which the user has, say, execute-only rights, as it trusts the workstation not to show these files to the user, but just to execute them.

Push and Pull Models

A distinction is made in the LABW-AC paper between the *pull* and *push* models of access control. The *pull* model requires the target recipient of an access request to look up in a database or directory to verify attributes such as group affiliations, and thus the consequent access rights. The *push* model is where a request to a target system must claim all the attributes that it asserts, *and* present the proof of its claims as well (typically having obtained the proof from an on-line trusted third-party server).

Regardless of which model is used, the same access control mediation procedure occurs once the user's attributes are *certified.* The LABW-AC paper proposes a conventional ACL. In an analogous way to POSIX.1 systems, this ACL is attached to each object and can be viewed as a set of principals, each with some rights to the ACL's object. In such a general model, a capability for an object can be viewed as a principal that is automatically on the ACL.

Restricted Proxies

The restricted proxy model in the N-RP paper extends the concept of delegated authenticated identity effectively to allow selective delegation of access rights. The model is that a proxy (delegation token) is given by a *grantor* (such as an authorisation server) to a *grantee* (client). The proxy normally contains selective restriction of the rights associated with the grantor. The *restricted proxy* enables the grantee to act as the grantor for the purpose of asserting the client's rights; for example, to access specific objects, or to assert group memberships. A target system can limit the access rights that it is prepared to permit an authorisation server to assert.

So, for a target system in a capability-based environment, an ACL for an object would simply contain an entry for the permitted grantor of access rights together with its own access rights. For conventional access control, an ACL associated with an object would contain subject security attributes together with the identity of the authorisation servers allowed to assert the security attributes.

5.1.3 Distributed Access Control Implementations

Architectures of secure distributed systems differ greatly in the mechanism for assigning privilege attributes, in the placement of ADF and AEF logic, and the access control determination algorithm. Some of these implementations are briefly discussed in this section, and summarised in Table 5-1 on page 32.

MIT Kerberos

In basic Kerberos, RFC 1510, a mechanism is provided which permits the identity of a subject to be authenticated to remote servers, and for the credentials of a subject to be delegated (in an unrestricted way). Once mutual authentication is established, facilities are provided for applications to prevent access control violations. This is achieved through the provision of continuing authentication by means of application data integrity or confidentiality. Extensibility is provided to enable authentication by Kerberos of a subject to the local system (and *vice versa*). Extensibility is also provided for the security server to assert authorisation information (but the internal syntax and usage of the authorisation data isn't addressed). Hence, for access control purposes, basic Kerberos provides the ability securely to transmit an authenticated identity (as an Internet or X.500 name). Kerberos is therefore limited to support of identity-based authorisation, unless authorisation is included in the ticket and can be interpreted by the target service.

OSF DCE

DCE Security, as defined in the AES-SEC paper, builds on the extension options provided by Kerberos. In DCE, subjects are granted privileges by a Privilege Server (PS). Kerberos-based mechanisms are used for the privileges to be securely transmitted to target systems. Privileges comprise principal, group and issuing cell (that is, domain) identifiers, and are represented as universal unique identifiers (UUIDs). Target system applications act as reference monitors (that is, comprise AEF and ADF components) for resources and provide access mediation and enforcement facilities. The ADF component is contained in the target application (although identified as a distinct ACL manager element). Although applications are not constrained to use it, the conventional access control algorithm is aligned with POSIX.6 in that it grants access rights according to the user-group-other precedence rule and effectively combines (logically ORs) access rights for groups. Using the POSIX.6 model for extensions, additional access control list entry types are added for ‘‘foreign’’ group and other privileges; that is, those which have not been asserted by the local PS. Also, specific permissions (not restricted to rwx) are defined for the different types of object supported in DCE. A DCE-conformant ACL manager may also support other access control entry types and application-specific permissions. The implementation of delegation in DCE1.1 makes the privilege attributes of delegates as well as initiators available to target systems. This allows the ACLs for resources to incorporate entries for delegates' access rights, thus enabling a distinction to be drawn between the rights a principal has when acting as an initiator and as a delegate

OSI Directory

The access control model in the directory assumes that authentication procedures will have established the accessor's identity (distinguished name and optional unique identity), and that the authentication procedure will itself have an associated authentication level, which is derived from the strength of the authentication mechanism. To perform directory operations that require access to attributes or attribute values, it is necessary to have entry access permission to the entry or entries that contain those attributes or values. When an attempt is made to perform an operation, the ADF (known in the ISO / IEC 9594-2 standard as the ACDF) is applied both for the named object (entry access) as well as for associated attributes (attribute access). Different permissions are defined for entry and attribute access. Whether or not access is granted depends on the access control information (ACI) associated with these protected items, and the prescriptive ACI (which applies to a subtree and therefore can be hierarchic or applied to only one entry). However, even this gives a simplified view because ACI can depend on authentication level and may also have an attribute that determines its precedence. Also, more than one ACI can apply to each entry and to each attribute or attribute value. In general the OSI Directory standard provides a sophisticated set of access controls, which are intended to accommodate the many different policy options a Directory might require. Unlike in the POSIX.6 or DCE model, a denial specified in ACI always overrides a grant; hence membership of a group that is denied access overrides membership of a group that is permitted access.

Novell NetWare 4

In NetWare, a security model similar to that of the Directory is implemented. An authentication proof is constructed which contains a number of security attributes including identity and workstation identifier. Access control is implemented in the NetWare Directory Service (NDS) for directory objects using an algorithm similar to that specified in POSIX.6 D13 except that access is granted based on a union of the permissions associated with the subject's privileges. Hence membership of a group with access to a resource grants that access even if the user identity has a more restricted access.

Trusted Mach

In Trusted Mach (see the TMACH draft standard), each object has a label and an ACL and every subject (task) has a label range and discretionary access identifier (DAC ID). The DAC ID corresponds to a combination of user and group names. In order to obtain access to an object the object must be accessible to the client task both with respect to its DAC ID and the ACL of the object, and with respect to its label range and label of the object. The access mediation procedure (ADF) is separated from the enforcement mechanism (AEF). The mediation is generalised to be the same for all objects, and performed by the name server as part of a request by the client to *open* an object. The enforcement (that is, granting of access to an object) is target-specific and is implemented by the target that encapsulates the object.

UNIX System V Release 4 Enhanced Security

In a similar manner to Trusted Mach, UNIX System V Release 4 Enhanced Security (SVR4ES) provides support for discretionary access control and mandatory access control. Access controls are implemented by the UNIX kernel.

SecureWare MaxSix

In MaxSix, trusted networking software is implemented such that TCSEC B1 and CMW computing environments can be supported. The network can transport security attributes such as sensitivity labels, information labels, audit user ids, and user privilege sets. Software access to these attributes is permitted through extensions to the sockets interface. This permits servers to enforce access checks on their objects; for example, trusted X servers enforce access controls on windows.

SESAME

In SESAME, privileges are assigned to a subject by a Privilege Attribute Server based on the subject's requested or default job role. At target systems the access control decision functionality is partitioned between the SESAME trusted PAC Validation Facility (PVF) and GSS-API security, run-time library, application-specific access controls. The validation of the PAC encapsulates authentication of the initiator, checking that a PAC (or chain of PACs) is valid and that the issuing Privilege Attribute Server is trusted. Then the security-context management software under the GSS-API may use locally defined policy to check that the subject is authorised to initiate a security context with this application. The application or its ADF sub-component can then obtain the attributes to implement its application-specific access control checks.

Privilege Attribute Types	**K**	**D**	**D1**	**O**	**N**	**T**	**U**	**W**	**S**
identity privilege attribute	X	X	X	X	X	X	X	X	X
group privilege attributes		X	X	X	X	X	X	X	X
role privilege attributes			X		X				X
legacy/application privilege attributes			X						X
Available Mechanism for Obtaining Privilege Attributes	**K**	**D**	**D1**	**O**	**N**	**T**	**U**	**W**	**S**
push	X	X	X			X		X	X
pull		X	X	X	X				
Access Control Type	**K**	**D**	**D1**	**O**	**N**	**T**	**U**	**W**	**S**
mandatory access control							X	X	
discretionary access control	X	X	X	X	X	X	X	X	X
Delegation Facilities Provided	**K**	**D**	**D1**	**O**	**N**	**T**	**U**	**W**	**S**
no delegation				X	X	X	X	X	
impersonation	X		X						X
simple delegation controls									X
target/group delegation controls			X						X

Table 5-1 Current Access Control Systems

5.1.4 Different Authorisation Models

For all distributed security systems the facility for two peers to communicate privilege attributes is supported — even if the privilege attribute communicated is only an authenticated identity. Different mechanisms are used for establishing a user's access control attributes.

- Security attributes can be sent from client to server in full binary form (as in Kerberos, DCE and SESAME).
- Security attributes can be sent in a tokenised form and mapped according to universal or bilateral conventions (MaxSix).
- Security attributes can be pulled by the target system (as in the DASS architecture).
- Security attributes can be sent in advance from a trusted third party (such as the TMach Root Name Server).

5.2 Portable Access Control Software

5.2.1 Requirements

Standard API

For a portable system, no assumption can be made about the mechanism for establishing access control attributes (that is, privilege and control attributes). There are three levels to this portability:

a. initialising security contexts with associated, possibly opaque, access control information

b. using an authorisation facility that takes that access control information as input

c. interpreting that access control information.

The extended GSS-API builds on the base GSS-API model of credentials and security contexts by permitting request or retrieval of extended attributes and controls, and additional operations on them.

The extensions proposed for attribute operation address portability issues a and b above. Issue c above is an issue that requires further study.

Standard Attributes

For portable software to work with the existing distributed access control implementations, or others yet to be defined, there must be a standard taxonomy of access control information.

Therefore, in addition to a standard API, there must also be an agreed set of access control attribute types and an agreement on their syntax.

5.2.2 Summary of Extended GSS-API

The functions to set and query privilege attributes and controls are:

- *gss_modify_cred*( )
- *gss_get_attributes*( ).

The functions to support traced delegation are:

- *gss_compound_cred*( )
- *gss_get_delegate_creds*( ).

The function to initialise default credentials is:

- *gss_set_default_cred*( ).

5.2.3 Concepts of Extended GSS-API Model

All security attributes are specified by:

- type
- value
- defining authority.

Credentials are associated with:

- identity (for example, audit identity, accounting identity)
- privilege attributes (for example, access identity, group, role, clearance)
- target controls (for example, permitted targets, restrictions, delegation limits).

Security contexts' attributes are derived from credentials.

5.2.4 Operations to Support Extended GSS-API Model

Security attribute request and retrieval operations are:

- Query credentials' privileges and controls.
- Produce variant credentials with specified privileges and controls.
- Constrain use of privileges to targets or groups of targets.
- Apply restrictions to specific targets or groups of targets.

Delegation support operations are:

- Selectively delegate privileges to specific targets or groups of targets.
- Retrieve delegates' attributes.
- Compound client credentials with specified delegate credentials.

5.2.5 Examples of Use of Extended GSS-API for Access Control

Example 5-1 Simple Role-based Access Control Policy

Use of role-based access control (RBAC) permits a server to make access control decisions which reflect a static attribute such as administrative role, rather than using the more transient identities of subjects. This minimises the administrative load in maintaining an access control policy.

To implement software which uses RBAC using the extended GSS-API an application server would contain logic represented using the following pseudo-code:

```
/* wait for connection indications */

/* authenticate the caller */
gss_accept_sec_context
   token                 - opaque authentication exchange token(s)
   h                     - returns h, a handle to a security context

/* query the caller's role attribute */
gss_get_attributes
   h                     - security context
   role_attribute_type   - role attribute type
   caller_role_attribute - returns a role attribute if one exists

/* check if the caller is an administrator by comparing the value of
   the returned attribute with "administrator" */
```

Example 5-2 Document Label-based Access Control Policy — with Delegation

In organisations where control of access to documents is important, there may be requirements to have access to each document constrained by the owner's clearance.

This example considers the implementation of both print and file servers. The design is such that the print server calls on a file server to obtain the required file, and the file server controls read access to documents based on the user's confidentiality class privilege attribute. The servers contain logic represented using the following pseudo-code:

```
/* Print Server */

/* wait for print request */

/* authenticate the caller */
gss_accept_sec_context
   token                   - opaque authentication exchange token(s)
   h                       - returns h, a handle to a security context
   c                       - returns c, a handle to delegated
                             credentials

/* establish a security context with the file server
   using delegated credentials */
gss_init_sec_context
   file_server             - identity of file server
   c                       - a handle to delegated credentials
   token                   - returns opaque authentication exchange
                             token(s)

/* send token(s) to file server */

/* File Server */

/* wait for file retrieval request */

/* authenticate the caller */
gss_accept_sec_context
   token                   - opaque authentication exchange token(s)
   h                       - returns h, a handle to a security context

/* query the caller's confidentiality class */
gss_get_attributes
   h                       - security context
   conf_class_attr_type    - confidentiality class attribute type
   caller_conf_class_attr  - returns a confidentiality class attribute
                             (if one exists)

/* check if the requested file's confidentiality class is present in
   the caller's confidentiality class attribute */
```

Example 5-3 Document Labelling Policy — with Traced Delegation

If the previous example is extended such that the print server calls on a file server to obtain the required file, and traced delegation is used, then additional checks are possible at the file server to verify that the print server is trusted to receive the document.

The implementation of the file server would be extended to distinguish between the initiator’s and delegate’s attributes as shown in the following pseudo-code:

```
/* File Server */

/* wait for file retrieval request */

/* authenticate the caller */
gss_accept_sec_context
   token                  - opaque authentication exchange token(s)
   h                      - returns h, a handle to a security context

/* query the delegate(s) involved in setting up the context */
gss_get_delegate_creds
   h                      - security context
   dh                     - returns the handle to the print server
                            credentials

/* query the initiator’s sensitivity label range */
gss_get_attributes
   h                      - security context
   conf_class_attr_type   - confidentiality class attribute type
   caller_conf_class_attr - returns a confidentiality class attribute
                            (if one exists)

/* query the delegate’s sensitivity label range */
gss_get_attributes
   dh                     - print server’s credential handle
   conf_class_attr_type   - confidentiality class attribute type
   dh_conf_class_attr     - returns a confidentiality class attribute
                            (if one exists)

/* check if both the initiator’s and delegate’s label ranges permit
   access to the file */
```

5.2.6 Standard Attribute Types

There are four classes of attributes that relate to access control: identifiers, privilege attributes, controls and restrictions.

- Identifiers are attribute types allocated to the subject that are not used in authorisation.
- Privileges are attribute types allocated to the subject that are used in implementing access control and information flow policies.
- Target controls are attribute types used to identify target systems or groups thereof.
- Restriction attributes contain additional information that may restrict an access decision.

Object identifiers and standard syntax are defined for each of these attributes, and they are represented at the API as the type **gss_attribute**.

The main attributes are as follows. These can be arbitrarily extended through the use of object identifiers.

Identifier Types

authenticated	identity granted by authentication
non-repudiation	identity used for non-repudiation
audit	identity used for audit
charging	identity used for accounting when billing
certificate-id	unique instances number for attributes
audit-level	level of audits required for subjects' actions
audit-mask	audit event mask for subject
session-identity	[unique] identifier of session
location-identity	identifier of workstation
home-domain-name	identifier of domain of attribute issuer

Privilege Attribute Types

access-identity	subject id used for access control
role-name	role (attribute set reference)
role	job role attribute
primary-group	main group to which subject is affiliated
group	a group of which subject is a member
capability	attribute conferring access rights to a security object
application-defined	application-specific access control attribute
confidentiality-class	set of confidentiality classes permitted
minimum-confidentiality-class	minimum confidentiality class permitted
confidentiality-hierarchy	highest hierarchical confidentiality level
min-confidentiality-hierarchy	lowest hierarchical confidentiality level

integrity-class	set of integrity classes permitted
minimum-integrity-class	minimum integrity class permitted
integrity-hierarchy	highest hierarchical integrity level
minimum-integrity-hierarchy	lowest hierarchical integrity level
need-to-know	permitted set of need-to-know category labels
minimum-need-to-know	minimum set of need-to-know category labels
session-MAC-range	set of MAC labels permitted for session
process-MAC-range	set of MAC labels permitted for process

Target Control Types

target-name	identity of target system
trust-group	identity of target system group

Restriction Types

optional-restriction	an access control restriction to be applied if understood by the application
mandatory-restriction	an access control restriction that must be understood by the application, or else access should be denied
targeted-restriction	an access control restriction that is specific to a named application

Chapter 6

PACs and Authorisation Services

This chapter is derived from Appendix A of RFC 1508.

Current Position

Consideration has been given to modifying the GSS-API service interface to recognise and manipulate Privilege Attribute Certificates (PACs) as in ECMA 138, carrying authorisation data as a side effect of establishing a security context, but no such modifications have been incorporated at this time. This chapter provides rationale for this decision and discusses compatibility alternatives between PACs and the GSS-API that do not require PACs to be made visible to GSS-API callers.

Rationale

Existing candidate mechanism types such as Kerberos and X.509 do not incorporate PAC manipulation features, and exclusion of such mechanisms from the set of candidates equipped to support fully the GSS-API seems inappropriate. Inclusion (and GSS-API visibility) of a feature supported by only a limited number of mechanisms could encourage the development of ostensibly portable applications, which would in fact have only limited portability.

The current situation, in which PACs are not visible across the GSS-API interface, does not preclude implementations in which PACs are carried transparently, within the tokens defined and used for certain mechanism types, and stored within peers' credentials and context-level data structures. While invisible to API callers, such PACs could be used by operating system or other local functions as inputs in the course of mediating access requests made by callers. This course of action allows dynamic selection of PAC contents, if such selection is administratively-directed rather than caller-directed.

In a distributed computing environment, authentication must span different systems; the need for such authentication provides motivation for GSS-API definition and usage. Heterogeneous systems in a network can intercommunicate, with globally authenticated names comprising the common bond between locally defined access control policies. Access control policies to which authentication provides inputs are often local, or specific to particular operating systems or environments. If the GSS-API made particular authorisation models visible across its service interface, its scope of application would become less general. The current GSS-API paradigm is consistent with the precedent set by Kerberos, neither defining the interpretation of authorisation-related data nor enforcing access controls based on such data.

The GSS-API is a general interface, whose callers may reside inside or outside any defined TCB or NTCB boundaries. Given this characteristic, it appears more realistic to provide facilities that provide value-added security services to its callers than to offer facilities that enforce restrictions on those callers. Authorisation decisions must often be mediated below the GSS-API level in a local manner against (or in spite of) applications, and cannot be selectively invoked or omitted at those applications' discretion. Given that the GSS-API's placement prevents it from providing a comprehensive solution to the authorisation issue, the value of a partial contribution specific to particular authorisation models is debatable.

Glossary

For common computing terms refer to the glossary in the **Procurement Guide**. For base GSS-API terms refer to the **Base GSS-API** specification.

accessor
The initiating client application which is seeking access to a server's resources.

ACL
A list associated with a resource specifying the principals and their access rights for specific operations on the resource.

client
In the context of a distributed system, a communicating peer which requests services available from a (server) peer.

impersonation
Transmission of an initiator's identity and privileges such that neither the identities nor privileges of the intermediate participants (delegates) in a chain of operations are visible to the target server.

OID
Object Identifier. A uniquely defined value (expressed as an ASN.1 construct) which, in the case of GSS-API, is used by the caller to select an underlying security mechanism.

PAC
Privilege Attribute Certificate. A collection of principal information including the principal's identity, privilege attributes used for access control, control attributes that constrain use of the PAC, and a cryptographic seal which identifies the originating authority of the PAC and prevents tampering. The PAC is used by an authorisation system when determining if access to a resource should be granted.

privilege attributes
A principal's security attributes, such as access identity, groups and role, used by an authorisation system when determining if access to a resource should be granted to that principal.

proxy
Same as delegation.

pull model of access control
An access control model for distributed systems where the target recipient of an access request is required to query a database or directory to obtain or verify attributes sent in the request and the consequent access rights.

push model of access control
An access control model for distributed systems where a request to a target system claims all of the attributes that it asserts and presents proof of its claims as well.

role
A named set of privilege attributes associated with a principal's job function and organisation affiliation (for example, government pay clerk).

server
In the context of a distributed system, a communicating peer which provides services to a requesting (client) peer.

simple delegation

Delegation in which the target server makes access control decisions based only on the initiating principal’s privilege attributes without regard to the intermediaries’ privilege attributes (which are not transmitted to the target server).

traced delegation

Delegation in which the privilege attributes of delegating intermediate servers, in addition to the privilege attributes of the initiator, are provided to the target server for use in making access control decisions.

Index