DISTRIBUTED APPLICATION SECURITY USING PRAESIDIUM AS

Paper #2011
Charles Knouse
Early Adopter Program
General Systems Division
19111 Pruneridge Avenue, MS 44L7
Cupertino, CA 95014
cwk@cup.hp.com

INTRODUCTION

Security, always an important requirement for applications, becomes even more critical and complex when an application is distributed over many computers on a network. This paper examines requirements for distributed application security and presents a new security product from Hewlett-Packard, the Praesidium Authorization Server (AS). Praesidium (from the Latin word for *fortress*) provides security services for applications that operate in a variety of distributed environments, such as the Distributed Computing Environment (DCE). The primary value of Praesidium AS is authorization of requests from clients to servers, using application-defined rules. Praesidium AS also provides a uniform interface to security services of the environment, including authentication and data protection. The Praesidium AS product includes servers to manage application security data, libraries, and tools for the administration and maintenance of the security data.

DISTRIBUTED SECURITY REQUIREMENTS

The security requirements for distributed applications can be classified as authentication, authorization, and data protection.

- **Authentication**
  Authentication asks the question "Who are you?" It identifies who is using the application. Typically this is done using some form of log-in, where users supply public names and private passwords to prove they are who they say they are. This is sufficient for a local operating system or application, where the password stays within the system. But in a distributed environment, the log-in password should not be transmitted in the clear over the network, in order to prevent it from being intercepted by an eavesdropper. There are network authentication methods available that solve this problem. The Kerberos authentication method is one of the most popular. It is used by DCE and will be described later.

- **Authorization**
  Authorization asks the question "What can you do?" Once a user is authenticated, an application needs to determine what types of operations the user is authorized to perform. A familar example is the permissions mask in the Unix file system, which indicates whether a user can read, write, or execute a file. Access control lists (ACLs) are a more sophisticated technique used in DCE. Praesidium AS provides a powerful authorization method using entitlement rules. This will be discussed in detail later.

- **Data Protection**
  Data protection determines "Can my data be seen or altered?" When data is sent over the network, it can be intercepted by an eavesdropper. Using encryption methods, the data can be rendered unintelligible to anyone but the sender or the intended receiver. This is called *data privacy*. An attacker might also attempt to surreptitiously alter data being transmitted over the network. To prevent this, an encrypted checksum can be attached to the data to determine if the data has been altered. This is called *data integrity*.

DISTRIBUTED APPLICATION ENVIRONMENTS

Praesidium AS can work with a number of distributed environments, including DCE, the Encina Transaction Processing Toolkit, from Transarc Corporation, and the Generic Security Service Application Programming Interface (GSS-API), an Internet standard. Each of these environments provides security services that can be used through Praesidium AS.

Distributed Computing Environment

The Distributed Computing Environment was developed by the Open Software Foundation (OSF) and is implemented on a wide variety of platforms. DCE provides a comprehensive set of services for distributed applications based on remote procedure calls (RPCs). An RPC is a request from a client to a server to perform an operation with input data. The server executes the operation and responds with the resulting output data. The packaging and transmission of the request and response are done automatically by DCE, using client and server stub modules generated by the DCE IDL compiler. Security between a client and a server in DCE is also based on RPCs.

Authentication

DCE uses the Kerberos authentication method, developed at M.I.T. This technique uses *tickets*, which are data structures that securely encode the identity of a *principal* (a user or program). Tickets are generated by the DCE Security Server. Each principal has a public name and a private key derived from a password that is known only to the principal and the Security Server. Tickets contain information that is encrypted with these keys to verify that the principal knows the password. The formats and protocols

used by Kerberos are quite complex; see section 6.8 of my book **Practical DCE Programming** (Prentice Hall PTR) for more details.

A user authenticates himself by executing the `dce_login` command, specifying his principal name and password. This generates a ticket that identifies the user. When the user runs a client program, that program usually inherits the user's identity. A server authenticates itself through a series of DCE security calls, which result in a ticket that identifies the server. Before the client makes its first RPC to the server, it requests another ticket to convey its identity to the server. This service ticket is attached to the first RPC sent from the client to the server. The server can then retrieve the client's identity from the service ticket.

Authorization

DCE provides *access control lists* (ACLs) for authorization. Each ACL is a list of principals with access permissions granted to each principal. An application server can match the client's authenticated identity to its ACLs to determine what operations the client can perform.

Originally, a DCE application had to provide most of the code necessary to set up and use ACLs. This was a considerable burden on application writers, which deterred the use of ACLs for many applications. OSF Release 1.1 of DCE (which corresponds to HP-UX DCE 1.4) includes an ACL Management Library that simplifies setting up and using ACLs. The server still needs to make a number of DCE calls to do these tasks, however. Praesidium AS provides a more powerful alternative to ACLs that is easier to use and manage.

Data Protection

DCE incorporates data protection into RPCs. A client can choose the level of data protection to be used for its RPCs, from no protection to data integrity and privacy. The server can inspect the protection level of incoming RPCs and reject those RPCs that do not meet a minimum level. Data integrity and privacy use an encryption key that is generated by the DCE Security Service and included in the service ticket sent from the client to the server.

Encina

The Encina Toolkit provides an environment for distributed transaction processing applications. Encina is built on top of DCE, so it uses many of the DCE security facilities, including Kerberos authentication. Encina uses transactional RPCs (TRPCs) that are built on top of DCE RPCs. Consequently, TRPCs can use the RPC data protection levels.

Generic Security Service API

The Generic Security Service Application Programming Interface (GSS-API) is a standard interface for security services, including authentication and data protection. Underneath GSS-API is a security package, such as Kerberos, that provides the actual authentication and data protection services. The OSF Release 1.1 of DCE provides GSS-API on top of the DCE/Kerberos services.

Unlike DCE, GSS-API does not provide the transmission of data between a client and a server. An application using GSS-API for its security must use another service, such as Sockets, to transmit messages between clients and servers. GSS-API also does not provide any authorization services.

AUTHORIZATION WITH ENTITLEMENTS

Praesidium AS provides authorization for DCE, Encina, and GSS-API using the *entitlement model*. This model provides more flexibility and power than the ACLs used with DCE. Important elements of this model are entitlements, privileges, profiles, and principals.

Entitlements

An *entitlement* is a condition that must be satisfied before an operation can be authorized to proceed. The condition is expressed as a *rule*, which compares the values of attributes associated with the entitlement. *Transaction attributes* are values supplied by the application and based on the requested operation. *Privilege attributes* are provided by Praesidium and based on the user's identity; they specify limits or requirements for the user. *Environment attributes* are pre-defined by Praesidium and provide information on the application's execution environment, like the time of day.

Consider as an example a simple banking application, with a `withdrawal()` operation that specifies (among other things) a type of account and an amount to withdraw from the account. This operation might have an entitlement named `WITHDRAWAL`, with the rule

```
ACCT_TYPE = VALID_ACCT_TYPE AND AMOUNT <= LIMIT
```

The transaction attributes for this entitlement are `ACCT_TYPE`, a character string that specifies the type of the account, and `AMOUNT`, an integer that gives the withdrawal amount. The privilege attributes are `VALID_ACCT_TYPE`, which defines the accounts from which the user can make a withdrawal, and `LIMIT`, which defines the maximum amount the user can withdraw from each type of account. The requested withdrawal must then be from a valid account and less than or equal to the limit for that account.

Privileges

A *privilege* is a set of values for the privilege attributes of an entitlement. A user can have one or more privileges for an entitlement that define the limits or requirements for that user. In the `WITHDRAWAL` example, a user might have privileges like

```
WITHDRAWAL:VALID_ACCT_TYPE='CHECKING',LIMIT='300'
WITHDRAWAL:VALID_ACCT_TYPE='SAVINGS',LIMIT='1000'
```

This states that the user can withdraw up to $300 from his checking account and up to $1000 from his savings account.

Entitlements and privileges are stored in the Praesidium AS databases. An entitlement can be *evaluated* for a user and for a set of transaction attribute values by plugging in the user's privilege values until one set of privilege values makes the entitlement rule true, or until all of the user's privilege values have been tried.

Profiles

A *profile* is a collection of privileges that can be defined for a class of users. All users within the class can be assigned the profile and then inherit privileges from the profile. Continuing with the banking example, there could be a profile `STD_CUSTOMER` that includes the `WITHDRAWAL` privileges listed above. Each standard customer can then be assigned the `STD_CUSTOMER` profile. If the bank decided to increase the checking withdrawal limit for its standard customers, it need only change the profile privilege. Another profile `PREF_CUSTOMER` could be set up for preferred customers, with higher withdrawal limits.

Profiles can contain other profiles. A user assigned a profile inherits all of the privileges from all of the nested profiles. This process is called *unraveling*. There are rules for resolving conflicting privileges from nested profiles.

Principals

Each user is assigned a *principal* identity that is used to retrieve privileges for the user. With DCE and Encina (and with the current implemention of GSS-API), this principal corresponds to the DCE principal for the user. The DCE principal has a name and a UUID (Universal Unique Identifier). Praesidium uses the principal UUID as a key for privileges stored in its databases.

Enabling

Entitlements, privileges, and principals must be *enabled* to be effective. There are two pieces of information that determine if an item is enabled: a flag that specifies "enabled" or "not enabled", and the beginning and ending dates and times for which the item is enabled. It is then possible to set up entitlements, privileges and principals

that are in effect only during a certain time range. Profiles do not include information regarding enabling; they are always in effect.

ADMINISTRATION

Praesidium AS provides several tools for the creation, modification, and deletion of entitlements, privileges, profiles, and principals. These tools send requests to the Praesidium AS servers to carry out the administrative tasks. The Architecture section at the end of this paper outlines the servers that are involved.

odss_admin

The `odss_admin` program provides a Motif graphical user interface (GUI) for Praesidium administration on HP-UX workstations. (The `odss_` prefix comes from an earlier name for the software: Open Distributed Security Server. It shows up in the names for various program and functions.) The GUI includes windows for viewing, creating, enabling, and deleting entitlements, privileges, profiles, and principals. It also allows you to track the state of pending and completed requests. `odss_admin` is the principal Praesidium tool used by security administrators.

authu_batch

The `authu_batch` program provides a command line user interface for the execution of most of the Praesidium administrative tasks. (The `authu_` prefix comes from the server that executes the requests, as discussed in the Architecture section later.) The commands are usually entered into a text file script that is input into `authu_batch`. This tool is intended for repetitive tasks like creating a common set of entitlements and privileges. It is also used for dumping and restoring the Praesidium AS databases.

Administration Authorization

A user must be authorized to make administrative requests through `odss_admin` or `authu_batch`. There is a large set of entitlements defined by Praesidium for this purpose. For example, a user must have a privilege for the `ODSS_CREATE_ENT` entitlement to be able to create an entitlement. There is a pre-defined profile, `ODSS_ADMIN_PROFILE`, which contains privileges for all of the administrative entitlements. A user can then be assigned this profile to be given authority to perform all of the administrative tasks. The `cell_admin` principal, which is defined by DCE as a kind of network superuser, is initially assigned the `ODSS_ADMIN_PROFILE`. Then `cell_admin` can assign this profile and privileges to other users as necessary. We recommend that you assign administrative authority to other principals as soon as possible to minimize the use of the powerful `cell_admin` principal.

Maker/Checker Approval

Administrative requests submitted by one user (called the *maker*) normally need to be approved by another user (called the *checker*) before they take effect. This dual-control of security administration prevents one user from subverting security, say, by giving himself more privileges then he is warranted. When a maker issues a request, it is put on a queue of requests awaiting checking. A checker, using `odss_admin`, can then review the queue of pending requests and approve or deny each request. The checker can also modify parameters of the request and resubmit it. Then the resubmitted request must be checked again, by someone other then the original checker.

A checker must have privileges to the appropriate entitlements to be able to check requests. There are pre-defined entitlements for checking each type of request. For example, the `ODSS_CK_CRT_ENT` entitlement authorizes a user to check requests for creating entitlements. The `ODSS_CHECKER_PROFILE` contains the privileges for all of the check entitlements. The `ODSS_ADMIN_PROFILE` includes this profile, so administrative users can also check requests.

The Maker/Checker facility is normally enabled. It can be selectively or completely disabled using the `odss_ admin` Check Table window. To do this, a user must have privileges for the `ODSS_USE_CHK_TBL` or `ODSS_UPD_CHK_TBL` entitlements.

Audit Trail

Praesidium AS maintains a log of all changes to security data for auditing. Also, each authorization performed by Praesidium AS is included in this audit trail. The audit trail can be connected to HP's OpenView network management facilities, so security events can be monitored and alarms can be raised.

odss_query and odss_query_batch

The `odss_query` program provides a Motif GUI for testing entitlement evaluations. This is intended for use by an entitlement developer or a security auditor. The user enters the entitlement name, a principal name, transaction attribute values, and (optionally) a date and time. `odss_query` sends a request to a Praesidium AS server to evaluate the entitlement rule using those parameters. The `odss_query_batch` program provides the same service with a command line interface. To use these tools, a user must have a privilege for the `ODSS_CHECK_EVAL` entitlement.

APPLICATION PROGRAMMING INTERFACE

Applications that use Praesidium AS call functions in the ODSS application programming interface (API). Some of these calls encapsulate and simplify security services in the underlying environment, such as DCE authentication. Other calls provide access to the Praesidium AS authorization facilities.

Initialization

There are several set-up calls for the ODSS API. These include a variable list of pairs of input parameters that specify option keys and argument values.

- `odss_set_environment()`
  Sets general options, such as the program type (client or server), control of interface or operation checking (described in the Extension section of this paper), and communication timeout values. (Some of these options are really specific to DCE, but they are in this call for historical reasons.)

- `odss_set_dce_environment()`
  Declares that the DCE environment is in use. There are no DCE-specific options.

- `odss_set_encina_environment()`
  Declares that the Encina (and hence DCE) environment is in use. Encina options include transaction naming and out-of-bound communications.

- `odss_set_gss_environment()`
  Declares that the GSS-API environment is in use. The only GSS-API option is the connection behavior between the application and the Authorization Server.

- `odss_init()`
  Initializes the ODSS runtime in the application, using options specified in the environment setup calls.

- `odss_terminate()`
  Terminates use of the ODSS runtime, releasing any resources in use.

Authentication and Data Protection

These calls set up the application identity and other security parameters to be used by the application client and server. They encapsulate the calls in the underlying environment that do the real work.

- `odss_establish_context()`
  Establishes the identity (*login context*) for the application. For a client, the login context is usually inherited from the user's `dce_login`. For a server, the login context is usually set up using a principal name and key from a *keytab file* specified in a parameter to this call.

- `odss_release_context()`
  Releases a login context, including any resources used by it.

- `odss_register_server_info()`
  Specifies security information to be used between a DCE or Encina client and server. This includes the server principal name and the data protection level(s) to be used in RPCs between the client and the server.

Authorization

Three of these calls are used to evaluate an entitlement using transaction attribute values from the application. The other two retrieve privilege attribute values for an entitlement.

- `odss_authz_eval()`
  Evaluates an entitlement for the application's identity and a set of transaction attribute values. The function return indicates if the authorization represented by the entitlement is granted or denied. The application can then take appropriate action based on the return.

- `odss_caller_authz_eval()`
  Evaluates an entitlement for the initiator of a remote call to a server. Transaction attribute values are usually extracted from the remote call's input parameters. The function return indicates if the authorization represented by the entitlement is granted or denied. Based on the return, the server can either allow the call to proceed or reject it as unauthorized.

- `odss_set_caller()`
  Sets the identity of the caller to be used in `odss_caller_authz_eval()`. This is used only with the GSS-API environment. This is done automatically for DCE and Encina, as discussed in the Extensions section below.

- `odss_inq_entitlements()`
  Returns the privileges granted to the application principal for one or more entitlements. This call allows an application client to determine what kind of things it is authorized to do before it actually makes any requests. The client may then disallow menu choices for actions that are not authorized.

- `odss_inq_interfaces()`
  Returns privileges for the `FUNCTION_ACCESS` entitlement (discussed in the Extensions section below), for specified interfaces. This is a special case of the `odss_inq_entitlements()` call.

EXTENSIONS FOR ENVIRONMENTS

Praesidium AS integrates its security functions as much as possible with the environment. For example, authorization of a remote procedure call can be done

automatically, without requiring the application developer to explicitly code a call to an ODSS function. To do this, Praesidium AS uses *extensions*, software modules that provide an interface between an environment and the ODSS runtime. There are extensions for the DCE and Encina environments. There is no extension for the GSS-API environment.

Function Authorization

One of the primary tasks for the DCE and Encina environments is automatic authorization of functions. For DCE, these are remote procedure call operations. For Encina, these are TRPCs and transactions. There is a pre-defined entitlement, `FUNCTION_ACCESS`, used for function authorization. The rule for this entitlement is

```
INTERFACE=VALID_INTERFACE AND OPERATION=VALID_OPERATION
```

where `INTERFACE` and `OPERATION` are transaction attributes, taken from the remote call or transaction, and `VALID_INTERFACE` and `VALID_OPERATION` are privilege attributes that define what functions a user can call. A privilege for this entitlement might look like

```
VALID_INTERFACE='BANK',VALID_OPERATION='WITHDRAWAL'
```

which states that the user can call the `withdrawal()`operation in the `bank` interface.

Authorization Levels

Authorization using Praesidium AS can be divided into three levels, depending on where the authorization is performed and what entitlement is used.

- *Level I (interface)*: The client evaluates the `FUNCTION_ACCESS` entitlement before initiating a remote call or transaction. (This has been termed an *interface* check for historical reasons, even though the client checks for access to both the interface AND the operation.) This check is automatically performed by the DCE and Encina Extensions.

- *Level II (operation)*: The server evaluates the `FUNCTION_ACCESS` entitlement when it receives an incoming remote call or transaction. This check is automatically performed by the DCE or Encina Extensions.

- *Level III (business rule)*: The application (normally the server, but occasionally the client as well) evaluates an application-defined entitlement using transaction attributes from the remote call or transaction. This check must be coded by the application developer.

The DCE Extension and odss_idl

The DCE Extension is invoked by the client and server stub modules generated by the DCE IDL compiler from the application interface specification. To do this, Praesidium AS supplies a surrogate for the IDL compiler called `odss_idl`. The application developer uses `odss_idl` in place of the `idl` command, and `odss_idl` generates client and server stubs that contain hooks into the DCE Extension. These hooks automate a number of DCE security tasks listed below.

On the client side:

- Set up the security for the client's RPC binding using the server name and protection levels specified by `odss_register_server_info()`.

- Execute the Level I authorization check before sending the call. Raise an exception if the Level I check fails, which normally aborts the client program. This exception can be caught by a `TRY-CATCH` code block in the client.

On the server side:

- Register the server's security parameters specified in its call to `odss_ register_server_info()`.

- Check the protection level of the incoming remote call to ensure that it is at least as high as the level specified by `odss_register_ server_info()`. If not, raise an exception which is relayed back to the client.

- Execute the Level II authorization check before executing the call. If the Level II check fails, raise an exception which is relayed back to the client.

Encina Extension

The Encina Extension performs the same tasks as the DCE Extension, but it works somewhat differently. The Encina Toolkit provides a *callout* facility, where applications can register functions to be called by Encina when certain events happen. The Encina Extension registers callouts for the beginning and completion of transactions and for the sending and receiving of TRPCs.

PRAESIDIUM AUTHORIZATION SERVER ARCHITECTURE

This section outlines the architecture of the Praesidium Authorization Server and its clients. Figure 1 shows the architecture. Praesidium components are shaded; other components are part of DCE or part of an application.

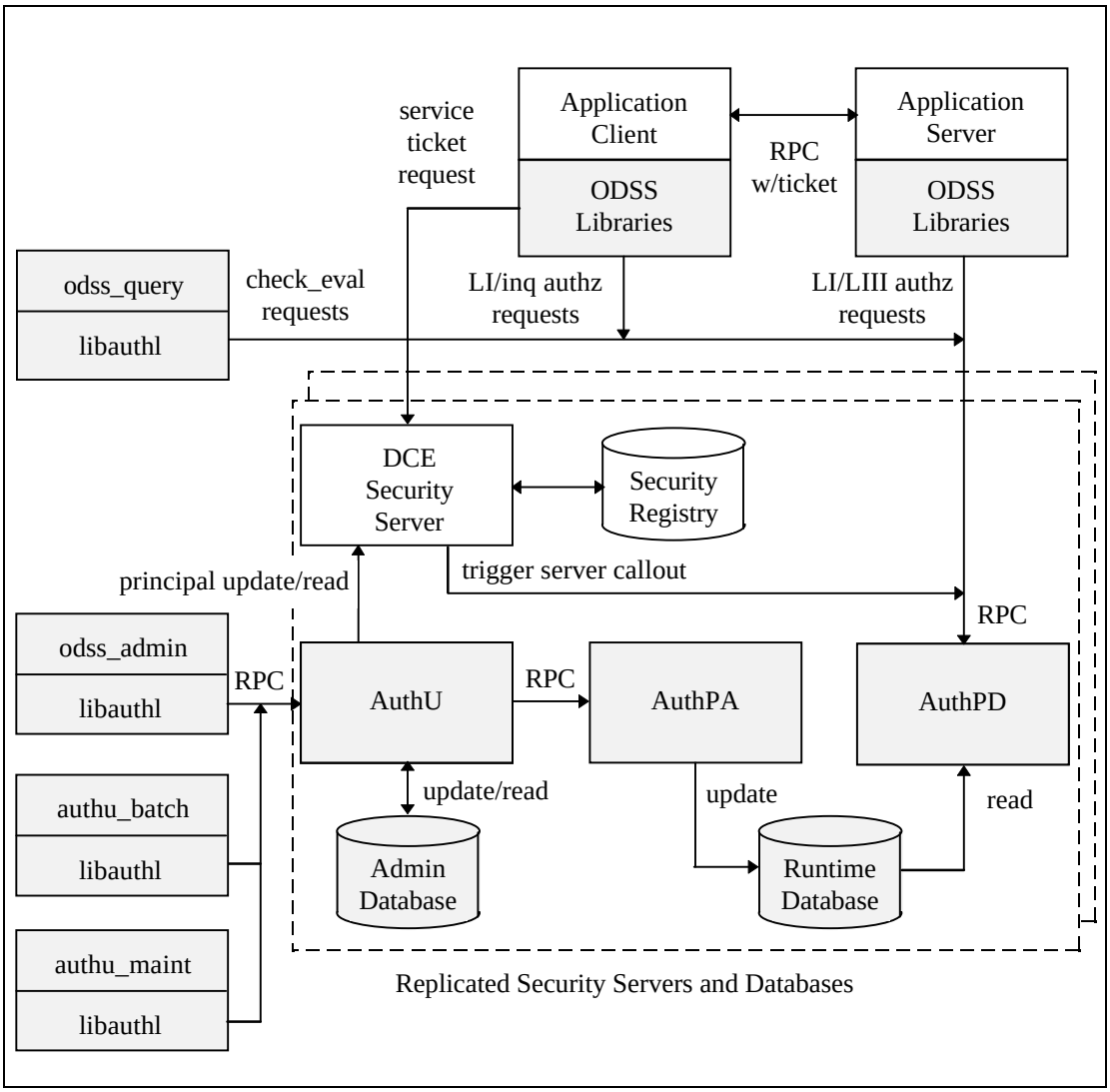

Figure 1. Praesidium Authorization Server Architecture

Server Components

The server components of Praesidium AS run on one or more secure host systems. The server components can be replicated to make sure at least one set of servers is always available and to enhance performance.

- The *Admin Database* contains records for each defined entitlement, profile, and principal. Privileges are attached to profile and principal records. Also, this database contains records for administrative requests that are awaiting checking, processing, or are completed (successfully, failed, or denied). The Admin Database is implemented using the Raima Data Manager.

- The *Runtime Database* contains records for every privilege, indexed by principal, and records for entitlement rules. This database is optimized for retrieval of privileges during runtime entitlement evaluation. The Runtime database is implemented using the Informix On-Line Database.

- The *AuthU Server* manages the entitlements, privileges, profiles, and principals in the Admin Database. It exports a DCE RPC interface with operations for the creation, deletion, modification, viewing, and listing of these objects. AuthU has update access to the Admin Database. It makes remote calls to the AuthPA server to update the Runtime Database when privileges and entitlements change. AuthU can also manage principals in the DCE Security Registry through the DCE `sec_rgy` interface.

- The *AuthPA Server* updates privileges and entitlement rules in the Runtime Database. It exports a DCE RPC interface used by AuthU. AuthPA has update access to the Runtime Database.

- The *AuthPD Server* retrieves privileges and entitlement rules from the Runtime database and includes the general-purpose authorization evaluation engine. It exports DCE RPC interfaces for privilege retrieval and entitlement evaluation, used by the ODSS Libraries, the `odss_query` program, and the trigger server callout (described below). It also has a sockets interface, used by the GSS-API component of the ODSS Libraries.

- The *DCE Security Server* and *Registry* are DCE components that interact with Praesidium AS components. When the ODSS Library `libodssd` (discussed later) requests a service ticket on behalf of an application client, it also requests that the Security Server attach to the ticket the client's `FUNCTION_ACCESS` privileges for the target interface. The Security Server makes a remote call to AuthPD to retrieve these privileges. This process is called a *trigger server callout*.

Administrative Client Components

The administration client components can be run on any HP-UX host in the network. (Some of the components require an X-Windows interface for their displays.)

- The `libauthl` library provides an interface for the creation, modification, and deletion of Praesidium objects in the Admin database, and for other administrative tasks. It binds to AuthU and makes remote calls to AuthU to carry out its tasks.

- The `odss_admin` program provides a GUI interface for management of Praesidium objects and administrative requests. It uses the `libauthl` library to interface to AuthU.

- The `authu_batch` program provides a command-line interface for administration of Praesidium objects. It uses the `libauthl` library to interface to AuthU.

- The `authu_maint` program provides maintenance facilities for the Admin and Runtime Databases, such as deleting old records and synchronization. It uses the `libauthl` library to interface to AuthU, which in turn makes calls to AuthPA.

- The `odss_query` program provides a GUI interface for trial evaluations of entitlements. It uses the `libauthl` library to send evaluation requests to AuthPD.

Application Components: ODSS Libraries

Application clients and servers that use Praesidium AS are linked with two (or more) libraries, collectively called the ODSS Libraries.

- The `libodssb` library provides the ODSS API. It is independent of the underlying environment, and it calls the `libodssd` or `libodssg` libraries to perform requested tasks. All applications must link with `libodssb`.

- The `libodssd` library provides the implementation of the ODSS API functions in the DCE environment. A DCE application must link with `libodssb` and `libodssd`. This library binds to AuthPD and makes remote calls to AuthPD to retrieve privileges and to evaluate entitlements with transaction attributes from remote calls. It encapsulates the DCE authentication and data protection facilities, and it includes the DCE Extension module. Part of the DCE Extension obtains the service ticket for a client, so this library initiates the trigger server callout to retrieve the interface `FUNCTION_ACCESS` privileges. These privileges are included in the ticket attached to the first RPC between the client and the server. Consequently the Level I and II checks can use the privileges in the ticket without having to make additional remote calls to AuthPD.

- The `libodsse` library includes the Encina Extension module. This library depends on the `libodssb` library, so an Encina application must link with `libodssb`, `libodssd`, and `libodsse`.

- The `libodssg` library provides the implementation of the ODSS API functions using sockets and GSS-API. Applications that do not use DCE link with `libodssb` and `libodssg` to use Praesidium authorization. This library authenticates itself to AuthPD using GSS-API and sends authorization requests to and receives responses from AuthPD over a TCP connection.

SUMMARY: BENEFITS OF PRAESIDIUM AS

The Praesidium Authorization Server provides a powerful set of tools for enforcing security in distributed applications. Benefits of Praesidium AS are listed below, in comparision to existing security methods like DCE ACLs.

- *Centralized administration and control*
  All application security data is collected in one set of databases that can be managed in a uniform way. In contrast, other security schemes like DCE ACLs require each application to maintain its own security data, leading to a number of different administration methods.

- *Adminstration focused on users*
  The administration of application security data is organized around users, so it is easy to determine what privileges a user possesses, and to revoke all of a user's privileges if necessary. Other security schemes like DCE ACLs are focused on applications, so it is necessary to find and query each application to which a user might have access.

- *Authorization using application rules*
  The Praesidium entitlement rule language allows developers to encode more complex authorization decisions than are allowed by the permission bits in ACLs. This is particularly true for authorization that requires value comparisons, like our `WITHDRAWAL` entitlement example.

- *Reduced development effort*
  The ODSS API encapsulates most of the work for authentication, authorization, and data protection, relieving the application developer from those tasks.

- *Consistent authorization policies*
  Use of the Praesidium authorization engine allows consistent policies to be implemented across all applications in an organization. Entitlements can be developed by security specialists and used by application developers without extension security training.

- *Support for multiple environments and platforms*
  Praesidium AS is currently available on HP-UX, Windows 3.1 (clients only), and Windows NT. It is easily portable to new platforms and distributed environments. Extensions can be developed that integrate Praesidium AS into new environments, as requested by customers.