Ingres 10.0

Security Guide

ING-10-SG-05

To the extent permitted by applicable law, INGRES PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL INGRES BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USER OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF INGRES IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The manufacturer of this Documentation is Ingres Corporation.

Contents

Chapter 1: Introduction to Ingres Security

This section contains the following topics:

Security Features

Data security is a concern for everyone. The need to ensure the safety of personal information and to protect vital corporate assets stored electronically is of paramount importance.

Ingres has a built-in hierarchical security system that any *privileged user* can use to fully control access to the database.

Security in Ingres is provided through the following features:

- Directory and file permissions
- User-related security features, including:
  - Users
  - Groups
  - Roles
  - Profiles
  - Subject privileges
- Object permissions
- Security alarms
- Security auditing
- Database procedures
- Data at rest encryption

Level of Security

Ingres installations can be administered in compliance with the C2 security standard, as defined by the United States Government National Computer Security Council (NCSC). Level C2 security requires individual logon with password and an audit mechanism.

(B1 security level requires Department of Defense clearance. B1 security is not included in Ingres Open Source.)

Understanding Ingres Security Mechanisms

Ingres supports four security methods (mechanisms). These mechanisms are listed under the Security component in Configuration-By-Forms (or Configuration Manager, if available). The default configuration setting for security mechanisms rarely needs to be changed. Multiple mechanisms are supported concurrently.

Valid mechanisms are:

Null

Allows users to authenticate without providing passwords or other types of authentication. Use of the Null security mechanism is strongly discouraged.

System

Allows authentication through user name and either OS-level passwords or installation passwords. The System security mechanism is provided for backward compatibility with pre-Ingres 9.0 releases.

Ingres

(Default) Allows user authentication against the operating system. The Ingres security mechanism is the preferred standard (static) mechanism. It provides better protection against malicious servers, and employs a more secure encryption mechanism than the System security mechanism.

Kerberos

Allows access through private key and requires a trusted third party. Kerberos is a dynamic mechanism because it uses third-party software and is loaded into the Ingres executable image at runtime. Kerberos is a highly secure alternative to OS security, and optionally allows encryption of the entire data stream between the DBMS Server and the client.

Directory and File Permissions

Databases are protected from user access by the permissions on the directories containing the database files and the permissions on the database files themselves. Users cannot look at the files in a database except through Ingres. Even in Ingres, files are protected from access except from the privileged accounts. The binary files are in a special format, making decoding of any information difficult.

User Authentication

Ingres authenticates users through the operating system account and password together with a corresponding user object definition in the Ingres master database. Additional passwords can also be set on users and roles.

Remote Users

Ingres Net allows access to databases on local or remote nodes. Users can only access data for which they are authorized.

Ingres Net can be configured to allow users access to remote nodes directly (through an installation password) or through local accounts.

Ingres Net encrypts the password entered in the netutil utility (or the Network Utility visual tool) and compares it with the encrypted password in "/etc/password" or in a similar password file of the specific machine.

Installation Passwords

Ingres Net allows you to set up an Installation Password to authorize access to a server installation from a remote client installation without setting up an operating system account on the server; the user retains his identity as defined on the client instance.

The main advantage of using installation passwords is that users on the client do not require a login account on the server.

A valid Ingres user object must be created in the Ingres master database using the same operating system user ID as on the remote client. OS authentication is done on the remote client, where the user must have a login and password.

For details of Ingres Net setup, see the *Connectivity Guide*.

The ingvalidpw Utility (UNIX)

In some environments, Ingres uses the ingvalidpw program (see page 15) to validate user passwords. Ingvalidpw is used depending on the requirements of the platform where the password is validated. For example, the Ingres DBMS Server uses the ingvalidpw program to validate shadow passwords on UNIX or to enforce C2 security in some UNIX environments.

Authorization Identifiers

Access can be granted to four authorization identifiers.

Identifiers are listed here from highest to lowest precedence, which determines the privilege enforced for a session if a particular privilege is defined for more than one authorization identifier associated with a session.

- Role

  Roles simplify access to the database by associating subject privileges and permissions with an application. Roles can be created with the option of an additional password. The EXTERNAL_PASSWORD option allows a role's password to be passed to an external authentication server for authentication.

- User

  For each valid Ingres user, a user object must be created in the Ingres master database iidbdb. The user object specifies the user name, default group, default profile, subject privileges, and other attributes.

- Group

  Groups simplify the managing of permissions because individual users can be added or removed from groups as required. Being a member of a group does not automatically give the user the permissions granted to the group. The user must have the group specified as default group or specify the group name in the session startup.

- Public

  Granting permissions on objects to PUBLIC allows any user, group, or role access to those objects. The use of grants to PUBLIC should be limited.

Subject Privileges

Subject privileges define the operations a user can perform, and are assigned to a user or a role. Subject privileges include: Auditor, Create Database, Maintain Audit, Maintain Locations, Maintain Users, Operator, Security, and Trace.

Object Permissions

Data access can be restricted through the granting of permissions on objects. Permissions can be granted on the following objects: database, table, view, procedure, database event, role, and current installation.

Security Alarms

Security alarms can be set up to monitor events against a database or individual tables. Such triggers on important databases and tables are useful in detecting unauthorized access.

Security alarms can monitor success or failure of connecting or disconnecting from a database, and selecting, deleting, inserting, or updating data in a table.

Security alarms can raise a database event (dbevent), which can be monitored by background programs that respond accordingly. Security alarms can be assigned to specific authorization identifiers to limit the monitoring to selected users, groups, or roles.

Security Auditing

Security events can be recorded for the entire Ingres installation. All objects or certain classes of objects can be targeted. Information in the audit logs can grow quickly, so you should carefully plan what events to audit.

Database Procedures

Database procedures provide an extra level of control over data access and modification. Database procedures can be used with security alarms to enhance security auditing.

Data at Rest Encryption

Specific database table columns can be encrypted to enhance data security, ensure privacy, and protect media that contains database records holding sensitive information. Data in the protected columns is stored on disk or other media in encrypted form and can only be accessed if the encryption passphrase is known.

Data at rest encryption protects table information, transaction information, and full database backups.

Chapter 2: Understanding Directory and File Permissions

This section contains the following topics:

Directory and File Permissions

Databases are protected from user access by the permissions on the directories containing the database files and the permissions on the database files themselves. Users cannot look at the files in a database except through Ingres. Even in Ingres, files are protected from access except from the privileged accounts. The binary files are in a special format, making decoding of any information difficult.

File Permissions on Windows

The Ingres database supports the special security features of the NTFS file systems. The appropriate file security and permissions are set using the setperm utility. During the installation process, only the installation owner is given access to utilities with powerful abilities such as the Interactive Performance Monitor (IPM). Permissions can be modified to allow access to less privileged users, but doing so can compromise the security and integrity of the database.

File Permissions on UNIX

Ingres is installed with the required file permissions to operate efficiently. The installation process creates the subdirectories with the appropriate file permissions and ownership.

Many of the files under the $II_SYSTEM directory are critical to the proper functioning of Ingres. Do not delete or alter any files that the installation process places in the $II_SYSTEM directory and its subdirectories. No other files or directories should be created in this directory or its subdirectories.

Important! Do not alter the permissions on the $II_SYSTEM directory, subdirectories, or any files in these directories; otherwise, the security and integrity of the database can be compromised.

Although it is possible to set more restrictive file permissions, you should do so only with extreme caution. The more restrictive file permissions will have to be tested extensively in a test environment that matches the production setup. Furthermore, such a change deviates from the Ingres standard and modified permissions could be overwritten when installing patches or when upgrading.

Chapter 3: Security Features on UNIX

This section contains the following topics:

Ingvalidpw Program (Password Validation)

In some environments, Ingres uses the ingvalidpw program to validate user passwords. The passwords may originate from any local application or from a remote application coming through Ingres Net or the Data Access Server.

Ingvalidpw is used depending on the requirements of the platform where the password is validated. For example, the Ingres DBMS Server uses the ingvalidpw program to validate shadow passwords on UNIX or to enforce C2 security in some UNIX environments.

(To see if your UNIX environment requires the ingvalidpw executable to enforce C2 security, refer to the Ingres platform-specific readme for your Ingres release.)

Create Password Validation Program (UNIX)

On UNIX, the Ingres DBMS Server uses the ingvalidpw program to validate shadow passwords. This executable is created at installation time or loaded from the distribution media.

The mkvalidpw script tries to recompile the ingvalidpw program if your machine has a C compiler available; otherwise it copies the supplied ingvalidpw program to the $II_SYSTEM/ingres/bin directory. The mkvalidpw script also sets the II_SHADOW_PWD variable in the Ingres symbol.tbl to enable shadow password validation.

To create the ingvalidpw program

1. Log in as root.
2. Set the II_SYSTEM and PATH variables to the same values as those for the user account that owns the installation.
3. Run the mkvalidpw script, located in the directory $II_SYSTEM/ingres/bin, as follows:

   ```
   mkvalidpw
   ```

   The ingvalidpw executable is created.
4. Shut down and restart the Name Server.

   The ingvalidpw program is ready for operation.

Ingvalidpam Program (Password Validation Through PAM)

In Linux and UNIX environments, the ingvalidpam program can be used instead of ingvalidpw to validate passwords through pluggable authentication modules (PAM). For more information, see the chapter "Implementing PAM in Ingres."

Access Control with Setuid (UNIX)

The UNIX Setuid bit can be used to control access to data, without needing to issue grants to specific users or the public. This technique allows a user to operate on the data, but only when the user has accessed it through a specific application program. The data is otherwise inaccessible to the user. After the user has exited the program, the user cannot access this same data using any Ingres utility.

Use Chmod to Set the Setuid Bit

After an embedded SQL application program has been created, the permissions of the program file can be set so that it can run with the effective user ID set to that of the owner of the file. If, for example, the owner of the file is the DBA, any user executing the program is recognized as the DBA—and has the same access to objects and data as the DBA—for the life of the program.

The UNIX chmod command issued at the operating system prompt is used to change the mode of a file. The following format of this command changes the mode of the specified file name to give “set user id on execution” and “execute” permission to everyone. The 4 sets the Setuid bit:

```
chmod 4711 filename
```

For example, if the following command is executed:

```
chmod 4711 app1prog
```

The resulting file permissions looks like this:

```
-rws--x--x     1 dba              7584 Mar 30 app1prog
```

Using this technique, the DBA (or other user, such as an application developer) can allow any user to temporarily become the *effective user id* for controlled access to specific application programs. The effective user ID is recognized when a connection is made to the Ingres DBMS Server.

Note: Only the application owner or the root user can run the chmod operating system command.

Example: Refer to Setuid in an Embedded SQL Application

An example is shown here that can be used in embedded SQL application programs that are made accessible by the Setuid bit. It checks that the connections are in place before allowing the program to be run.

The program requires that the login of the program owner be known (in this case, fred, defined in line 2. It uses the UNIX getpwnam function to fetch the effective user ID from the passwd structure. For details, see the UNIX documentation.

The program checks to see if the caller can access the database, printing an error message if not. This can occur, for example, if a chmod command has not been issued on the application program, as described in the previous section.

The application code appears between the CONNECT and DISCONNECT statements. In the sample code that follows, the action of the program is to print the effective user ID:

```
#include  <pwd.h>
#define   DBA       "fred"
/* Example to demonstrate how to run an embedded SQL program
** in such a way that there is no requirement for the users to have
** privileges on the tables accessed by the program.
**
** This example assumes the program will access tables owned by the
** DBA, but this scheme could be used for any user. Additionally, assume
** the DBA has granted no privileges.
**
** This program checks that it is running setuid to the DBA.
** If it is, it does a database connect.
** If it is not running setuid to the DBA,
** it gives an error message to the user and exits.
** This check is done simply as a convenience to the user.
** If this check isn't done and the user is a valid INGRES user,
** he or she can still connect to the database,
** but will not be able to access any of the data.
** This would be frustrating to the user,
** so the program stops them before they get connected.
**
** For this scheme to work, the executable MUST be setuid to the DBA.
*/

main()
{
  EXEC SQL begin declare section;
    char  username[32];
    char  dbname[32];
  EXEC SQL end declare section;

  int   user;
  int   ret_val = 0;

  struct  passwd   *getpwnam();
  struct  passwd   *password_entry;

  password_entry = getpwnam(DBA);

  /* Check to see if the user interface is running setuid
  ** to the dba (fred).
  ** If not, give the user a message and abort.
  ** If so, connect to the database.
  */
  if ((user = geteuid()) != (password_entry->pw_uid))
  {
    printf("Error starting application. Contact the DBA.\n");
    ret_val = 1;
    exit(ret_val);
  }

  EXEC SQL connect dbname;

  /* The following query will demonstrate that the dbms connection
  ** was done by user fred, not the actual user.
  */

EXEC SQL select username() into :username;
  printf("User is %s\n", username);

  EXEC SQL disconnect;

}
```

Chapter 4: Authorizing User Access

This section contains the following topics:

Common Types of Ingres Users

In most installations, there are four types of users:

Installation Owner

The installation owner is typically an account named "ingres", but the ingres name is not required.

By default, this user has the Security privilege and most of the other privileges. Some of the privileges, however, can be revoked from this user and the system will still operate correctly. In a good production system, this user performs only administrative tasks on the system (such as startup and shutdown).

System Administrator

The system administrator is sometimes the "root" account. This account is commonly owned by the Information Technology (IT) department, but is also commonly owned by a user who has been defined as the Ingres System Administrator.

In a large production environment, there may be one or a few of these users. These users have the Security privilege, which allows them to use the -u flag on commands to imitate other users, and usually possess other privileges such as Maintain_locations and Maintain_users; if security auditing is enabled, they will also typically have Auditor and Maintain_audit privileges. The responsibility of this user is to perform administrative tasks that affect the entire Ingres instance such as creating and destroying Ingres users, allowing Ingres to use new disk drives, and monitoring the Ingres security audit logs.

In smaller environments, the system administrator and the installation owner may be the same user.

Database Administrator (DBA)

The DBA typically has only the Createdb privilege. DBAs can use the -u flag in their own databases only.

Typically, the DBA is not the installation owner, and in a good production system, does not have the Security privilege. The definition of the primary DBA for any given database is the user who ran the createdb command to create that database. Additional DBAs can be defined for a database by granting (see page 48) them the Db_admin privilege for that database.

End User

The end user typically has no privileges and cannot create a database.

Ingres Users and the DBA

Ingres is designed for a wide variety of users, from database management experts who create and maintain databases, to end users who only examine or update data. Moreover, users can have multiple roles. For example, a user can be the database administrator of one Ingres database and the end user of another.

One company, for example, can have a single database administrator who controls all access to databases, whereas another company has a primary database administrator at its corporate headquarters and a local database administrator at each of its satellite sites. In the latter case, the primary database administrator controls access to corporate databases, such as sales, inventory, payroll, and human resources; and the local database administrators are responsible for authorizing access to production or research databases.

Regardless of the type of enterprise, if you are a database administrator who has been granted the maintain_users privilege, you are able to add new users to an Ingres database.

How to Establish User Access

The process of establishing access to Ingres is as follows:

1. The system administrator defines user accounts in the operating system.

   Accounts are needed for local users and for remote users who access the product through a local account.

   Note: This step is optional if an installation password is defined, in which case users access Ingres directly, without having to go through a local account.

   All accounts can be set up before or after Ingres is installed (except for the installation owner account, which is set up during installation, belongs to the system administrator, and is assigned maximum Ingres privileges to perform all operations).

2. The database administrator defines user objects.

   After the accounts are set up, a database administrator or system administrator starts Ingres and defines user objects. Part of the user object definition is a user ID, which corresponds to the user ID used to log on to the operating system.

Typically, the system administrator sets up a user object for the database administrator, who in turn sets up user objects for other users.

Users and Profiles

Users are defined using user objects and, optionally, profile objects.

A *user object* is a definition that specifies the user's name, default group, default profile, subject privileges, and several other attributes.

A *profile* is a template that defines a set of subject privileges and other attributes that can be applied to one or more users. The user authorization process can be streamlined by using profiles.

Working with User Objects

You can perform the following basic operations on user objects:

- Create and alter user objects
- View existing user objects, including the detailed properties of each object
- Drop user objects

These tasks can be accomplished using the **accessdb** forms-based utility.

In SQL, you can use the CREATE USER, ALTER USER, and DROP USER statements when working in a session connected to the iidbdb database. For details, see the *SQL Reference Guide*.

In VDBA, use the Users branch in the Database Object Manager window. For detailed steps, see VDBA online help.

Note: Many of the features associated with a user object, such as subject privileges, password, expiration date, and security auditing, are security-related features, described later in this guide.

Create a New User with Accessdb

You must have maintain_users permission to authorize users. Using the accessdb utility, you can add, modify, or delete users and grant them database access permissions.

To authorize a new user

1. Start accessdb by issuing the following command at the operating system prompt:

   ```
   accessdb
   ```

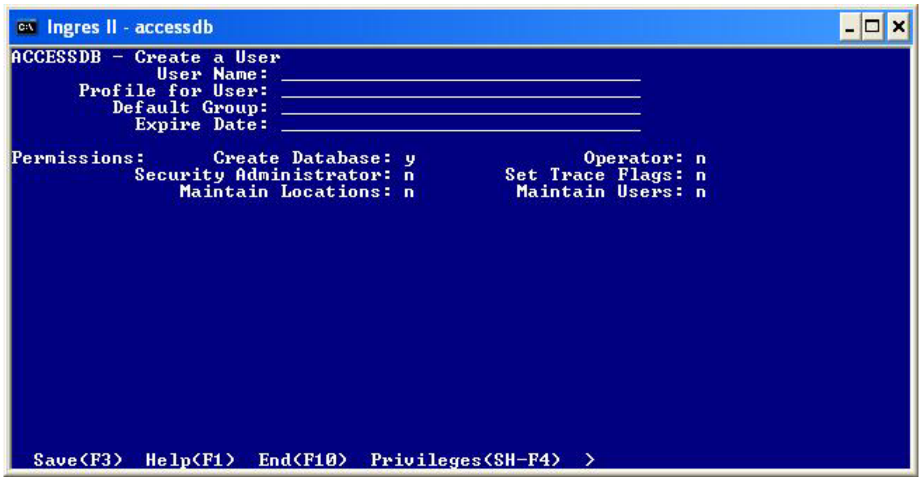
   The accessdb main menu appears.

2. Select Users from the accessdb main menu.

   The Users Catalog screen appears.

3. Select Create.

   The Create a User screen appears.



```
ACCESSDB - Create a User
              User Name: ________________________________
       Profile for User: ________________________________
          Default Group: ________________________________
            Expire Date: ________________________________

Permissions:        Create Database: y          Operator: n
           Security Administrator: n     Set Trace Flags: n
               Maintain Locations: n      Maintain Users: n




  Save(F3)  Help(F1)  End(F10)  Privileges(SH-F4)  >
```

4. Type the user information into the following fields:

   User Name

   Login name of the user. The name can be a regular or delimited identifier.

   For example, to use a numeric user ID, the name must be delimited (enclosed in double quotes, as in "888282").

   (For details on delimited identifiers, see the *SQL Reference Guide*.)

   Profile for User

   (Optional) Default profile for the user.

   Default Group

   (Optional) Default group (see page 31) the user is assigned to.

   Expire Date

   (Optional) User expiration date (see page 27).

   Note: After you save the user definition, you can assign a User Password (see page 27). User passwords are optional.

5. In the Permissions section, change the default subject privilege settings for the user by tabbing to the desired field and typing the appropriate value:

   y

   Grants the privilege

   n

   Denies the privilege

   r

   Makes the privilege requestable. For details on requestable privileges, see SET SESSION in the *SQL Reference Guide*.

6. Select Save from the menu.

   The user entry is saved.

7. Repeat steps 3-6 for each new user you want to authorize.

8. Select End twice.

   You are returned to the accessdb main menu.

9. Select Quit.

   Note: If you do not see the Quit function listed, press ESC to scroll through the menu options.

   The accessdb utility ends.

User Expiration Date

The user expiration date is an optional part of the user definition. It determines the date after which the user can no longer access Ingres.

An expiration date can be specified as any valid Ingres date or as a date or time interval. For example, you might specify an interval of '1 month' or '1 year,' or an absolute date, such as '5-jan-2007.'

The user expiration date is checked each time the user connects to the Ingres DBMS Server. If the expiration date has passed, then access is denied.

To enable an expired user to connect, the associated user (or profile) object must be modified to reset the expiration date.

User Password

A password can be specified as part of the user definition.

Note: If the Ingres DBMS Server is located on a remote node, this user password is in addition to the login password or installation password that the user must specify as part of the vnode definition.

When a session requires a password and one is not specified, a prompt requests a password anytime Ingres makes a connection between an Ingres tool and the DBMS. For security reasons, a password prompt is issued if either a required password is missing or the user name is unknown or illegal. This behavior is consistent with that of operating systems during logon.

If the connection specifies a password directly, as is the case with an application, no prompt is issued. This must be done if the application cannot prompt for a password. If the application can prompt for a password, it does. Then it passes the value entered using the DBMS_PASSWORD clause of the CONNECT statement.

User passwords are validated directly by the Ingres DBMS Server or by an external authentication mechanism, depending on how the user object is configured.

Note: If a user with the Security privilege starts a session using the –u flag to impersonate another user, the real user's password—not the impersonator's—is required.

Any user is permitted to change their own password; to do so, however, they must supply their old password. Any user with the maintain_users privilege can change the password of another user, in addition to changing the method of password validation or removing the password altogether.

Note: Passwords also apply to roles.

Authorize Multiple Users with SQLscript

sing accessdb you can create a file of the users at your installation and their corresponding permissions. This file is useful for copying installations.

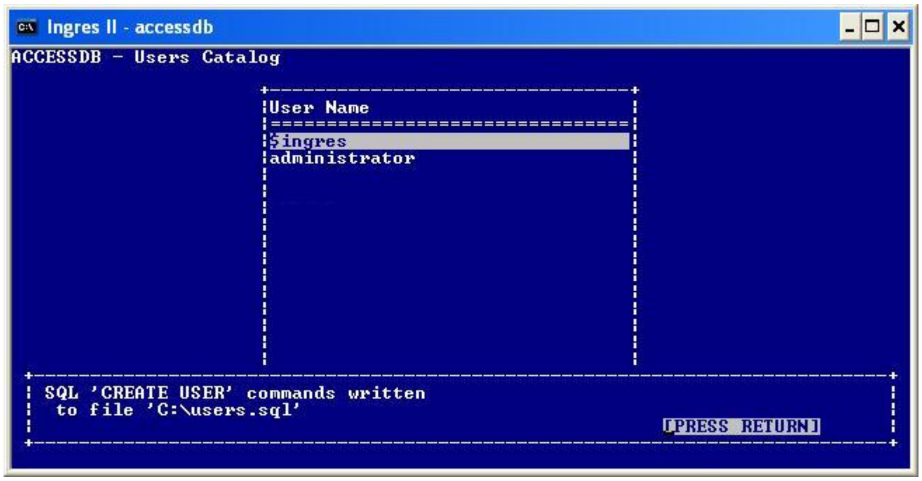
To create a file of users

1. From the accessdb main menu choose Users.

   The Users Catalog screen appears.

2. Choose the SQLscript menu item.

   The accessdb utility creates an SQL script and displays an SQLscript message indicating the file location.



```
ACCESSDB - Users Catalog

                          User Name
                          ================================
                          $ingres
                          administrator

 SQL 'CREATE USER' commands written
  to file 'C:\users.sql'
                                                   [PRESS RETURN]
```

3. Press Return.

   The message is cleared from the screen.

4. Select End.

   Note: If you do not see the End function listed, press ESC to scroll through the menu options.

   You are returned to the accessdb main menu.

Note: The SQLscript function creates users only, not the profiles, groups, or roles associated with each user. Roles and groups must be unloaded and reloaded for the script to generate the expected results.

Working with Profile Objects

You can perform the following basic operations on profile objects:

- Create and alter user objects
- View existing user objects, including the detailed properties of each object
- Drop user objects

In SQL, you can use the CREATE PROFILE, ALTER PROFILE, and DROP PROFILE statements when working in a session connected to the iidbdb database. For details, see the *SQL Reference Guide*.

In VDBA, use the Profiles branch in the Database Object Manager window. For detailed steps, see VDBA online help.

Example of Using a Profile

After a profile is created, it can be associated with a new or existing user object as the default profile for that user. By doing so, the attributes defined in the profile are associated with the user, and the user's attributes are updated whenever the profile is modified.

Attributes can also be set directly at the user level to override settings at the profile level.

For example, a company conducts an analysis of the tasks and responsibilities of its database operators at multiple sites. They find three tasks that are common to this type of user: database and file location maintenance, debugging, and database backups.

They create a profile for maintaining databases called dbop (database operator) with the appropriate subject privileges:

- maintain_locations
- trace
- operator

Whenever the company hires a new database operator, the database administrator can associate the dbop profile with that new user. Doing so automatically assigns the maintain_locations, trace, and operator privileges to the user.

If the company alters the dbop profile to include the maintain_users privilege, the change automatically affects any user currently using the profile.

Because the dbop profile did not specify the option to audit the query text associated with user queries, users associated with this profile are not audited for query text. To audit the query text for only one of the users associated with the dbop profile, this option can be turned on at the user level (by using the ALTER USER statement or Alter User dialog in VDBA). This overrides the default for that particular user, without affecting any other users of the dbop profile.

Default Profile

A *default profile* is the profile initially assigned to a user if one is not explicitly assigned.

The default profile specifies the following:

- No default group
- No subject privileges or default privileges
- No expiration date
- No security audit options (that is, default events are audited)

Note: You can alter the default profile but you cannot drop it.

Note: Altering the default profile will alter privilege attributes of all users that have not been given a specific profile.

In SQL, you can change the default profile using the ALTER DEFAULT PROFILE statement. For more information, see the *SQL Reference Guide*.

In VDBA, the default profile is indicated as (default profile) in the Profiles branch in the Database Object Manager window. For more information, see VDBA online help topic Altering a Profile.

Groups and Roles

Groups and roles can simplify control of database access. Groups are used to apply permissions to a list of users, while roles are used to associate subject privileges and permissions with an application.

Groups

A *group* is an identifier that can be used to apply permissions to a list of users associated with the identifier.

A group allows multiple users to be referenced by a single name.

For example, a company has an accounting group to identify the accounting department employees as a whole, and a payroll group to identify the payroll department employees as a whole. To define these groups, the DBA creates the groups and adds all the users in the associated departments to their respective groups. The groups can be easily maintained by adding and dropping users as they join or leave the departments.

Note: A user can be a member of more than one group.

Working with Group Objects

You can perform the following basic operations on group objects:

- Create and alter group objects
- View existing group objects, including the detailed properties of each individual object
- Drop group objects

In SQL, you can accomplish these tasks with the CREATE GROUP, ALTER GROUP, and DROP GROUP statements when working in a session connected to the iidbdb database. For details, see the *SQL Reference Guide*.

In VDBA, use the Groups branch in the Database Object Manager window. For details, see the VDBA online help.

Example: Creating, Altering, and Dropping a Group using SQL Statements

To create a new group, specify a user-defined group ID in a CREATE GROUP statement, and then list the users in the group on the WITH USERS clause. The group can be amended later by ADD USERS or DROP USERS clauses on the ALTER GROUP statement. The group can be deleted with the DROP GROUP statement.

Here are examples of using the CREATE GROUP, ALTER GROUP, and DROP GROUP statements:

1. Create a group identifier for a company's telephone sales force and put the salespersons' user IDs in the group user list:

   ```
   CREATE GROUP tel_sales
      WITH USERS = (harryk, joanb, jerryw, arlenep);
   ```

2. Create the group identifier and_muchmuchmore and reserve it for later use. This is done by omitting the WITH USERS clause:

   ```
   CREATE GROUP and_muchmuchmore;
   ```

3. Add two users to the group tel_sales and drop three users.

   The adding and dropping must be done in separate ALTER statements:

   ```
   ALTER GROUP tel_sales
      ADD USERS (dannyh, helent);
   ```

   ```
   ALTER GROUP tel_sales
      DROP USERS (harryk, joanb, arlenep);
   ```

4. Drop all users from the group researchers. Then drop the group. The DROP ALL option should be run prior to dropping a group, since a group cannot be dropped if its user list has any members.

   ```
   ALTER GROUP researchers DROP ALL;
   DROP GROUP researchers;
   ```

   If a user is dropped from a group but is currently active in a session, that session continues to have the group's permissions until it terminates. "Currently active" means that the user is currently connected to a database. If a group is dropped, all the permissions assigned to the group are dropped.

Groups and Permissions

After a group is created, you can associate permissions with it. When you grant permission to a group, you are, in effect, granting that same permission to each user in the group.

Groups are a convenient way to give the same permissions to many users at once.

Groups also make managing the permissions easy by allowing you to add users to (and remove users from) the group. For example, grant the payroll group insert, delete, and select permissions on the payroll tables, which gives all the users in the group those permissions. If an employee leaves the payroll department, or if a new employee joins, you simply have to drop or add a user from the group, without modifying the permissions. Similarly, if you find that the group needs fewer or more permissions, revoke or grant the permissions once, for the entire group, rather than individually for each member of the group.

Being a member of a group, however, does not automatically give a user the permissions granted to the group. Users must specifically identify themselves as part of a group to be allowed the associated permissions.

A user can be identified as part of a group in two ways:

- Specifying a group ID at session startup
- Specifying a default group for the user. A default group is specified for a user using the SQL statements CREATE USER or ALTER USER.

Specifying Group ID at Session Startup

When starting a session, a user can specify a group identifier, as follows:

- On the –G flag for many system commands. For details, see the *Command Reference Guide*.
- With the CONNECT statement as part of an application.
- As part of the connection profile for an OpenROAD session. For more information, see online help for the Create Connection Profile dialog in OpenROAD.

Roles

A *role* is an identifier that can be used to associate permissions with applications.

A role is typically associated with one or more applications to grant permissions to those applications.

For example, a company uses a restricted application that performs certain checks before updating the payroll tables to ensure that these tables are updated correctly. The DBA defines a role, for example update_payroll, and later assigns appropriate permissions for the necessary tables. The application developer associates the role with the application.

Note: When defining a role, the DBA normally works with the application developer, so that they can agree on what role identifier and password to use for specific applications.

For further security, a role password can be specified. The role password is optional.

Working with Role Objects

You can perform the following basic operations on roles:

- Create and alter role objects

  Note: Passwords can be associated with role objects as well as with user objects. The section User Password (see page 27) contains more information on passwords.

- View existing role objects, including the detailed properties of each object
- Drop role objects

In SQL, you implement roles with the CREATE ROLE, ALTER ROLE, and DROP ROLE statements when working in a session connected to the iidbdb database. For details, see the *SQL Reference Guide*.

In VDBA, roles are implemented using the Roles branch in the Database Object Manager window. For details, see VDBA online help.

Example: Creating, Altering, and Dropping a Role using SQL Statements

Here are examples of using the CREATE ROLE, ALTER ROLE, and DROP ROLE statements:

1. Create a role identifier and password for an inventory application for a bookstore:

   ```
   CREATE ROLE bks_onhand
   WITH PASSWORD = ’hgwells’;
   ```

   Note: Enclose any special characters or blanks in the roll password in single quotes. Blanks are not significant in passwords—for example, ’a b c’ is equivalent to ’abc’. The password can be up to 24 characters in length. (You should choose long passwords that cannot be easily guessed.) If no password is assigned, then all users that have been granted access to that role have access to the role identifier and its associated permissions. (See the GRANT ROLE statement in the *SQL Reference Guide*.) If a password is assigned, then the user or application must additionally use the password to access the role.

2. Create a role identifier with no password for the daily sales application for a bookstore:

   ```
   CREATE ROLE dly_sales WITH NOPASSWORD;
   ```

3. Create dual role IDs (these function as synonyms) and password for a recommended list application for a bookstore:

   ```
   CREATE ROLE sclemens, mtwain
   WITH PASSWORD = ’goodluck’;
   ```

4. Change the password for the existing role identifier new_accounts to eggbasket:

   ```
   ALTER ROLE new_accounts
   WITH PASSWORD = eggbasket;
   ```

5. Drop the existing role identifier sales_report:

   ```
   DROP ROLE sales_report;
   ```

Roles and Permissions

After a role is created, you can then associate permissions with it and create grants to it for individual users. For example, for the associated application to execute properly, grant update permission to all payroll tables for the update_payroll role. For details, see Object Permissions.

When you grant an object permission or a subject privilege to a role, you are, in effect, granting that same permission or privilege to any session that is started using that role.

Specifying Role ID at Session Startup

When starting a session, you must specify a role identifier, which puts into effect the associated permissions and subject privileges. For example:

- On the –R flag for many system commands. For details, see the *Command Reference Guide*.
- With the CONNECT statement as part of an application.
- For an application image as part of the connection profile for an OpenROAD session. For more information, see online help for the Create Connection Profile dialog in OpenROAD.

For the DBA or a user (such as the system administrator) who has the security privilege, neither role identifier nor password is validated. For any other user, the specified role must exist, the user must be granted permission to use the role, and any required password must be specified correctly; otherwise, the connection is refused.

Chapter 5: Assigning Privileges and Granting Permissions

This section contains the following topics:

Subject Privileges

A *subject privilege* defines the type of operations permissible in a user session. Subject privileges are assigned to a user (subject).

Subject privileges are typically assigned when a user object is created or modified. Subject privileges can also be assigned to roles, as discussed in Groups and Roles (see page 31).

To set or change subject privileges for a user, you must have the maintain_users privilege.

Important! Subject privileges allow many trusted operations to be performed. Therefore, assign privileges with care, especially the Security privilege.

The subject privileges are as follows:

auditor

Enables the user to query the security audit log

createdb

Enables the user to create and destroy databases

maintain_audit

Enables the user to control what information is written to the security audit log

maintain_locations

Enables the user to manage database and file locations

maintain_users

Enables the user to perform various user-related functions, such as creating users and roles

operator

Enables the user to perform database backups and other maintenance operations

security

Enables the user to perform security-related operations, including impersonating other users, and to avoid certain security checks, such as database privilege checks

trace

Enables the user access to tracing and debugging features

Auditor Privilege

The Auditor privilege allows a user to obtain information from the audit log.

A user with this privilege can:

- Register the audit log file to a virtual table using the REGISTER TABLE statement (or perform the equivalent operation in VDBA).
- Remove the registration for an audit log file using the REMOVE TABLE statement (or perform the equivalent operation in VDBA).
- Query the audit log once it has been registered as a virtual table.
- Obtain the audit log file name by calling dbmsinfo(‘security_audit_log’).

Related Information

Maintain_Audit Privilege (see page 41)
Security Auditing (see page 66)

Createdb Privilege

The createdb privilege gives the user the ability to create databases.

This privilege is required to use the createdb system command or to use the equivalent operation in Visual DBA, as described in the "Creating Databases" chapter of the *Database Administrator Guide*.

Maintain_Audit Privilege

The maintain_audit privilege allows a user to manage auditing features, including determining the security audit activity level for profiles, users, and roles, and the ability to turn security auditing on and off.

The maintain_audit privilege is typically assigned to the system administrator, the database administrator, or a separate security administrator.

A user with this privilege can:

- Issue the ENABLE and DISABLE SECURITY_AUDIT statements (or perform the equivalent operations in VDBA).
- Change the current audit state using the ALTER SECURITY_AUDIT statement (or perform the equivalent operation in VDBA).
- Specify the SECURITY_AUDIT clause for ALTER/CREATE PROFILE, ALTER/CREATE USER, and ALTER/CREATE ROLE statements (or similarly determine the security audit activity level when working with profile, user, and role objects in VDBA).

Related Information

Auditor Privilege (see page 40)
Security Auditing (see page 66)
Maintain_Users Privilege (see page 42)

Maintain_Locations Privilege

The maintain_locations privilege allows a user to control the allocation of disk space, create new locations or allow new locations to be created, and allow existing locations to be modified or removed.

This privilege is needed to issue the CREATE, ALTER, and DROP LOCATION statements (or to perform the equivalent operations on location objects in VDBA).

Maintain_Users Privilege

The maintain_users privilege allows a user to perform various user-related functions.

A user with this privilege can:

- Issue CREATE/ALTER/DROP PROFILE statements to maintain profiles (or perform the equivalent operations on profile objects in VDBA).
- Issue CREATE/ALTER/DROP USER statements to maintain users (or perform the equivalent operations on user objects in VDBA).
- Issue CREATE/ALTER/DROP GROUP statements to maintain groups (or perform the equivalent operations on group objects in VDBA).
- Issue CREATE/ALTER/DROP ROLE statements to maintain roles (or perform the equivalent operations on role objects in VDBA).

Related Information

Maintain_Audit Privilege (see page 41)

Operator Privilege

The Operator privilege allows a user to run the following system commands:

- ckpdb
- rollforwarddb
- auditdb
- sysmod
- verifydb
- relocatedb
- fastload
- alterdb
- infodb

A user who is responsible for running Ingres requires the Operator privilege.

These system commands can alternatively be run through the Remote Command (rmcmd) Server by a (client) user who has the rmcmd privileges rather than the Operator privilege (assuming that the user who launched rmcmd on the server side has the Operator privileges). The sysmod command, however, requires the client user to have the security privilege or be the user who launched rmcmd on the server side. For details, see Grant Access to Remote Users and How Remote Commands Are Executed in the *System Administration Guide*.

Security Privilege

The Security privilege allows a user to monitor the security of the system and the activities of its users. The Security privilege and all other privileges are automatically bestowed on the installation owner.

A user with this privilege can:

- Use the -u flag on commands to impersonate other users (or perform the equivalent using the Users branch of the Virtual Nodes toolbar in VDBA).
- Connect to any database with unlimited database privileges. (In effect, database privileges are not enforced for users with the Security privilege.)
- Issue CREATE/DROP SECURITY_ALARM statements to configure database and installation security alarms (or perform the equivalent operations in VDBA).

Important! The Security privilege is powerful because it allows the holder to impersonate any other user. At least one user with the Security privilege is required, but the privilege can be restricted as tightly as possible so that your system security is not compromised.

Note: The security privilege does not allow a user to bypass granted permissions on a database object; unless permission is granted to the user they are impersonating, they will not be able to access the object.

Trace Privilege

The Trace privilege allows a user to perform tracing, troubleshooting, and debugging operations. It enables the user to set the debugging trace flags using the following statements:

- SET[NO]PRINTQRY
- SET[NO]RULES
- SET[NO]PRINTRULES
- SET[NO]IO_TRACE
- SET[NO]LOCK_TRACE
- SET[NO]LOG_TRACE
- SET TRACE POINT

The Trace privilege permits access to possibly confidential information, so it should be enabled for the installation owner or security administrator only.

For details on tracing, see the *System Administrator Guide*.

Sets of Privileges Associated with a Session

In addition to assigning subject privileges to a user, Ingres lets you define a default set of subject privileges that will be available at session startup.

In addition, any privilege assigned to the user can be added or dropped during the life of the session; this capability effectively applies the principle of least privilege.

The principle of *least privilege* asserts that a subject must have the minimum privileges required to perform an operation, and that these privileges must be active for the minimum amount of time necessary to perform that operation.

Thus, a session has three sets of privileges associated with it:

- The *default privilege set* contains those privileges that become active when an Ingres connection is initiated.
- The *working privilege set* contains those privileges that are active at any particular time (at session startup, the working privilege set is equivalent to the default privilege set).
- The *maximum privilege set* contains all privileges that a particular user is allowed to have.

The working privilege set is determined during the life of the session, when privileges can be made active as necessary to allow a privileged operation to be performed and made inactive on completion of the task.

The working privilege set is specified using the SET SESSION statement (described in the *SQL Reference Guide*). Using SET SESSION, you can:

- Add allowed privileges to the working privilege set
- Drop privileges from the working privilege set
- Replace the working privilege set with specified allowed privileges
- Set the working privilege set to all allowed privileges
- Reset the working privilege set to the default privilege set
- Remove all privileges from the working privilege set

In VDBA, the maximum privilege set consists of all the privileges enabled in the Users column of the Create User or Alter User dialog. The default privilege set, which is a subset of the maximum privilege set, consists of all the privileges enabled in the Default column of the Create User or Alter User dialog.

Object Permissions

An *object permission* defines a capability related to a specific object, such as a database or a table. Object permissions are assigned to selected groups, roles, or users. Object permissions are also called grants, permits, or object privileges.

The owner of the object can grant and revoke object permissions and can grant other users the privilege to grant permission on the object. The granting of permissions is typically the responsibility of the DBA.

Using permissions, data access can be restricted in several ways. Grants on objects can range from general to specific.

Permissions are classified according to the type of objects they affect. Object types include:

- Database
- Table
- View
- Procedure
- Database event
- Role
- Current installation

Working with Grants

You can perform the following basic operations on grants (object permissions):

- Grant any permission allowed for a particular object type to any group, role, or user (including public, which encompasses all current and future users)
- View all types of object permissions granted to a particular group, role, or user, or view the permissions granted for a particular object type
- Revoke a previously granted permission

In SQL, you can accomplish these tasks using the GRANT and REVOKE statements.

In VDBA, you can access grants in a number of ways using the Database Object Manager. For example, if you expand the branch for a group, role, or user object, there is a Grants sub-branch where you can access all permissions that have been granted to that particular group, role, or user. You can also expand the branch for other object types, such as a database or a table, and use the associated Grantees… sub-branch to access all groups, roles, and users that have been granted each permission allowed for that type of object. For the detailed steps for performing these procedures, see online help.

Object Ownership and Granting Object Permissions

When you create an object, you become the owner of that object.

As the owner, you are automatically entitled to grant and revoke permissions for the object (with views, you must also own the base tables). When you grant permissions for an object (other than a database) to another user, you can also grant permission for that user to grant permissions for the object to other users, and you can likewise revoke that permission if necessary.

The GRANT Statement

The GRANT statement is used to grant permissions. This statement has the general form:

```
GRANT privilege ON object TO whom
```

The full syntax of a GRANT statement is:

```
GRANT ALL [PRIVILEGES] | privilege {, privilege}
   [ON [object_type] [schema.]object_name {, [schema.]object_name}]
   TO PUBLIC | [authorization_type] auth_id {, auth_id} [WITH GRANT OPTION];
```

The default object type is TABLE, which is used for any table or view. The default authorization type is USER.

Authorization identifiers specify who is receiving the permissions. Authorizations can be specified for:

- Individual users

  Permissions defined by a particular GRANT statement can be issued to one or more end users, specifying the login user identifier.

- The key word PUBLIC, which includes all users. The authorization type PUBLIC is not followed by any auth_ids.

  For example: Grant all query permissions for the games table to all sessions:

  ```
  GRANT ALL ON games TO PUBLIC;
  ```

- A defined group
- A defined role

Authorizations for the individual user and PUBLIC are always in effect but can be adjusted by group and role permissions.

For complete information on the GRANT statement, see the *SQL Reference Guide*.

Database Grants

Database permissions are defined on the database as a whole. They set a number of limits that affect the *authorization identifiers* (that is, groups, roles, users, or public) specified when the grant is defined.

Most of the database permissions are *prohibiting* permissions—if not specified, the default is no restrictions. Prohibiting permissions, even if defined, are not enforced for the owner of the database or for any user with the security privilege, such as the system administrator.

Note: To override the default for database permission, create a grant for the permission that specifies the grantee as public.

The valid database permissions are as follows:

Access

Enables grantees to connect to the database.

Default: All authorization identifiers can connect to all public databases.

Private databases can be accessed only by users who are explicitly granted permission to access them. Permission to access a private database can be granted in the following ways:

- Using a database grant
- Enabling the database under Access to Non-Granted Databases in the appropriate dialog (for example, the Create User dialog)

Connect_time_limit

Specifies the maximum time (in seconds) that a session can consume.

Default: No connect time limit

Create_procedure

Enables grantees to create database procedures in the database.

Default: All authorization identifiers can create database procedures.

Create_table

Enables grantees to create tables in the database.

Default: All authorization identifiers can create tables.

Db_admin

Gives grantees unlimited database privileges for the database and the ability to impersonate another user (using the -u flag).

Default: Granted to the owner of the database and to any user with the security privilege, such as the system administrator. For all other users, the default is not to allow unlimited database privileges.

Idle_time_limit

Specifies the maximum time that a session can take between issuing statements.

Default: No idle time limit

Lockmode

Enables grantees to issue the set lockmode statement.

Default: All authorization identifiers can issue the set lockmode statement.

Query_cost_limit

Specifies the maximum cost per query on the database, in terms of disk I/O and CPU usage.

Default: All authorization identifiers are allowed an unlimited cost per query.

Query_cpu_limit

Specifies the maximum CPU usage per query on the database.

Default: All authorization identifiers are allowed unlimited CPU usage per query.

Query_io_limit

Specifies the maximum number of I/O requests per query on the database.

Default: All authorization identifiers are allowed an unlimited number of I/O requests.

The database privileges query_io_limit and query_row_limit are enforced based on estimates from the Ingres query optimizer. If the optimizer predicts that a query can require more I/O operations or return more rows than are allowed for the session, the query is aborted prior to execution. This prevents resource consumption by queries that are not likely to succeed.

Query_page_limit

Specifies the maximum number pages per query on the database.

Default: All authorization identifiers are allowed an unlimited number of pages per query.

Query_row_limit

Specifies the maximum number of rows returned per query on the database.

Default: All authorization identifiers are allowed an unlimited number of rows per query.

Select_syscat

Allows a session to query system catalogs to determine schema information.

Default: Sessions are allowed to query the system catalogs.

Session_priority

Determines whether a session is allowed to change its priority, and if so what its initial and highest priority can be.

Default: A session cannot change its priority.

Table_statistics

Allows grantees to view and create database table statistics.

Default: All authorization identifiers can view and create table statistics.

Update_syscat

Allows grantees to update system catalogs.

Default: No authorization identifier can update system catalogs.

Preventing Permissions—Each permission has a corresponding *preventing permission* to specifically disallow the permission. For example, to prevent access to the database, specify the Noaccess permission.

How Database Permissions for a Session are Determined

The database permissions for a session are calculated when the session connects to the database and remain in effect for the duration of the session. If, after a session connects to a database, the database permissions for one of that session's authorization identifiers are changed, the active session is not affected. Any new sessions that are established with the same authorization identifiers are subject to the revised database permissions.

Database Grant Examples

Here are examples of granting permissions on a database:

1. Define a query row limit of 100 rows on the new_accts database for user Ralph:

   ```
   GRANT QUERY_ROW_LIMIT 100
   ON DATABASE new_accts TO ralph;
   ```

2. Prohibit group *prodrams* from creating tables and database procedures in the new_accts database:

   ```
   GRANT NOCREATE_TABLE, NOCREATE_PROCEDURE
   ON DATABASE new_accts TO prodrams;
   ```

3. A database privilege can be superseded by issuing a subsequent GRANT statement for the user authorization. For example, assume that user karenk has been granted a query row limit of 1000 rows on the customers database:

   ```
   GRANT QUERY_ROW_LIMIT 1000
   ON DATABASE customers TO karenk;
   ```

   Her job changes and she does not need to access so much of the database, so the DBA issues a new GRANT statement giving her a query row limit of 250:

   ```
   GRANT QUERY_ROW_LIMIT 250
   ON DATABASE customers TO karenk;
   ```

   This new privilege replaces the old 1000-row privilege. If the DBA subsequently revokes the new limit:

   ```
   GRANT NOQUERY_ROW_LIMIT
   ON DATABASE customers TO karenk;
   ```

   karenk’s query row limit privilege for the database becomes undefined (the old limit of 1000 is not re-established). At this point if no value for QUERY_ROW_LIMIT has been defined for any of the other authorization identifiers associated with karenk’s session, then the number of rows that her session’s queries can return is unrestricted.

Table and View Grants

Ingres allows data sharing and updating if users have been issued grant permissions on the tables or views used in the query.

Table and view permissions are *enabling* permissions—if no permission is granted, the default is to prohibit access. Table and view permissions are not enforced for the owner of the table or view.

Permissions on Tables and Views

The following query permissions can be granted on both tables and views:

Select

Enables grantees to select rows from the table or view, for example using a SELECT statement or a WHERE clause.

Insert

Enables grantees to add rows to the table or view, for example using an INSERT statement.

Delete

Enables grantees to delete rows from the table or view, for example using a DELETE statement.

Update

Enables grantees to change existing rows in the table or view, for example using an UPDATE statement. An update grant can apply to all columns in the table or view, or only to specific columns.

Permissions on Tables

The following query permissions can be granted on tables only:

Copy_into

Enables grantees to copy the contents of the table to a data file, for example using the INTO clause of the COPY statement.

Copy_from

Enables grantees to copy the contents of a file to the table, for example using the FROM clause of the COPY statement.

References

Enables grantees to create tables that reference the table. A references grant can apply to all columns in the table, or only to specific columns.

If a user is not the owner and does not have the references permission on a table, that user cannot create a referential constraint that references the table.

Table Grant Examples

Here are examples of granting permissions on tables:

1. Grant select permission on the employee table to user freddy:

   ```
   GRANT SELECT ON employee TO freddy;
   ```

2. Grant select permission on the employee and department_table tables to individual users sally and ralph:

   ```
   GRANT SELECT ON employee, department_table
   TO sally, ralph;
   ```

3. Grant both select and update permissions on the employee table to user rollin. Note that you must be able to select values to update them:

   ```
   GRANT SELECT, UPDATE ON employee TO rollin;
   ```

4. Grant select and update permissions on the columns empname and empaddress in the employee table to users joank and gerryr:

   ```
   GRANT SELECT, UPDATE (empname, empaddress)
   ON employee TO joank, gerryr;
   ```

5. Grant references permission on the "address" table to user "joe":

   ```
   GRANT REFERENCES ON address TO joe;
   ```

6. Grant references permission on selected columns of the "finder" table to user "joe":

   ```
   GRANT REFERENCES ON finder (lname, finit, state)
   TO joe;
   ```

   User "joe" can then create a referential constraint on table "address" or the specified columns of table "finder." Note that he does not need the select permission to create the referential constraint.

7. Grant all query permissions (select, insert, update, delete, and references) on the phonelist table to all users:

   ```
   GRANT ALL ON phonelist TO PUBLIC;
   ```

Procedure Grants

For database procedures, the only valid permission is the Execute permission, which allows the grantees to execute the procedure.

Granting permission to execute a procedure makes database queries contained in the procedure code available to grantees. Granting execute permission to a database procedure also allows grantees to create rules that trigger the procedure.

The Execute permission is an enabling permission. By default, execution is prohibited unless the permission is specifically granted. This permission is not enforced for the owner of the procedure.

Permission to create procedures in the database is described in Database Grants (see page 49).

Database Event Grants

The valid database event permissions are summarized below:

Raise

Allows grantees to raise the database event (using the RAISE DBEVENT statement).

Register

Allows grantees to register to receive the database event (using the REGISTER DBEVENT statement).

These are enabling permissions—by default, execution is prohibited unless the permission is specifically granted. Database event permissions are not enforced for the owner of the event.

Role Grants

When a role is created, an implicit grant is issued on the role to the user creating the role.

Role access must be granted to other users (or public) before they can use the role. Role access is an enabling permission—by default, access to the role is prohibited unless the permission is specifically granted.

How Grants Restrict Data Access

Grants allow for data access to be restricted in the following ways:

- Operational restrictions (for example, Select, Insert, Update and Delete permissions applied to some or all of the columns of a table)
- Data value restrictions (data restrictions), which are implemented through views.
- Resource restrictions, which are permissions defined for the database as a whole, rather than individual tables or columns.

In a session where permissions are in effect, when you issue a query (for example, from an application or the SQL Scratchpad window in VDBA) the query is passed to the Ingres DBMS Server. Ingres then evaluates the grants on the tables involved in the query. If an operation does not pass an operational restriction, an error message is returned.

If an operation does not pass a data restriction, it means that views are being used and grants have been placed on the views, but the user authorization does not pass the grants on the data. In this case no error is returned, but the number of rows returned is affected. For example, if Mary is accessing a view that returns rows only from the Shoe department, then if she asks for information from the Toy department, no rows are returned.

Grant Overhead

Grants can affect query processing time. Queries for a table or view have overhead if:

- Permissions have been granted on the table or view
- Column-specific permissions are granted
- Many permissions are granted in general in the database

For the following, however, there is no overhead:

- For the table owner
- On certain public grants:
  - In select operations
  - Any operation for which all allowed permissions are specified for public
- If no permissions qualify (the query is simply aborted)

There is additional overhead during session initialization to evaluate database privileges for the authorization identifiers associated with the session. Because session initialization must read the catalogs in which groups, roles, and database privileges are stored, certain operations issued by the DBA or system administrator that write to these catalogs can be committed or rolled back as soon as possible. These operations include:

- Granting or revoking database privileges
- Creating, altering, or dropping a group
- Creating, altering, or dropping a role

Multiple Permission Checks

Multiple permissions can apply to the same query, because the system catalog is scanned for all possible permissions that apply. Generally, this means the broadest grant applies. The hierarchy of evaluation is described in more detail below, but the hierarchy is generally not something the DBA needs to formally consider.

For example, assume that grants have been created to allow all permissions on the employee table to public, and that a grant has been created to allow a particular user, Susan, the select privilege on the employee table. Susan, as part of the public, can perform all operations on the employee table, even though her individual grant was only for select permission.

Note: If you want more restrictive grants to apply, the solution is to drop the inclusive grants to public, and define specific grants for specified groups or users.

How Privileges for a Session Are Determined

In any session, the privileges in effect for that session are derived from the privileges granted to the authorization identifiers (role, user, group, and public) associated with the session, and any applicable defaults. If a particular privilege is defined for more than one authorization identifier associated with a session, then a hierarchy is used to determine which defined privilege is enforced for that session.

The authorization hierarchy, in order of highest to lowest precedence, is:

1. role
2. user
3. group
4. public

For each accessed object in a session, there is a search for a defined privilege that specifies that object and the desired access characteristics (for example, Select, Insert, Execute, and so on).

Access to Tables, Views, or Procedures and the Authorization Hierarchy

If the specified object attempting to be accessed is a table, view, or database procedure, then one of the authorization identifiers in effect for the session must have the required privilege for that object in order for the session to access that object. In the case of these *granted* privileges that are otherwise restricted, the authorization identifiers are searched for one that gives the required authorization.

For example, to insert into a specified table, one of the authorization identifiers associated with the session must have the Insert permission defined for the specified table. If none of the authorization identifiers associated with the session has this permission and the user does not own the table, then the internal default is used. In this case, because the internal default for the Insert permission is not to allow inserts, inserts are not allowed into the specified table.

Access to Databases and the Authorization Hierarchy

When the specified object attempting to be accessed is the database, the authorization hierarchy is also important because the privileges defined on the database can be defined with different values for different authorization identifiers. When a database privilege is defined at differing levels, the hierarchy is used to determine which privilege to enforce.

For example, assume that query row limits have been defined differently for each authorization level as follows:

Authorization Identifier	Query Row Limit
The role identifier	1700
The user	1500
The group identifier	2000
The public	1000

If a user starts a session and specifies both group and role identifiers, the limit defined for the role is enforced because it has the highest order of precedence in the hierarchy, giving the session a query row limit of 1700.

Several other possible scenarios are described below:

- If no query row limit was defined for role, then the query row limit defined for that user is enforced, which is 1500 rows. This is also the case if the user had not specified a role identifier.
- If no query row limit was defined for that user, then the query row limit defined for the group (2000 rows) is enforced.
- If no query row limit was defined for group, or if the user had not specified a group identifier, then the query row limit defined for public (1000 rows) is enforced.
- If none of the identifiers had a query row limit defined, the internal default is enforced, which in this case is an unlimited numbers of rows.

Note: In cases where multiple authorizations apply, the resource limit associated with the highest order of precedence applies, not necessarily the one that grants the most resources.

How Database Privileges for a Session Are Determined

The authorization hierarchy (see page 58) is used to determine the session's database privileges. The hierarchy includes the privileges granted to the authorization identifiers in effect for the session, and the internal defaults.

When a user begins a session:

- The privileges in effect for that session are derived from the privileges defined for the user identifier and for public. For example, while you might have the privilege to select all the tables in the database, you might only have the update permission on a limited number of those tables. If the user includes the -G or -R flag, or both, on the command line when beginning the session, then the privileges for the specified group or role identifier are also in effect for the session.
- If the user has a default group identifier defined for the user ID, when the user begins a session without specifying a group identifier, the default group identifier is automatically applied to the session. A default group identifier can be specified for a user when a user object is created or modified.

  For more information on the command line flags, -G and -R, see the *Command Reference Guide*.

Dbmsinfo—View Permissions for Current Session

You can use the dbmsinfo function to obtain the current value of any database privilege in effect for the current session.

To issue a dbmsinfo request, use the following syntax:

```
select dbmsinfo('request_name');
```

The *request_name* can be any of the following parameters:

connect_time_limt

The session’s value for the connect time limit, or -1 if none

create_procedure

"Y" if the session has create procedure privileges or "N" if not

create_table

"Y" if the session has create table privileges or "N" if not

db_admin

"Y" if the session has the db_admin privilege or "N" if not

idle_time_limit

The session's value for the idle time limit or -1 if none

lockmode

"Y" if the session can issue the set lockmode statement or "N" if not

query_cost_limit

The session's value for the query cost limit or -1 if none

query_cpu_limit

The session's value for the CPU limit or -1 if none

query_io_limit

The session's value for the query I/O limit or -1 if none

query_page_limit

The session's value for the query page limit or -1 if none

query_row_limit

The session's value for the query row limit or -1 if none

session_priority

The session's current priority or -1 if none

select_syscat

"Y" if the session has the select_syscat privilege or "N" if not

table_statistics

"Y" if the session has the table_statistics privilege or "N" if not

update_syscat

"Y" if the session has the update_syscat privilege or "N" if not

Example: Return the Value of Query Row Limit for Current Session

Assuming the Query_row_limit permission for the current session is 50, the following query returns the value “50” in *x*:

```
select x = dbmsinfo('query_row_limit') as x;
```

Note: The dbmsinfo function allows other *request_name* values relating to other aspects of the current session. For details, see the chapter “Transactions and Error Handling” in the *SQL Reference Guide*.

Chapter 6: Implementing Security Auditing

This section contains the following topics:

Security Alarms

Security alarms allow you to specify the events to be recorded in the security audit log for individual tables and databases. Using them, you can place triggers on important databases and tables to detect when users attempt to perform access operations that are not normally expected.

For tables, you can monitor the success or failure of any of the following events:

- Select
- Delete
- Insert
- Update

For databases, you can monitor the success or failure of these events:

- Connect
- Disconnect

Security alarm events are considered successful if the user succeeds in performing the specified type of access. If a particular query triggers a security alarm event, however, it does not necessarily mean that the query completed successfully. It simply means that the security access tests for the specified types of events (for example, select, delete, insert, and update) were passed.

Failure of a security alarm event means that the user attempted to perform the associated operation and failed for some security-related reason. For example, a user can fail to gain access to a table or a database because he or she lacks the required permissions. A query or database operation might fail for other reasons, unrelated to security, but these failures do not trigger the associated security alarm event.

Security alarms can be assigned to specific authorization identifiers (individual users or the public, and groups and roles) so that you can limit monitoring to certain users. You can also specify a database event to be raised when a security alarm is triggered. Database Event Grants (see page 55) describes the database event permissions required to raise an event.

Working with Security Alarm Objects

When working with security alarms, you can do the following:

- Create security alarm objects of various types for specific tables and databases
- View existing security alarm objects, including the detailed properties of each individual object
- Drop security alarm objects

You can accomplish these tasks using the SQL statements CREATE SECURITY_ALARM, HELP SECURITY_ALARM, and DROP SECURITY_ALARM. For complete details on these statements, see the *SQL Reference Guide*.

In VDBA, use the Security Alarm branch in the Database Object Manager window. For detailed steps, see the Procedures section of VDBA online help.

How to Implement a Security Alarm

To implement a security alarm, follow these basic steps:

1. Create the security alarm. Issue the CREATE SECURITY_ALARM statement to define the conditions that will trigger the alarm. For example:

   ```
   CREATE SECURITY_ALARM ON TABLE employees IF FAILURE;
   ```

   (In VDBA, use the appropriate Security Alarm branch in the Database Object Manager window.)

2. Have an authorized user issue the ENABLE SECURITY_AUDIT ALARM statement to enable auditing of security alarms. You can also use ENABLE SECURITY_AUDIT (see page 66) to specify other types of auditing.

   When user access to the specified database or table triggers the alarm, a record is written to the audit log and the associated database event, if any is defined, is raised.

To drop a security alarm

1. Issue a HELP SECURITY_ALARM statement to obtain the security alarm number. For example:

   ```
   HELP SECURITY_ALARM employees;

   Security alarms on employees are:

   Security alarm 2:
   create security_alarm on table employees if failure
   ```

2. Issue a DROP SECURITY_ALARM statement. For example:

   ```
   DROP SECURITY_ALARM ON employees 2;
   ```

Security Alarm Example

A typical scenario is to audit all accesses to databases and security-relevant events (such as the creation and deletion of users and the granting of special privileges). The Ingres security administrator, however, may decide that although access to certain tables should be monitored, imposing a general auditing control on all tables is not desired.

In this example assume that:

- Table “addresses” contains a list of addresses is to be audited. Updates or changes to existing information are to be recorded in the audit log.
- Table “all_summary”, a large database table, is used infrequently. Accesses are to be audited to determine whether it should be archived and deleted.

The following statements could be issued to audit security-related events:

```
ENABLE SECURITY_AUDIT SECURITY;
ENABLE SECURITY_AUDIT USER;

CREATE SECURITY_ALARM ON TABLE addresses
WHEN INSERT, UPDATE, DELETE;

CREATE SECURITY_ALARM ON TABLE all_summary;
```

Security Auditing

Security auditing is the recording of all or specified classes of security events for the entire Ingres installation.

Selected classes of events, such as use of database procedures or access to tables, can be recorded in the security audit log file for later analysis. Criteria can be selected that apply to a single object or across an entire class of installation objects.

Security auditing is controlled by a user with the maintain_audit privilege.

Related Information

Auditor Privilege (see page 40)
Maintain_Audit Privilege (see page 41)

Audit Focus

The information in the audit log can quickly grow in volume. You can achieve maximum benefit of security auditing by focusing the audit information produced by the system.

Security auditing has an impact on resource consumption. Audit records are recorded in a shared buffer before being written to the audit page and then to the log file. The performance impact of security auditing should be tested thoroughly before implementation.

Focusing the audit information both reduces resource consumption and makes it easier to examine the logs for possible security infringements.

The coarse and fine selection criteria can be used together to create a suitable security-auditing environment that meets the needs of any security administrator.

How to Enable Security Auditing

By default, security auditing is disabled. You must enable security auditing by setting the security_auditing configuration parameter. In addition, you must specify the level of auditing using the ENABLE SECURITY_AUDIT statement.

To enable security auditing follow these steps:

1. In CBF, select Security, Configure, Auditing.

   The Configure Security Auditing screen appears.

2. Scroll to security_auditing. Select Edit to toggle the setting to ON.
3. (Optional) Tab to the Audit log files and use the Edit function to change the location and names of the security audit log files.
4. Connect to the Ingres master database iidbdb as the installation owner or security administrator.
5. Issue statements similar to the following to enable the level of security auditing.
   - To enable security auditing on all operations by all users, installation wide:

     ```
     ENABLE SECURITY_AUDIT ALL
     ```

   - To enable query text auditing by a specific user:

     ```
     ALTER USER username WITH SECURITY_AUDIT=(QUERY_TEXT)
     ```

How to Verify Security Auditing Levels

You can verify security auditing levels by querying the appropriate system catalog.

To check that security auditing was enabled on all operations by all users, installation wide, follow these steps:

1. Log on as the installation owner.
2. Connect to Ingres master database iidbdb.

   Query the system catalog iisecurity_state by issuing the following command:

   ```
   SELECT STATE FROM iisecurity_state
      WHERE NAME = 'All';
   ```

   The value returned should be E (enabled).

To check that query text auditing was enabled by a specific user, follow these steps:

1. Log on as the installation owner.
2. Connect to Ingres master database iidbdb.
3. Query the system catalog iiusers by issuing the following command:

   ```
   SELECT AUDIT_QUERY_TEXT FROM iiusers
   WHERE USER_NAME = ‘username’;
   ```

   The value returned should be Y.

Security Auditing Configuration Parameters

A security audit log file is created as part of the installation process. Audit records are recorded in a shared buffer before being written to the audit page and then to the log file. The auditing derived parameters will affect performance so concurrent performance testing is advised before implementing security auditing.

The security auditing configuration parameters are as follows (for more detail, see the online help for CBF or Configuration Manager):

audit_mechanism

Used for the auditing destination.

Default: INGRES

log_page_size

Specifies the page size of each audit log page

max_log_size

Specifies the maximum log size in kilobytes

on_error

Specifies the audit action to take on an error, either SHUTDOWN or STOPAUDIT

on_log_full

Specifies the audit action to take on “log full” condition, either SHUTDOWN or STOPAUDIT

on_switch_log

Specifies the full path of the utility to execute when an audit log is full or before a new log is initialized

security_auditing

Specifies whether security auditing is ON or OFF

Audit log files

Specify the full path of each audit log

Security Audit Statements

Levels of security auditing are enabled and disabled with the SQL statements ENABLE SECURITY_AUDIT and DISABLE SECURITY_AUDIT.

To use these statements, you must have the maintain_audit privilege and be connected to the iidbdb database.

Keywords on these statements allow you to specify the types of security events you want to audit. For example:

- DATABASE—to control logging of database access
- PROCEDURE—to control logging of procedure access
- ALL—to control logging of all possible security events

The events specified using these statements are known as the *default events*, which is a term that applies when specifying auditing levels for users, profiles, and roles, as described in the next section.

For complete syntax and keywords for these statements, see the *SQL Reference Guide*.

Security Audit Levels for Users and Roles

Security audit levels can also be specified for individual users (directly or through a profile) and for roles (requires the maintain_audit privilege). You can specify the security audit level whenever you create or modify the user, profile, or role.

By default, users are audited for default events (as specified by the security audit statements). However, you can specify that a user be audited for all events and even that the query text associated with the user's queries be audited.

Important! Because query text auditing is detailed and takes up a lot of space in the security log file, it must be explicitly enabled at the user level and using ENABLE SECURITY_AUDIT QUERY_TEXT. Otherwise, no query text auditing can take place.

By default, roles are audited according to the settings for the individuals using the role. However, because a role can give a user privileges the user does not otherwise have, you can specify that anyone who uses a role be audited for all events while using the role, regardless of that user's audit state.

Note: Default auditing levels (as well as other default user and role attributes) are determined by the default profile. If the default profile is modified, the defaults stated in this section do not apply.

Changes to Security Audit Status During a Session

The security status for a session is determined at the time of initial connection. Thereafter, during the session:

- If the auditing level of a user, profile, or role is changed from default auditing to auditing all events, or vice versa, the change in status can apply only to new sessions connecting after the change has been made.
- All other security-auditing related changes take effect immediately.

Access to the Security Audit Log

Access to the security audit log is established through registering the security audit log file as a virtual table. After it has been successfully registered, the security audit log file can be queried as any other table.

Registering the Security Audit Log File

To access the security audit log file contents with SQL query statements, you must first register the audit log file as a virtual table using the REGISTER TABLE statement with the DBMS=SXA clause.

The following statements, for example, make a subset of the security audit log file sal1.log available through the table sal1:

```
REGISTER TABLE sal1(
    database      CHAR(24) NOT NULL,
    audittime     DATE     NOT NULL,
    user_name     CHAR(24) NOT NULL,
    auditstatus   CHAR(1)  NOT NULL,
    auditevent    CHAR(24) NOT NULL,
    objecttype    CHAR(24) NOT NULL,
    objectname    CHAR(24) NOT NULL,
    description   CHAR(80) NOT NULL
    )
  AS IMPORT FROM 'sal1.log'
  WITH DBMS = SXA;
```

The REGISTER TABLE statement, when used to register the security audit log file, requires the auditor privilege.

When the virtual table is no longer needed, a user with the auditor privilege can use the REMOVE TABLE statement, specifying the name of the virtual table created using REGISTER TABLE.

To display information on registered objects, use the HELP REGISTER statement.

For the complete syntax (including specifications for the security log audit file format) for the REGISTER TABLE and REMOVE TABLE statements, see the *SQL Reference Guide*.

Querying the Registered Virtual Table

After the security audit log is registered, any user with the auditor privilege can perform queries on the registered virtual table to view its contents.

For example, to obtain all events by the user spy against the database securedb, query the table sal1 as follows:

```
SELECT audittime, auditstatus, auditevent,
    objecttype, objectname, description
  FROM sal1
  WHERE DATABASE = 'securedb' AND user_name = 'spy'
  ORDER BY audittime;
```

The result of the query might be similar to the following:

```
       audittime  auditstatus  auditevent  objecttype  objectname  descrpt
01-Jan-2008 01:00            N      SELECT       TABLE    salaries  Attempt to
                                                                   access a TABLE
                                                                   01-Jan-2008
```

Obtain the Current Audit File Name

The dbmsinfo function can be used to find the name of the audit log file. You must have the auditor privilege to use this call.

To obtain the name of the current security audit log file

Issue the following statement:

```
dbmsinfo('security_audit_log')
```

The function returns the file name only, not the full file specification.

Alternatively, you can access the current file through the iiaudit system catalog, in the iidbdb system database.

Chapter 7: Controlling Access through Database Procedures

This section contains the following topics:

Database Procedures

A *database procedure* is a set of SQL statements and control statements in a begin/end block that are stored as a unit in the database. It usually contains at least one query into the database, which is stored in compiled form with a Query Execution Plan.

A database procedure can have the Execute permission granted on it.

Database procedures provide the following security benefits:

- They provide an extra level of control over data access and modification.
- They can be used with security alarms to enhance the security-auditing features.

Working with Procedure Objects

You can perform the following basic operations on database procedures:

- Create database procedures

  Note: By default, any user can create a database procedure, but this ability can be restricted using database permissions.

- View existing database procedures, including the detailed properties of each individual object
- Drop database procedures

In SQL, you can accomplish these tasks using the CREATE PROCEDURE, HELP PROCEDURE, and DROP PROCEDURE statements. For details on these statements, see the *SQL Reference Guide*.

In VDBA, database procedures are implemented using the Procedures branch for a particular database in the Database Object Manager window. For detailed steps, see the Procedures section of online help.

How to Implement a Database Procedure

To implement a database procedure, follow these basic steps:

1. Create the procedure using the CREATE PROCEDURE statement. You can do this interactively or in Embedded SQL. (In VBDA, use the appropriate Procedures branch in the Database Object Manager window.)
2. Grant Execute permission on the database procedure to specified users, groups, or roles, as described in Object Permissions (see page 46).
3. Invoke the database procedure by issuing an EXECUTE PROCEDURE statement, firing a rule, or triggering a security alarm. Any user who has been granted Execute permission can perform this step.

Database Procedure Example

The following database procedure accepts as input an employee ID number. The employee matching that ID is moved from the employee table and added to the emptrans table.

```
CREATE PROCEDURE move_emp
   (id INTEGER NOT NULL) AS
BEGIN
   INSERT INTO emptrans
      SELECT * FROM employee
         WHERE id = :id;
   DELETE FROM employee
      WHERE id = :id;
END;
```

Access Control through Database Procedures

Database procedures provide the DBA with greater control over database access.

The DBA can grant permission to execute a database procedure even if the user has no direct access to the underlying tables referenced in the procedure. With Execute permissions, the DBA can give users limited, specific access to tables without needing to give the users full query grants (such as SELECT) on the tables. In this way, the DBA controls exactly what operations a user can perform on a database.

For example, both tables used in the previous example can be inaccessible to users except through the procedure. The DBA grants Execute permission, as in the following example, to allow users in the acctg group to access the tables for this procedure only:

```
GRANT EXECUTE ON PROCEDURE move_emp TO acctg
```

When the procedure is invoked, the executing application passes a single integer parameter.

For example, the following statement calls the move_emp procedure for the employee ID “56742”:

```
EXEC SQL EXECUTE PROCEDURE move_emp (id = 56742);
```

Chapter 8: Implementing PAM in Ingres

This section contains the following topics:

What Is PAM?

The Pluggable Authentication Module (PAM) allows applications (in this case, Ingres) to authenticate users with a plug-in mechanism, as an alternative to the default authentication mechanism of a Linux or UNIX environment. Each plug-in is a shared library that is dynamically loaded into the PAM module.

PAM provides plug-ins for authentication at the operating system level or using third-party solutions, such as LDAP or Kerberos.

PAM allows applications such as Ingres to authenticate users transparently, regardless of the underlying authentication mechanism. Applications do not have to be recompiled or reconfigured if your authentication mechanism changes.

The Ingvalidpam Program

The Ingres ingvalidpam program supports authentication through PAM.

Ingvalidpam is a password validation program that can be used instead of the ingvalidpw program. Like ingvalidpw, ingvalidpam is used only in Linux and UNIX environments. If the DBMS Server runs on Linux or UNIX, the Ingres client can run on any platform and PAM can be used to authenticate.

Requirements for Using PAM

The following are required to use PAM in Ingres:

- You must have root access to edit configuration files.
- Your machine must have PAM installed.

  You can download PAM from the vendor site for your operating system. For example, for Solaris see http://sun.com/software/solaris/pam.

You may need to build and install PAM if no binaries are available for your operating system. If your machine does not have a robust C compiler, you may need to download and build gcc from http://gcc.gnu.org/releases.html. As a prerequisite to gcc, you may also need to download gmake from http://gnu.org/software/make.

Build the Ingvalidpam Program

The invalidpam executable and source is included in the Ingres distribution. In most cases, the executable works fine as delivered, but it can be built from the source, if necessary.

The following .c and .h files, which are required to build the ingvalidpam executable, are available in the directory $II_SYSTEM/ingres/files/iipwd:

- ingvalidpam.c
- ingpwutil.c
- ingpwutil.h

To build the invalidpam executable

1. Change directory as follows:

   ```
   cd $II_SYSTEM/ingres/files/iipwd
   ```

2. Issue the following command:

   ```
   cc -o $II_SYSTEM/ingres/bin/ingvalidpam ingvalidpam.c inpwutil.c –lpam
   ```

   The ingvalidpam executable is created.

To enable password validation through the ingvalidpam program

1. Change directory:

   ```
   cd $II_SYSTEM/ingres/bin
   ```

2. Log in as root.

3. Change the group ownership of the ingvalidpam file:

   ```
   chgrp shadow ingvalidpam
   ```

4. Change permissions on the ingvalidpam file:

   ```
   chmod g+s ingvalidpam
   ```

5. Set the II_SHADOW_PWD environment variable to ingvalidpam:

   ```
   ingsetenv II_SHADOW_PWD $II_SYSTEM/ingres/bin/ingvalidpam
   ```

   Shadow password validation through PAM is enabled.

6. Ensure the PAM configuration file named "ingres" exists at this location:

   Linux: /etc/pam.d

   UNIX: /etc/pam.conf

How to Implement Standard Linux or UNIX Security Using PAM

To use the ingvalidpam program for standard operating system user authentication on Linux or UNIX, you can either use the supplied ingvalidpam program or compile the program, if necessary.

If using the supplied ingvalidpam program

Follow the steps under "To enable password validation through the ingvalidpam program" in Build the ingvalidpam Program (see page 80) to enable shadow password validation through PAM. The ingvalidpam executable is located in $II_SYSTEM/ingres/bin.

If compiling ingvalidpam.c

Follow all steps in Build the ingvalidpam Program (see page 80).

Ingres PAM Configuration File (For Linux or UNIX)

The contents of the PAM configuration file for some UNIX environments are shown here. (For field descriptions, refer to the PAM documentation.)

SuSE Linux

```
% cat /etc/pam.d/ingres
#%PAM-1.0
auth     include        common-auth
account  include        common-account
```

Redhat Linux

```
$ cat /etc/pam.d/ingres
#%PAM-1.0
auth       required     pam_unix.so
account    required     pam_unix.so
```

HP-UX and Solaris

The service name OTHER can be used or "ingres" can be added to /etc/pam.conf, which will use the pam_unix.1 module.

```
% cat /etc/pam.conf
#ident  "@(#)pam.conf 1.19     95/11/30 SMI"
#
# PAM configuration
#
# Authentication management
#

OTHER    auth         required       /usr/lib/security/libpam_unix.1
OTHER    account      required       /usr/lib/security/libpam_unix.1
```

How to Implement LDAP Authentication Using PAM

The LDAP module of PAM authenticates clients using the TLS (Transport Layer Security) protocol. The TLS protocol uses encrypted keys and certificates instead of cleartext user names and passwords for authentication.

Follow this process to implement LDAP authentication through PAM:

1. Update the ldap.conf file with the appropriate entries.
2. Update the ingres PAM configuration file with the appropriate entries.
3. Update netutil with the ingvalidpam user names and passwords.

LDAP Requirements

Your LDAP environment must have a working slapd server and directory of users. Your LDAP environment must also support LDAP version 3.

The ldap.conf File—Configure LDAP Daemon (slapd)

For the LDAP module to know how LDAP authentication is to be performed, client LDAP processes must refer to an ldap.conf file. In this case, the ingvalidpam program is the only client process that references the ldap.conf file, therefore, the ldap.conf file needs to be configured on the server side only. Client machines do not need to configure ldap.conf.

By default, ldap.conf resides in the /etc directory, but you can override the path and file name of ldap.conf with the LDAPCONF environment variable.

By default, the PAM LDAP module searches the slapd database for object classes of posixAccount. A Distinguished Name for a posixAccount user might look like this:

```
uid=johnDoe,ou=people,dc=myDomain,dc=com
```

In ldap.conf, you define a BASE attribute of:

```
ou=people,dc=myHost,dc=com
```

This shorthand allows you to define only "johnDoe" as the user name in netutil instead of the entire Distinguished Name.

The only other attribute required is the HOST, which is defined as the FQDN (fully qualified domain name) of the server as defined by TCP/IP. Therefore, your ldap.conf file would look like this:

```
HOST myHost.myDomain.com
BASE ou=people,dc=myDomain,dc=com
```

Note: PAM expects ldap.conf to reside in /etc, and ignores the LDAPCONF variable. So if your ldap.conf directory resides elsewhere, set up a symbolic link:

```
ln -s /etc/openldap/ldap.conf /etc/ldap.conf
```

Browse slapd Database

To see the format of the Distinguished Names in the user database, you can browse the database using the ldapsearch command.

To list the contents of the slapd database

Issue the following command:

```
ldapsearch '(objectclass=*)' -H ldap://myHost.mydomain.com:389 -b
"dc=myDomain,dc=com" -x
```

Note: You need to know the root domain name of the slapd database, defined above as "dc=myDomain,dc=com". In many cases, it corresponds to the domain and domain suffix of your host name. If this approach does not work, consult your system administrator.

The Ingres PAM Configuration File (for LDAP)

The PAM component of the ingvalidpam program references a service configuration file named /etc/pam.d/ingres. For PAM over LDAP, the following entries are required:

```
auth     sufficient    pam_ldap.so
auth     sufficient    pam_nologin.so
account  sufficient    pam_ldap.so
```

Note: The ldap.conf and ingres files are owned by root, but must be world-readable. Execute chmod 644 on these files to ensure they have the correct permissions.

Active Directory Configuration

The Active Directory authentication is configured almost the same way as slapd, but with a few additions.

By default, the PAM LDAP module binds anonymously and looks for login attributes of type "uid". The previously described ldap.conf configuration may work for Active Directory servers if the AD server allows anonymous binds and that the authentication accounts of interest are of type "posixAccount", or at least an object with a "uid" attribute.

Often, Active Directory servers do not allow anonymous binding, and the object classes of the user database do not include "posixAccount". So, the ldap.conf file must include a user name and password for binding purposes and a directive to look for an attribute other than "uid".

Adding these entries allows ingvalidpam to authenticate against the Active Directory, assuming they have the login attribute sAMAccountName:

```
binddn  CN=proxySearch,OU=myCity,OU=USA,OU=Americas,DC=myDomain,DC=com
bindpw mySecretPassword
pam_login_attribute sAMAccountName
```

Since the bind domain and password are presented in cleartext in a world-readable file, the user "proxySearch" is created to perform the Active Directory lookup. The permissions on "proxySearch" can be set so that the "proxySearch" user can only search the Active Directory. You can use your own user name and password for testing purposes.

Browse Active Directory Database

To find the format of Distinguished Names, you may have to browse the Active Directory database.

The ldapsearch command lists all of the contents of the Active Directory database.

To browse the Active Directory database

1. Issue the following command:

   ```
   ldapsearch -V -Y DIGEST-MD5 -H ldap://myHost.myDomain.com:389
   '(objectclass=*)'
   ```

   You can use the objectclass filter without wildcards to limit the search.

   You are prompted for a password.

2. Enter the password of your own Active Directory account.

Example—The following ldapsearch command browses the Active Directory for user "johnDoe@myDomain.com" with a sAMAccountName of "johnDoe" and can serve as a test of the ldap.conf configuration:

```
ldapsearch -x -W -D "johnDoe@myDomain.com" -LLL "(sAMAccountName=johnDoe)"
```

How to Implement Kerberos Authentication Using PAM

If your enterprise has no access to an Active Directory, you must configure and populate a Kerberos KDC (Key Distribution Center). For details, see http://web.mit.edu/Kerberos.

Follow this process to implement Kerberos authentication through PAM:

1. Update the krb5.conf file with the appropriate entries.
2. Update the ingres PAM configuration file with the appropriate entries.
3. Update netutil with the ingvalidpam user names and passwords.

Ingres Kerberos Driver versus Ingvalidpam

Use of PAM with Kerberos is less secure than the Ingres Kerberos driver and loses the single sign-on capability. The ingvalidpam environment uses Kerberos merely as an alternative to operating-system user names and passwords. The Ingres Kerberos driver, in contrast, is a more complete approach to using Kerberos for authentication. (For details on the Ingres Kerberos driver, see the chapter "Configuring Ingres to Use Kerberos.")

In the Kerberos driver environment, the process running the application must be recognized as a valid Kerberos service principal and be pre-authenticated with Kerberos tickets. The netutil database requires no user names and passwords for Kerberos connection targets.

In the ingvalidpam environment, you must specify the Kerberos user name and password in the netutil database. The process owner does not need to be pre-authenticated through Kerberos, and does not have to be recognized as a valid Kerberos service principal.

The krb5.conf File—Configure Kerberos

The krb5.conf file tells Kerberos clients where the Kerberos server is on the network. On Linux and UNIX machines, krb5.conf typically resides on /etc.

The following krb5.conf file contains the minimum configuration for a Kerberos or Active Directory server on myHost.myDomain.com:

```
[libdefaults]
    default_realm = MYDOMAIN.COM

[realms]
    INGRES.PRV = {
        kdc = MYHOST.MYDOMAIN.COM
        admin_server = MYHOST.MYDOMAIN.COM
    }

[domain_realm]
        myDomain.com = MYDOMAIN.COM
        .mydomain.com = MYDOMAIN.COM
```

Note: The Kerberos realm defined by MYDOMAIN.COM happens to be the same as the network domain name and extension, but the realm can have a different name.

The Ingres PAM Configuration File (for Kerberos)

The contents of the Ingres PAM file are very similar to the LDAP configuration, except Kerberos libraries are substituted for the LDAP libraries. The following configuration will work equally well for Active Directory and Kerberos KDS servers:

```
auth     sufficient    pam_krb5.so
auth     sufficient    pam_nologin.so
account  sufficient    pam_krb5.so
```

Note: The krb5.conf and ingres files are owned by root, but must be world-readable. Execute chmod 644 on these files to ensure they have the correct permissions.

Netutil Entries for Ingvalidpam

After you have the appropriate authentication mechanism configured with PAM, you can convert your clients to use the user names and passwords for ingvalidpam instead of the system or Ingres installation passwords, if they are different.

To update netutil with the ingvalidpam user names and passwords

1. Ensure that II_SHADOW_PWD is defined as $II_SYSTEM/ingres/bin/ingvalidpam using the following command:

   ```
   ingprenv II_SHADOW_PWD
   ```

2. Redefine II_SHADOW_PWD, if necessary:

   ```
   ingsetenv II_SHADOW_PWD $II_SYSTEM/ingres/bin/ingvalidpam
   ```

   Stop and then restart the Name Server, as follows:

   ```
   ingstop -iigcn
   ```

   ```
   ingstart -iigcn
   ```

3. Add user names and passwords appropriate to the authentication method in the netutil database. (For details, see the *Connectivity Guide*.)

   The Name Server will use ingvalidpam to authenticate.

Test Ingvalidpam

If you have problems authenticating after you have updated netutil with the user names and passwords, you can use the II_INGVALIDPW_LOG environment variable to troubleshoot.

To turn on logging to troubleshoot the connection

1. Stop the Name Server by issuing the following command:

   ```
   ingstop -iigcn
   ```

2. Set the II_INGVALIDPW_LOG environment variable to an appropriate path name:

   C shell:

   ```
   setenv II_INGVALIDPW_LOG /tmp/ingvalidpam.log
   ```

3. Restart the Name Server:

   ```
   ingstart -iigcn
   ```

   The results of ingvalidpam are listed in /tmp/ingvalidpam.log (or the path you defined).

4. Turn off logging when you are finished testing:

   C shell:

   ```
   unsetenv II_INGVALIDPW_LOG
   ```

Chapter 9: Using Data at Rest Encryption

This section contains the following topics:

What Is Data at Rest Encryption?

Data "at rest" refers to data on physical media recorded in a persistent form in Ingres database table, transaction log, journal, and checkpoint files.

Data at rest encryption allows specific database table columns to be encrypted. Data in the protected columns is stored on disk or other media in encrypted form and can only be accessed if the encryption passphrase is known.

Encrypted columns are stored in the database files using 128-, 192-, or 256-bit Advanced Encryption Standard (AES) encryption. A single AES key protects any data in a table that contains encrypted columns. The encryption is transparent to the applications accessing the data.

Data at rest encryption does not protect data outside of the database, which includes:

- Data passed back and forth to applications
- Transactions that implement data replication at a logical (vs. binary, journal application) level
- Files created using copydb

Note: If the security of data transmitted over a network is important, you can implement protection using other mechanisms such as public key encryption. Flat files containing sensitive information that is encrypted in the database should be stored in encrypted files or on encrypted media.

How Encryption Works

When an encrypted table is created, an AES key is randomly generated (or can be specified with the AESKEY= option). The key is then encrypted using an AES key derived from the specified PASSPHRASE. The AES encryption specified for the user data encryption (AES128, AES192, or AES256) is also used for the passphrase protection of the internal catalog-stored key.

After an encrypted table is created, access to the encrypted data must be enabled through a MODIFY statement that specifies the correct passphrase. At this point, an in-memory-only decrypted key is created for use by the encryption and decryption code. At server shutdown, this decrypted key is cleared and the encrypted data is effectively locked. At server startup, the MODIFY must be issued again to access the encrypted data.

A fixed number of encryption keys are available while the Ingres server is active. This value is controlled by the dmf_crypt_maxkeys configuration parameter in CBF. This number represents the largest number of encrypted tables that can be unlocked and accessed at any one time for all databases in the installation. Slots in this shared memory table are freed when a table is relocked or dropped, and when a database is deleted with destroydb.

The Power of Encryption

While the data at rest encryption feature does not secure data in all aspects of its life cycle, do not underestimate its power. Without proper care of the passphrase, the data owner himself can be locked out of the data!

With the passphrase, which unlocks the internal AES key, the data is transparently accessible. Without the passphrase, the logical data is unreadable, meaningless bit patterns.

Important! If you lose the passphrase, encrypted data remains inaccessible.

Transparent vs. Function-based Encryption

Column values can be encrypted at either of the following levels:

- Transparent column encryption (see page 93), done at the DBMS Server level

  If you want the Ingres server to handle encryption for the application and be assured that data at rest is encrypted, declare the columns as encrypted on the CREATE TABLE statement, where you also define an encryption passphrase.

  The passphrase applies to all encrypted rows and columns in the table.

- Function-based encryption (see page 96), done at the application level

  If you want to control the process and provide the passphrase at the application level, use the Ingres SQL functions AES_ENCRYPT and AES_DECRYPT.

  The passphrase can apply to one row.

You can combine the two levels, declaring encrypted columns on CREATE TABLE, and then storing application-encrypted data in them.

Transparent Column Encryption (DBMS Server-level Encryption)

To use DBMS Server-level encryption, use the CREATE TABLE statement to encrypt column values and define the encryption passphrase. To enable access to the table, use the MODIFY statement.

For syntax details, see the *SQL Reference Guide*.

Create an Encrypted Table

The following example creates an encrypted table, enables access to it, inserts rows, and then selects them:

```
CREATE TABLE socsec1
      (fname char(10),
       lname char(20),
       socsec char(11) encrypt nosalt
  WITH ENCRYPTION=AES256,
      PASSPHRASE='transparent encryption example';

MODIFY socsec1 ENCRYPT
  WITH PASSPHRASE='transparent encryption example';

SET TRACE POINT DM805;

INSERT INTO socsec1 VALUES ('John', 'Smith', '012-33-4567');

INSERT INTO socsec1 VALUES ('Lois', 'Lane', '010-40-1234');

INSERT INTO socsec1 VALUES ('Charlie', 'Brown', '012-44-9876');

SELECT * FROM socsec1;
```

The following results are returned:

```
+----------+--------------------+-----------+
|fname     |lname               |socsec     |
+----------+--------------------+-----------+
|John      |Smith               |012-33-4567|
|Lois      |Lane                |010-40-1234|
|Charlie   |Brown               |012-44-9876|
+----------+--------------------+-----------+
(3 rows)
```

The encryption is transparent to the application (in this case, the Ingres SQL terminal monitor), as shown by the plain text values in the socsec column.

The SET TRACE POINT DM805 statement sends a dump of encrypted buffers immediately after the encryption processing to II_DBMS_LOG. Trace point DM806 sends a dump of encrypted buffers immediately before decryption processing. The log shows the values that are stored for the social security numbers:

```
AES 256-bit encrypt blocks:
     1C90492C913D7D9195FED8507F0D1BFE >,I...}=.P.......<
AES 256-bit encrypt blocks:
     FF24FB9037A156F6D4CE57921F0EFD07 >..$..V.7.W......<
AES 256-bit encrypt blocks:
     94EE866C722BEA0AF096EF3D64347271 >l.....+r=...qr4d<
```

For transparent encryption, the value stored includes salt (see page 103) (if any) and a verifying hash, in addition to the user data itself.

Enable Access to Encrypted Data

Access to encrypted data in a table that contains encrypted columns is possible only when it is enabled after server startup with the MODIFY statement:

```
MODIFY socsec1 ENCRYPT
  WITH PASSPHRASE='transparent encryption example';
```

Disable Access to Encrypted Data

Perhaps a sensitive table is needed only during regular work hours, even though the DBMS runs continually. Disabling encryption and decryption at the end of the work day will prevent unauthorized access during the night shift.

To make the encrypted data inaccessible, disable the passphrase.

To disable the passphrase

Use the MODIFY statement, specifying an empty string for the passphrase:

```
MODIFY tablename ENCRYPT WITH PASSPHRASE='';
```

An attempt to access the data will result in the following message:

```
E_US24BF A query has been issued against a table for which encryption has not
been unlocked with the MODIFY statement. This unlock statement must be issued
every time the Ingres installation is restarted. Please contact your system
administrator.
```

Change the Passphrase

The circle of trust for the table may change, or the law may require that the passphrase change periodically.

To change the passphrase for encrypted data, use the NEW_PASSPHRASE option on the MODIFY statement.

To change the passphrase

1. Issue the MODIFY statement with the following options:

   ```
   MODIFY tablename ENCRYPT
       WITH PASSPHRASE='encryption passphrase',
           NEW_PASSPHRASE='new encryption passphrase';
   ```

2. Issue another MODIFY statement to enable the table with the new phrase:

   ```
   MODIFY tablename ENCRYPT WITH PASSPHRASE = 'new encryption passphrase';
   ```

   Issuing the second MODIFY re-enables access to encrypted data and ensures that the changed passphrase was typed correctly. We recommend doing this immediately, and before a COMMIT is issued. If a problem occurs, ROLLBACK to the old passphrase and try again.

Function-based Encryption (Application-level Encryption)

The SQL functions AES_ENCRYPT and AES_DECRYPT allow AES 128-bit encryption at the application (rather than the DBMS Server) level, by using encryption options on DML such as SELECT, INSERT, and UPDATE statements.

This example creates table socsec2 to store function-encrypted data and insert the data from table socsec1.

```
CREATE TABLE socsec2
       (fname char(10),
        lname char(20),
        socsec byte(16));

INSERT INTO socsec2 SELECT
        fname, lname,
        AES_ENCRYPT(socsec,'user function encryption')
        FROM socsec1;
```

The stored data is encrypted, even though Ingres does not regard table socsec2 as encrypted. There is no need to MODIFY socsec2 to enable access to encrypted data.

A SELECT on the table shows that the data is not plain text:

```
SELECT fname, lname, HEX(socsec) AS socsec FROM socsec2
Executing . . .

+----------+--------------------+--------------------------------+
|fname     |lname               |socsec                          |
+----------+--------------------+--------------------------------+
|John      |Smith               |5722A4EEDA081CB25955E826DDFA2A3F|
|Lois      |Lane                |814ACE20D419A8F36944625941155709|
|Charlie   |Brown               |40CD699016608827C7A9A4E7CDB161DF|
+----------+--------------------+--------------------------------+
(3 rows)
```

For AES_ENCRYPT, the data is stored as a variable length byte string that encrypts both the contents and length of the value being encrypted.

To be able to read the data, use the AES_DECRYPT function, supplying the secret passphrase:

```
SELECT fname, lname,
        AES_DECRYPT(socsec,'user function encryption') AS socsec
        FROM socsec2
Executing . . .

+----------+--------------------+----------------+
|fname     |lname               |socsec          |
+----------+--------------------+----------------+
|John      |Smith               |012-33-4567     |
|Lois      |Lane                |010-40-1234     |
|Charlie   |Brown               |012-44-9876     |
+----------+--------------------+----------------+
(3 rows)
```

With function-based encryption, the application decides what byte values are saved in the column. Data in a single column can be unencrypted, encrypted with different passphrases, doubly encrypted, and so on.

In this example we change the passphrase for just one row in the table.

```
UPDATE socsec2 SET socsec =
  AES_ENCRYPT(AES_DECRYPT(socsec,'user function encryption'),'Smith socsec')
  WHERE lname='Smith';

SELECT AES_DECRYPT(socsec,'Smith socsec') FROM socsec2 WHERE lname='Smith'
Executing . . .

+----------------+
|col1            |
+----------------+
|012-33-4567     |
+----------------+
(1 row)
```

If we then select all rows from the table, supplying the original passphrase, decryption of the row for 'Smith' fails, and a blank string is returned for that row.

```
SELECT AES_DECRYPT(socsec,'user function encryption') FROM socsec2
Executing . . .

+----------------+
|col1            |
+----------------+
|                |
|010-40-1234     |
|012-44-9876     |
+----------------+
(3 rows)
```

Function-based decryption with the wrong passphrase typically returns an empty string, but may return random data. If necessary, the application can add a scheme for sanity checking returned data. (Such sanity checking is built into *transparent encryption* through a hash validation value that is stored with the encrypted data.)

Encryption Information Displayed with HELP TABLE

The HELP TABLE statement displays encryption information for tables that contain encrypted columns.

For example, column encryption of the socsec1 table is defined at the DBMS level. Here is an excerpt from the output from **HELP TABLE socsec1**:

```
Column Information:
                                                        Key
Column Name            Type       Length Nulls Defaults Seq
fname                  char           10  yes    null
lname                  char           20  yes    null
socsec                 char           11  yes    null

Secondary indexes:    none

Column encryption:    AES256
Alter table totwidth: 48
Encrypted width:      48

Encrypted Column Name            Type        Width Salt
socsec                           char           16   no
```

Encryption-related fields are:

Column encryption

Encryption type

Alter table totwidth

Physical width of encrypted columns

Encrypted width

Physical width of the table

Encrypted Column Name...Width

Physical width of the named column

In this example, the logical width of the table (width as seen by applications) is 44, which is the sum of the column widths including the NULL bytes. Since AES is a block encryption algorithm, encrypted column widths as stored on disk will always be a multiple of 16. The NULL byte and a data verification hash value are included in the encrypted data. In this case, the socsec column with NULL (12 bytes) plus verifying hash (4 bytes) exactly fits in one AES block. Partial AES blocks are padded as needed in cases where the user data is not an exact fit.

In contrast, the socsec2 table is used to store encrypted data, but encryption is defined at the application level with the AES_ENCRYPT function. Here is an excerpt from the output of **HELP TABLE socsec2**:

```
Column Information:
                                                            Key
Column Name            Type           Length Nulls Defaults Seq
fname                  char               10  yes    null
lname                  char               20  yes    null
socsec                 byte               16  yes    null

Secondary indexes:    none
```

No encryption information is shown because there is nothing special about the table. The application has the responsibility to ensure that the encrypted data fits in the BYTE column that is used for that purpose. In this case, the sum of the VARBYTE length for the encrypted data (2) plus that data itself (11) fits within one AES block, which is stored in the BYTE(16) column.

How to Compute the Width of Encrypted Data

Encrypted data takes up more room than unencrypted data for the following reasons:

1. AES is a block cipher that operates only on 16-byte chunks, so padding is often necessary.
2. At-rest encrypted data includes a 4-byte hash to validate decryption processing.
3. Adding SALT to guarantee the unique encryption of each row of an encrypted column adds 16-bytes of overhead.

The HELP TABLE command displays the physical width of encrypted columns in an encryption section (see page 99) of the report.

Use the following algorithm to calculate the width of an encrypted column:

1. Start with the natural width of the column.
2. Add 1 for nullable columns.
3. Add 4 for the verification hash.
4. Round up to the nearest multiple of 16.

The goal of encryption is to make meaningful data appear to be a random series of bits until the encryption algorithm in combination with the encryption key is used to restore the data to its original state. One result is that encrypted data does not compress well, so Ingres does not compress encrypted columns.

Thus, the net effect on disk storage needs of encryption is a combination of the expansion of the encrypted rows for necessary overhead, and the loss of compressibility of the encrypted columns.

The AES_ENCRYPT function accepts as input a string of type VARBYTE and encrypts the entire string, including the 2-byte prefix that holds the VARBYTE length.

To compute the length of the encrypted output of the AES_ENCRYPT function, use the following algorithm:

1. Start with the length of the input as VARBYTE (that is, after casting to that data type if the original input is of another type).
2. Add 2 for the length prefix.
3. Round up to the nearest multiple of 16.

If you store AES_ENCRYPT encrypted data in a database table, be sure to allocate sufficient space for the full encrypted data length. Truncated encrypted data cannot be decrypted successfully.

Data at Rest Encryption Restrictions

Restrictions for encrypting columns are as follows:

- An encrypted column cannot be part of a table key.
- An encrypted column can be indexed, but the column must be defined with NOSALT, the index must be on the one column only (no composite indexes) and the index must be of type HASH.

  Note: Due to the nature of encrypted data, only exact lookups are possible using the index. Because range and pattern queries cannot use the index, they would typically require a full table scan; such queries may be prohibitively costly.
- Long data types (LOBs) cannot be encrypted.

Implications of Data Encryption for Database Design and Operations

When an encrypted table is created, an AES key is created (either generated randomly or according to the key bit pattern specified on AESKEY=). The AES key—not the passphrase—controls the encrypted binary representation of data, and is stored in the catalog at the table level.

The passphrase is used to derive another AES key (not stored), which secures the catalog-stored AES key. The passphrase is essentially a lockbox for the encrypted AES key.

The fact that there is one AES key that encrypts the user data and another passphrase-derived key that protects the first AES key has the following implications for database design and operations:

- When the passphrase is changed, database backups before this point in time must be accessed using the old passphrase. Subsequent backups must be accessed using the new passphrase. When changing the passphrase, it is **not** necessary to suspend use of the table by rebuilding it (though the encryption access will be momentarily suspended until the new passphrase is confirmed with a second MODIFY command).

  When the passphrase is changed, the catalog-stored key is decrypted with the old passphrase, re-encrypted with the new passphrase, and then replaced in the catalog. The user data is not re-encrypted because only the passphrase—not the underlying user key—has changed.

- If a backup or replication scheme works at the binary level by transferring and applying journal records, the original table and the copy table must use the same catalog-stored AES key. This will be the case if (1) the catalogs were copied at the binary level or (2) the AESKEY= option was used when the table was created.

  Note: Tables that share an underlying AES key for encrypting user data can be protected by different passphrases.

- Encrypted data is usually unique at the binary level for each stored instance or at least in each stored table. It is therefore problematic to protect it outside of the table in the same manner that it is protected in the Ingres table files. When data is copied to a flat file, it is unprotected. Such files should use a different protection scheme, such as password encrypting the entire file, or writing it to an encrypted device or file system.

Understanding Salt

The NOSALT column-level option overrides the default (SALT), which adds 16 bytes of random bits to data before encryption and strips them out after decryption.

To understand salt, consider this example. A medical database contains a column indicating the status for each patient: whether they have had a particular medical test and, if so, whether they tested positive or negative for the condition. A value of Y indicates a positive result, N indicates a negative result, and NULL that the test has not been done. This data must be keep confidential. If the column is encrypted, has the patient information been protected?

If SALT is used the data is indeed protected. But if NOSALT is specified, the data is essentially unprotected. Given identical input and an identical key, encryption is repeatable. If Y is encrypted again and again with the same AES key, the encrypted representation of each value is the same.

Someone working to "crack" the database can correlate the repeated patterns with the easily learned frequency of positive and negative test results in the general population. The secret is revealed without needing to crack the actual AES encryption. The problem is that hiding the values Y and N does not hide the **information** that Y and N represent.

If salt is added to the encryption, every row contains, in the indicator column, a different encrypted value. Because the null indicator is included in the encryption protection, it is impossible to know who has been tested or who has tested positive for the condition. In fact, no statistical information can be gleaned. Without the passphrase, which protects the AES key for the table, the medical test result column appears to contain nothing but meaningless binary noise, which is the goal of encryption.

But consider another column in the medical database that contains values that do not repeat: the U.S. Social Security number of the patient. The encrypted values in the column will be distinct, regardless of whether SALT or NOSALT is specified. (Furthermore, the values are distinct from the same Social Security number stored encrypted in another table, because the other table has a different AES key, even if the tables are protected with the same passphrase.)

In the case of Social Security numbers, NOSALT still yields apparent randomness in the encrypted values. The only information leaked is the fact that Social Security numbers are different for each patient, which is common knowledge.

Using SALT (the default):

- Provides further protection of the encrypted data.
- Hides column value repetitions (for example, Y, N, null).

- Provides a unique encrypted representation of each value (because a different, random, salt prefix is generated for each encryption operation). SALT renders the encryption non-repeatable.

NOSALT can be used:

- When you want to index a table on the encrypted value of the column
- When the column has values that are unique before encryption
- To use less disk storage. Salt adds one AES block (16 bytes) to the column.

  Note: The cost is a small price to pay for better protection, considering the relatively low cost of storage.

Indexing Encrypted Columns

Because encryption randomizes data in a way that destroys alphabetical order, an index built on an encrypted column is limited to “exact hit” lookups. The index is built with a tuple ID pointer (tidp) that records the row in the base table that contains the encrypted value.

The following restrictions apply to indexes on encrypted columns:

- Indexed encrypted columns must not use SALT.
- Indexes on tables that contain encrypted columns must be defined with HASH storage structure.
- Because such an index only supports exact indexed value lookup, the cost of doing a range lookup or string pattern match on encrypted columns is **much** higher than a similar operation on an unencrypted column. In general, such queries should be avoided for performance reasons.

Encryption and Copydb/Unloaddb Considerations

Data at rest encryption does not protect data outside the database, including files created using the copydb and unloaddb commands.

The copydb command creates a copy.out file that writes the decryption-enabled table to a flat file as plaintext. It is important that the resulting file be encrypted with file or disk encryption facilities.

The corresponding copy.in file contains encryption syntax to recreate the exported table with the original column encryption. The standard passphrase 'TEMPORARY PASSPHRASE' is encoded in the copy.in file. A good practice is to use this or other temporary phrase to import the table, and then to MODIFY the passphrase immediately after import.

Optimizedb Considerations for Data at Rest Encryption

When the optimizedb utility is used to create statistics for encrypted columns, the histogram cells will contain unencrypted (plain text) data. The histogram cells are statistical extracts, and cannot be linked back to any particular row in the table (except when there is only a single table row). Nevertheless, depending on the nature of the data and your business, this may be considered a security breach.

By default, the optimizedb program skips encrypted columns, but they can be included by specifying the -ze flag. Alternatively, the -r (relation) and -a (attribute) flags can be used to specify exactly which tables and columns to include.

You can verify which columns have had statistics generated for them with the statdump command.

Note: For Ingres Star databases, optimizedb is not able to determine whether or not a column is encrypted. To exclude encrypted columns, use the -r and -a flags to specify the columns for which statistical data is gathered.

Encrypted Data in Log Records and Auditdb Output

UPDATE statements are recorded in the log file differently for encrypted tables because:

- Encrypted changes are physically wider than the values of the decrypted column values
- When SALT is used, each re-encryption of the data yields a different physically stored value

Such characteristics of encrypted data in log and journal files can be seen in the auditdb output, which displays encrypted records in hex format and without decryption.

Consider this script, in which a table with two integer columns (only the second of which is encrypted) is created, populated, and then updated:

```
CREATE TABLE auditdemo (c1 INT, c2 INT ENCRYPT)
WITH ENCRYPTION=AES256, PASSPHRASE='auditdb demo';
MODIFY auditdemo ENCRYPT WITH PASSPHRASE='auditdb demo';
INSERT INTO auditdemo VALUES (1,2);
UPDATE auditdemo SET c1=3 WHERE c1=1;
UPDATE auditdemo SET c1=4 WHERE c1=3;
```

Even in the hex display we can see that c1 starts off as value 1 and then is updated to 3 and then to 4. But notice also that the remainder of these rows change too, because each rewrite of the row causes a re-encryption that produces a different value.

```
Audit for database aud
18-Aug-2010 12:50:03.59                   Page    1

Begin   : Transaction Id 00004c584d495a7e 18-Aug-2010 12:49:56.89
        Username ingres
  Create  : Transaction Id 00004c584d495a7e Id (268,0) Table [auditdemo,ingres   ]
        Location $default
  Insert/Append  : Transaction Id 00004c584d495a7e Id (268,0) Table [auditdemo,ingres]
    Record: 01000000006d8f35d1cff37f6e7eab7070403c13b785bc20e6e489f41731daa0230db78884
  Update/Replace : Transaction Id 00004c584d495a7e Id (268,0) Table [auditdemo,ingres]
    Old:    01000000006d8f35d1cff37f6e7eab7070403c13b785bc20e6e489f41731daa0230db78884
    New:    03000000006e954a05f5c9f8a5b73234ec957386c23bf686d8141cb180d190a4b8c7bc17f6
  Update/Replace : Transaction Id 00004c584d495a7e Id (268,0) Table [auditdemo,ingres]
    Old:    03000000006e954a05f5c9f8a5b73234ec957386c23bf686d8141cb180d190a4b8c7bc17f6
    New:    0400000000829e143ee15459d75aa0f8d3238a2b1ea1643a594b6dffb4792c5b59f8657bb5
End     : Transaction Id 00004c584d495a7e 18-Aug-2010 12:49:56.90
```

The SQL statements in the script do not update column c2. So, despite the various encrypted incarnations of the record, the underlying value for column c2 is the one that was specified when the row was first inserted:

```
SELECT * FROM auditdemo
Executing . . .

+-------------+-------------+
|c1           |c2           |
+-------------+-------------+
|            4|            2|
+-------------+-------------+
(1 row)
```

Encryption and Tables that Hold Audit Trails

Encrypted columns are in their encrypted format in the journals and thus in the audit trails created with auditdb. This means that the encrypted byte format must be specified for the table into which an audit trail (.trl file) is copied.

The table that holds the audit data must match the .trl file that is processed with the COPY statement, so for encrypted columns, use the length as displayed in the encrypted section of HELP TABLE.

Consider the auditdb example (in the *Command Reference Guide*) in which a table to hold audit data is created for the emp table. We decide to encrypt the salary column in the emp table:

```
CREATE TABLE emp
(eno I2,
ename CHAR(10),
age I1,
job I2,
salary MONEY ENCRYPT,
dept I2)
WITH ENCRYPTION=AES192, PASSPHRASE='encrypted audit example';
```

The HELP TABLE output for emp shows that the salary column will be 32 bytes long in the audit record:

```
Column encryption:    AES192
Alter table totwidth: 54
Encrypted width:      54


Encrypted Column Name            Type        Width Salt

salary                           money          32  yes
```

So when we create an empaudit table to hold the audit trail for emp, we need to define the salary column as a non-nullable byte(32):

```
CREATE TABLE empaudit
(date DATE NOT NULL WITH DEFAULT,
usrname CHAR(32) NOT NULL WITH DEFAULT,
operation CHAR(8) NOT NULL WITH DEFAULT,
tranid1 INTEGER NOT NULL WITH DEFAULT,
tranid2 INTEGER NOT NULL WITH DEFAULT,
table_id1 INTEGER NOT NULL WITH DEFAULT,
table_id2 INTEGER NOT NULL WITH DEFAULT,
eno I2,
ename CHAR(10),
age I1,
job I2,
salary BYTE(32) NOT NULL,
dept I2);
```

Although the terminal output format for auditdb for encrypted tables is in hex and therefore hard for us to read, the imported audit trail is readable. The encrypted column is in AES encrypted format and thus readable only at this level. The plaintext value is protected.

```
SELECT date, operation, eno, hex(salary) AS salary FROM empaudit
Executing . . .

--------------------+------+---+----------------------------------------------------------------+
date                |operat|eno|salary                                                          |
--------------------+------+---+----------------------------------------------------------------+
01-oct-2010 18:46:05|append|  1|CC438DD9ECEE3E406D52C4F7FDCA85FBD5BD11374E77E6819E024818E0C646B0|
01-oct-2010 18:46:05|append|  2|C3A5D3012B3EAAD877F2E2AFD02725225FE987370C7A12A97BAE32F6C7AF0B6B|
--------------------+------+---+----------------------------------------------------------------+
```

Encryption and Partitioned Tables

Partitioning on an encrypted column should be avoided because of a security issue: the partition into which each row is assigned will reveal how the encrypted column's plaintext value tested against the partitioning rules.

Although such partitioning is allowed, we recommend avoiding it unless the security concerns are well understood and accepted and the application design requires it.

Note: This warning applies only to partitioning on the encrypted columns themselves.

Encrypted columns can be included in partitioned tables. An index can be created on an encrypted column in a partitioned table.

Appendix A: Configuring Ingres to Use Kerberos

This section contains the following topics:

Kerberos

The Kerberos authentication mechanism can be used as an alternative to the Ingres or System security mechanisms. Kerberos is a network authentication and encryption protocol that provides a highly secure alternative to operating system-level password authentication, and optionally allows encryption of the entire data stream exchanged between the DBMS server and client.

The Ingres and System security mechanisms are called “static” mechanisms, because they are embedded in Ingres. The Kerberos security mechanism is called a “dynamic” mechanism, because it depends on third-party software that is dynamically loaded into Ingres executable images at runtime.

Kerberos is available as freeware from the Massachusetts Institute of Technology at http://web.mit.edu/kerberos/. Kerberos is also available commercially or may be available natively on certain operating systems, such as Linux. The MIT site contains extensive documentation on Kerberos installation and configuration.

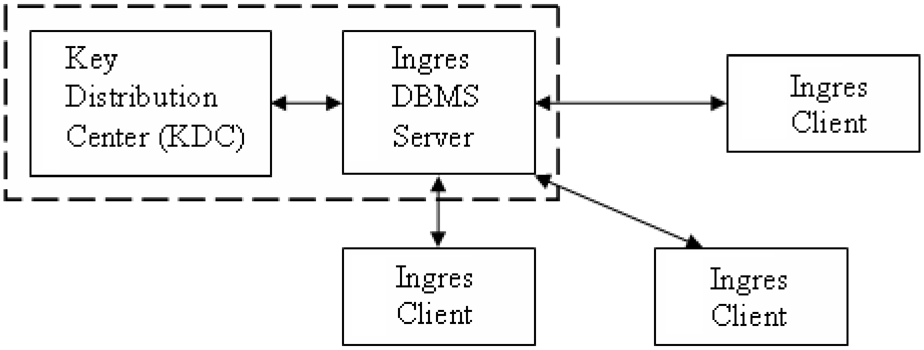
The Ingres Kerberos driver references authentication and encryption routines in the Kerberos environment, most notably, the shared library or DLL containing GSS API authentication routines.

Kerberos Configuration in the Enterprise

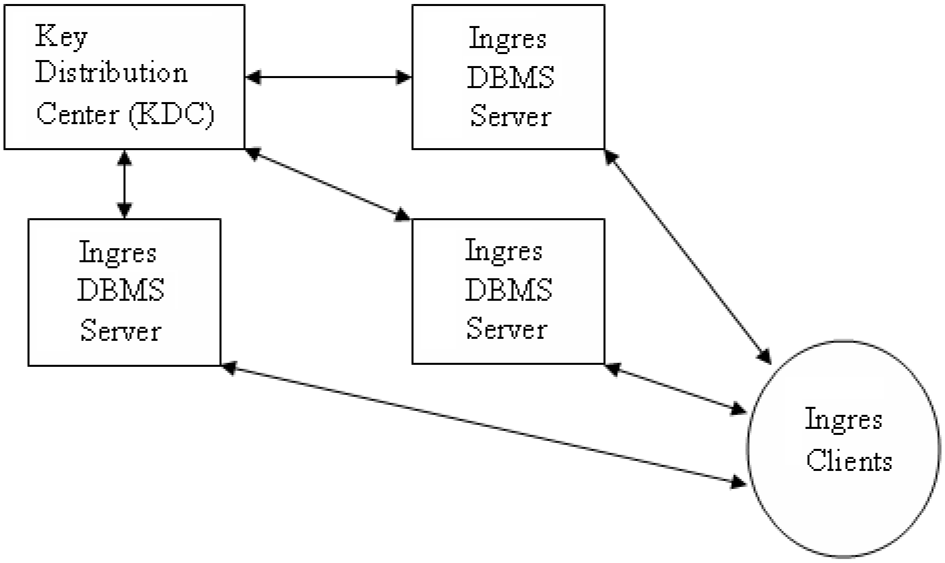
Before using Kerberos with Ingres, Kerberos should be appropriately configured in your enterprise.

A primary component of Kerberos is the Key Distribution Center (KDC). The KDC is a server process that performs the core authentication. The authentication protocol is a set of encrypted tickets that are passed from the KDC to client processes or intermediate agents known as “service principals.” For the sake of simplicity, let us assume that a single KDC will perform the Kerberos authentication.

If the enterprise contains only one Ingres DBMS Server, a possible option is to execute the KDC on the same machine as the DBMS Server:



If enough resources are available, it is desirable to install the KDC on a network node separate from the Ingres installation. In this way, security restrictions can be imposed on the Kerberos node that may not be possible if Kerberos resided on the same machine as an Ingres DBMS:



The example above demonstrates why Kerberos is sometimes referred to as "distributed authentication." The KDC performs authentication for all Ingres nodes in the enterprise, even though the KDC itself resides on a separate network node.

Note: The above example assumes all the Ingres nodes will use Kerberos for authentication, but this is not a requirement; some nodes may continue to use Ingres or System authentication.

Kerberos Configuration Files—Configure Kerberos for Ingres

Here are examples of Kerberos configuration files. These examples assume that the KDC resides on the node foo.xyz.com and the Kerberos domain is named MYDOMAIN.XYZ.COM,

The krb5.conf file may look like this:

```
[libdefaults]
    default_realm = MYDOMAIN.XYZ.COM

[realms]
    SSF.XYZ.COM = {
        kdc = foo.xyz.com
        admin_server = foo.xyz.com
    }

[domain_realm]
        .xyz.com = MYDOMAIN.XYZ.COM
        xyz.com = MYDOMAIN.XYZ.COM

[logging]
    kdc = FILE:/var/log/krb5kdc.log
    admin_server = FILE:/var/log/kadmin.log
    default = FILE:/var/log/krb5lib.log
```

The kdc.conf file may look like this:

```
[kdcdefaults]
    kdc_ports = 88

[realms]
    MYDOMAIN.XYZ.COM = {
        kadmind_port = 749
        max_life = 12h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        master_key_type = des3-hmac-sha1
        supported_enctypes = des3-hmac-sha1:normal des-cbc-crc:normal des-cbc-
crc:v4
    }
```

The Ingres Service Principal—Authorize Client Connections

A Kerberos principal is an entity to which credentials (validated tickets) may be assigned. Most principals of concern to Ingres are simply those that correspond to the login names of the Ingres users. For instance, for the domain MYDOMAIN.XYZ.COM, a principal representing the "ingres" user is "ingres@MYDOMAIN.XYZ.COM".

Note: The credentials associated with the "ingres" user are valid for all "ingres" logins in the Kerberos domain, regardless of the system passwords associated with the "ingres" login name on each machine.

A KDC must define user principals for each Ingres user that exists in the enterprise, and for each Ingres service principal. An Ingres service principal does not correspond to a login user name. Instead, the Ingres service principal represents an Ingres process that performs authentication on behalf of the user.

User principals get tickets directly from the KDC through the kinit or Leash Ticket Manager or Network Identify Manager programs, but an Ingres service principal requires no such initialization. Instead, the Ingres service principal relies on the Kerberos keytab file to establish its credentials.

An Ingres service principal definition is required for each node on the Kerberos domain that has an Ingres installation. The KDC installation must define a keytab file for all Ingres service principals in order to decrypt tickets received from the KDC. A copy of the keytab file must be installed on each Ingres node in the Kerberos domain. For the best security, set ownership of the keytab file to "ingres" or the installation owner, and set read-only permissions to the keytab file.

We strongly recommend that you define the KR5_KTNAME environment variable as the full path and file name of the keytab file. On Windows, this is mandatory.

The Ingres service principal uses the standard Kerberos "primary/instance@realm" format, as follows:

```
$ingres/hostname@realm
```

hostname

Is the fully-qualified domain name of the host on which the Ingres installation is running. To find the fully-qualified host name for your machine, execute the iinethost utility.

realm

Is the Kerberos administrative domain name.

In the example host name foo.xyz.com, the Ingres service principal would be named “$ingres/foo.xyz.com@MYDOMAIN.COM”.

Note: The fully-qualified host name is required when defining the Ingres service principal. Thus, the name “$ingres.foo@MYDOMAIN.COM” is not a valid Ingres service principal name. The “$ingres/” prefix is mandatory. A principal name such as “ingres.foo@MYDOMAIN.COM” is invalid due to the missing dollar sign ($).

How to Configure Ingres to Use Kerberos

The process for configuring Ingres to use Kerberos is as follows:

1. Set the basic configuration for using Kerberos by doing **either** of the following:

   - Run the iisukerberos utility (see page 116).
   - Set parameters in Configuration-By-Forms, as described in Basic Configuration for Kerberos (see page 117).

2. Set other parameters in Configuration-By-Forms, as needed, according to your environment.

3. Obtain authorization tickets by using the kinit command (Windows, VMS and UNIX), the Leash Utility (Windows), or the Network Identity Manager (Windows).

4. Stop and restart Ingres.

   Startup will be successful if the Kerberos GSS API library exists in your LD_LIBRARY_PATH definition (UNIX and Linux), if the GSSAPI32.DLL file resides in your system environment path (Windows), or if the file SYS$LIBRARY:GSS$RTL32.EXE is installed (VMS).

5. Test your server using a loopback test.

   To test a loopback connection using Kerberos, the local Name Server must be configured for Kerberos authentication by using the iisukerberos utility or by setting the "remote_mechanism" setting in the Name Server to "kerberos" in the Configuration-By-Forms utility. In addition, your loopback vnode entry, as defined in netutil, must have an attribute named "authentication_mechanism" and an attribute value of "kerberos", as described in vnode Connection Attributes (see page 119).

   If you do not want to define a loopback vnode, proceed to step 7.

6. Test your connection using the Terminal Monitor, as follows:

   ```
   sql loopback::iidbdb
   ```

   The loopback vnode should be as described in the preceding step.

7. Set up your clients. Your netutil definitions are almost the same as when using os-level authentication, but you should leave the login/password data blank.

iisukerberos Command—Perform Basic Kerberos Configuration

The iisukerberos utility provides a simple interface for the most commonly-used Kerberos configuration options.

This command has the following format:

UNIX, Linux, Windows:

```
iisukerberos
```

VMS:

```
@II_SYSTEM:[ingres.utility]iisukerberos.com
```

The iisukerberos utility configures Kerberos at the following levels:

- Client configuration

  Client configuration enables this installation to use Kerberos to authenticate against a remote installation that has been configured to use Kerberos for authentication. This is the minimum level of Kerberos authentication.

  Note: You must add the "authentication_mechanism" attribute in netutil for each remote node you wish to authenticate using Kerberos. The "authentication_mechanism" attribute must be set to "kerberos". There is no need to define users or passwords for vnodes using Kerberos authentication.

- Name Server authentication

  Name Server authentication allows the local Name Server to authenticate using Kerberos.

- Server-level authentication

  Server-level authentication forces all local servers, such as the DBMS Server and the Communications Server, and applications to authenticate against the Name Server using Kerberos.

You can select any combination of client, Name Server, and server-level authentications.

Ingres Configuration Options for Kerberos

To configure Ingres to use Kerberos, you must set certain parameters in Configuration-By-Forms. In addition, connection attributes may be required depending on the requirements of the enterprise.

The following system components in Configuration-By-Forms are relevant:

- Name Server
- Net Server
- Security, Configure, System
- Security, Configure, System, Mechanisms

Basic Configuration for Kerberos

When you configure Ingres to use Kerberos, you should first check the basic configuration. The basic configuration consists of the mechanisms parameter and the domain parameter in the Security component.

mechanisms Parameter—Specify Dynamic Mechanism

For Ingres to use Kerberos as a dynamic mechanism, the mechanism parameter must be set to kerberos. In Configuration-By-Forms, the mechanisms parameter is located in Security, Configure, System.

The setting should look similar to this:

Name	Value	Units
mechanisms	kerberos	mechanism list

Note: The ingres, system, or null mechanisms are invalid entries to this list, since its purpose is to specify the dynamic authentication mechanisms.

domain Parameter—Specify Domain Name

In addition to the mechanism parameter, the domain parameter must be set to configure Ingres to use Kerberos.

In Configuration-By-Forms, the domain parameter is located in Security, Configure, System, Mechanisms, Kerberos.

The domain parameter must contain the fully qualified host name of the local installation. This name corresponds to the Ingres service principal name. For example, for machine foo.xyz.com, the value for the domain parameter should be "foo.xyz.com." If the entry reads simply "foo," edit and correct the entry.

The setting should look similar to this:

Name	**Value**	**Units**
domain	foo.xyz.com	hostname

remote_mechanism Parameter—Configure Client in a Homogeneous Kerberos Environment

The Name Server can be configured to use Kerberos for authentication for all remote targets. If so configured, connection attempts on non-Kerberos targets will fail. Use the remote_mechanism parameter for this purpose.

In Configuration-By-Forms, the remote_mechanism parameter is located in the Name Server component. Add kerberos to the mechanism list (if not already added in the Security configuration), and specify kerberos as the value on the remote_mechanism parameter.

The setting should look similar to this:

Name	**Value**	**Units**
remote_mechanism	kerberos	none, default, mechansim name

In a homogeneous Kerberos environment, it is not necessary to add login/password information for the vnode definitions in netutil. They are ignored at connect time.

vnode Connection Attributes—Configure Client in a Heterogeneous Kerberos Environment

Heterogeneous Kerberos environments are those in which both Kerberos and non-Kerberos connection targets exist in the enterprise. In such an environment, the Name Server settings in Configuration-By-Forms must remain at their default values. The local client behavior must change, depending on the connection target.

To configure the client in a heterogeneous Kerberos environment, specify connection attributes for a vnode using the netutil utility.

Here is a sample vnode configuration in netutil:

Connection data for vnode 'newyork'

Type	Net Address	Protocol	Listen Address
Global	newyork-xp1.	tcp_ip	TS

Other attribute data for vnode 'newyork'

Type	Attr_Name	Attr_Value
Private	authentication_mechanism	kerberos

Note: The login/password entry for a Kerberos target should remain blank. A login/password entry is not required because the local Kerberos user principal is used for authentication, and the KDC authenticates using the ticket cache of the local user, rather than the system password on the remote connection target.

Note: Kerberos authentication requires a TCP/IP-compatible network protocol on the local installation. On Windows, tcp_ip or win_tcp are acceptable protocol settings. On VMS, dec_tcp is the TCP/IP specifier, and will work if TCP/IP is supported through Multinet, TCP/IP, or TCP/IP over DECnet. Non-TCP/IP protocols, such as DECnet, are not supported.

Encryption Parameters—Enable Kerberos Encryption

To specify encryption, the following options are available in Configuration-By-Forms under the Net Server (also known as Communications Server) component:

ib_encrypt_mech

Determine the encryption mechanism for inbound connections. Valid values are

kerberos

Specifies that Kerberos be used.

*****

Specifies that Kerberos will be used if included as an item on the mechanism list.

ob_encrypt_mech

Determine the encryption mechanism for outbound connections. Valid values are the same as for ib_encrypt_mech.

ib_encrypt_mode

Determines the encryption mode for the inbound data stream. Valid values are as follows:

Off

Specifies that encryption be neither requested nor allowed.

Optional

Specifies that encryption may occur but is not requested.

On

Specifies that encryption is requested, if possible (if both ends support it).

Required

Specifies that encryption must always occur.

ob_encrypt_mode

Determines the encryption mode for the outbound data stream. Valid values are the same as for ib_encrypt_mode.

Outbound connection items may be configured as connection attributes in netutil.

The following example specifies Kerberos encryption for all inbound connections:

Name	Value	Units
ib_incrypt_mech	kerberos	*, mechanism name
ib_incrypt_mode	required	off, optional, on, required

How Name Server Delegation Works

Delegation provides an alternate method of acquiring and forwarding authentication. When delegation is configured, the Name Server generates authentication certificates as if it were the client.

This method requires Kerberos to be configured as both the local and remote authentication mechanism. The client process generates an authentication certificate for the local Name Server. The local Name Server, in turn, uses its delegation capabilities to generate an authentication certificate, and forwards the certificate on behalf of the client to the remote Name Server.

If delegation is not enabled, or Kerberos is not configured as the local authentication mechanism, then the Name Server cannot generate the remote authentication certificate. Instead, the client acquires the authentication certificate prior to making the remote connection. The client then forwards the credentials directly to the remote Name Server. Either method is valid for making secure connections through Kerberos.

Set Delegation

The process of acquiring and forwarding authentication can be delegated to the Name Server.

To set delegation

1. Start Configuration-By-Forms and select Security, Configure, Mechanisms, Kerberos.
2. Set the delegation parameter to on.

Service Principal Host Name Resolution

The KDC will not resolve the fully qualified host name (FQDN) correctly (even though you specify it on the Configuration-By-Forms domain parameter (see page 118)) unless it resolves the host name passed from the client as the FQDN.

The FQDN is picked up from your network configuration (rather than the Ingres config.dat setting) when the Kerberos driver calls gss.init.sec..context(). Often the unqualified host name is passed to the KDC, and gss.init.sec..context() fails.

To ensure that the KDC can resolve the fully qualified host name

UNIX and Linux:

In environments other than Windows, edit the local host file with the FQDN and not the alias for your local host as the first entry. On UNIX and Linux, the file is /etc/hosts and often looks like this:

```
# Syntax:
#
# IP-Address  Full-Qualified-Hostname  Short-Hostname
#

127.0.0.1       localhost
nn.nn.nn.nn     myhost.mydomain.com    myhost
```

VMS: On VMS, use the TCPIP utility:

```
$ tcpip

TCPIP> show hosts

      LOCAL database

Host address    Host name

127.0.0.1       localhost
nn.nn.nn.nn     myhost.mydomain.com   myhost
```

For details on configuration of the local hosts file, see your system administrator.

VMS Considerations

Kerberos commands are accepted in uppercase by default (as in DCL commands). Accordingly, principal names should be enclosed in double quotes, as in the following example:

```
$ kinit "tingresxx"
```

If "tingresxx" had not been enclosed in double quotes, the KDC would have looked for the principal "TINGRESXX" instead of "tingresxx". Creation of the Kerberos principal "TINGRESXX" (all caps) will not help; the Ingres security mechanism fails the authentication due to "TINGRESXX" not equating to "tingresxx". The Ingres security mechanism is still in force as the default authentication mechanism regardless of whether Kerberos is configured.

If server-to-server authentication is desired (CBF, Security, Configure, user_mechanism), the VMS logical KRB$USER must be defined at the group level. The SYS$MANAGER:KRB$SYMBOLS.COM script defines only at the process level by default. Otherwise, security contexts will not be accepted and the error "GSS-API error gss_init_sec_context: Credentials cache I/O operation failed XXX" will appear in a GCS trace or the error log. This problem occurs because servers run as detached processes. Detached processes run without a CLI and have no visibility of logical definitions at the process level.

The following command file defines KRB$USER at the group level based on the process-level definition, runs ingstart, and de-assigns KRB$USER:

```
$!
$! RINGSTART.COM - Run Ingstart with KRB$USER
$!
$ krb = F$TRNLNM("KRB$USER","LNM$PROCESS")
$ define/group/executive/translation=concealed KRB$USER 'krb'
$ ingstart
$ deassign/group/exec KRB$USER
$ exit
```

Glossary

authorization identifier

An *authorization identifier* is an entity to which access to database objects can be granted. The four authorization identifiers are: Role, User, Group, or PUBLIC.

data at rest

Data "at rest" is data on physical media recorded in a persistent form.

database procedure

A *database procedure* is a set of SQL statements and control statements in a begin/end block that are stored as a unit in the database.

default group

A *default group* is the group assigned to a user when a user object is created or modified.

default profile

A *default profile* is the profile initially assigned to a user if one is not explicitly assigned.

grant

A *grant* is an object permission assigned to a group, role, or user.

group

A *group* is an identifier that can be used to apply permissions to a list of users associated with the identifier.

object permission

An *object permission* defines a capability related to a specific object, such as a database or a table. Object permissions are assigned to selected groups, roles, or users. Object permissions are also called grants, permits, or object privileges.

privileged user

A *privileged user* is any user with the necessary privileges to perform security-related operations. The system administrator, database administrator, or security administrator are privileged users.

profile

A *profile* is a template that defines a set of subject privileges and other attributes that can be applied to one or more users. The user authorization process can be streamlined by using profiles.

role

A *role* is an identifier that can be used to associate permissions with applications.

security alarm

A *security alarm* is an entity that can be created to monitor a security-related event, such as connecting to a database or updating a table.

security auditing

Security auditing is the recording of all or specified classes of security events for the entire Ingres installation.

subject privilege

A *subject privilege* defines the type of operations permissible in a user session. Subject privileges are assigned to a user (subject).

user object

A *user object* is a definition that specifies the user’s name, default group, default profile, subject privileges, and several other attributes.

Index