Greenplum® Database

Security Configuration Guide

Rev: A01

Greenplum Database Security Configuration Guide 4.2 - Contents

Preface

This guide provides information for system administrators and database superusers responsible for configuring and maintaining the security for a Greenplum Database system.

About This Guide

This guide assumes knowledge of Linux/UNIX system administration, database management systems, database administration, and structured query language (SQL).

Because Greenplum Database is based on PostgreSQL, this guide assumes some familiarity with PostgreSQL. Links and cross-references to PostgreSQL documentation are provided throughout this guide for features that are similar to those in Greenplum Database.

About Pivotal, Inc.

Greenplum is currently transitioning to a new corporate identity (Pivotal, Inc.). We estimate that this transition will be completed in 2013. During this transition, there will be some legacy instances of our former corporate identity (Greenplum) appearing in our products and documentation. If you have any questions or concerns, please do not hesitate to contact us through our web site: `http://gopivotal.com/about-pivotal/support`.

Document Conventions

The following conventions are used throughout the Greenplum Database documentation to help you identify certain types of information.

Text Conventions

Table 0.1 Text Conventions

Text Convention	Usage	Examples
bold	Button, menu, tab, page, and field names in GUI applications	Click **Cancel** to exit the page without saving your changes.
italics	New terms where they are defined Database objects, such as schema, table, or columns names	The *master instance* is the `postgres` process that accepts client connections. Catalog information for Greenplum Database resides in the *pg_catalog* schema.
`monospace`	File names and path names Programs and executables Command names and syntax Parameter names	Edit the `postgresql.conf` file. Use `gpstart` to start Greenplum Database.
`monospace italics`	Variable information within file paths and file names Variable information within command syntax	`/home/gpadmin/`*`config_file`* `COPY` *`tablename`* `FROM '`*`filename`*`'`
`monospace bold`	Used to call attention to a particular part of a command, parameter, or code snippet.	Change the host name, port, and database name in the JDBC connection URL: `jdbc:postgresql://`**`host:5432/mydb`**
`UPPERCASE`	Environment variables SQL commands Keyboard keys	Make sure that the Java `/bin` directory is in your `$PATH`. `SELECT * FROM` *`my_table`*`;` Press `CTRL+C` to escape.

Command Syntax Conventions

Table 0.2 Command Syntax Conventions

Text Convention	Usage	Examples
`{ }`	Within command syntax, curly braces group related command options. Do not type the curly braces.	`FROM { 'filename' | STDIN }`
`[ ]`	Within command syntax, square brackets denote optional arguments. Do not type the brackets.	`TRUNCATE [ TABLE ] name`
`...`	Within command syntax, an ellipsis denotes repetition of a command, variable, or option. Do not type the ellipsis.	`DROP TABLE name [, ...]`
`|`	Within command syntax, the pipe symbol denotes an "OR" relationship. Do not type the pipe symbol.	`VACUUM [ FULL | FREEZE ]`

`$ system_command`
`# root_system_command`
`=> gpdb_command`
`=# su_gpdb_command`
	Denotes a command prompt - do not type the prompt symbol. `$` and `#` denote terminal command prompts. `=>` and `=#` denote Greenplum Database interactive program command prompts (`psql` or `gpssh`, for example).	`$ createdb mydatabase`
		`# chown gpadmin -R /datadir`
		`=> SELECT * FROM mytable;`
		`=# SELECT * FROM pg_database;`

Getting Support

Pivotal/Greenplum support, product, and licensing information can be obtained as follows.

Product information and Technical Support

For technical support, documentation, release notes, software updates, or for information about Pivotal products, licensing, and services, go to www.gopivotal.com.

Additionally you can still obtain product and support information from the EMC Support Site at: http://support.emc.com.

1. Greenplum Security Overview

The essential security requirements for the Greenplum Database server falls into the following four broad categories:

- **Authentication**, which covers the mechanisms that are supported and that can be used by the Greenplum database server to establish the identity of a client application. See Chapter 2, "Configuring Client Authentication" for details.
- **Authorization**, which pertains to the privilege and permission models used by the database to authorize client access. See Chapter 5, "Managing Roles and Privileges" for more details.
- **Auditing** or log settings, covers the logging options available in the Greenplum database to track successful or failed user actions. See Chapter 3, "Auditing" for more details.
- **Data Encryption** addresses the encryption capabilities that are available for protecting data both at rest and in transit. Chapter 4, "Data Encryption" for more details.

  The last category describes the security certifications that are relevant to the Greenplum Database.

2. Configuring Client Authentication

When a Greenplum Database system is first initialized, the system contains one predefined *superuser* role. This role will have the same name as the operating system user who initialized the Greenplum Database system. This role is referred to as `gpadmin`. By default, the system is configured to only allow local connections to the database from the `gpadmin` role. If you want to allow any other roles to connect, or if you want to allow connections from remote hosts, you have to configure Greenplum Database to allow such connections. This chapter explains how to configure client connections and authentication to Greenplum Database.

Allowing Connections to Greenplum Database

Client access and authentication is controlled by a configuration file named `pg_hba.conf` (the standard PostgreSQL host-based authentication file). For detailed information about this file, see The pg_hba.conf File in the PostgreSQL documentation.

In Greenplum Database, the `pg_hba.conf` file of the master instance controls client access and authentication to your Greenplum system. The segments also have `pg_hba.conf` files, but these are already correctly configured to only allow client connections from the master host. The segments never accept outside client connections, so there is no need to alter the `pg_hba.conf` file on your segments.

The general format of the `pg_hba.conf` file is a set of records, one per line. Blank lines are ignored, as is any text after the `#` comment character. A record is made up of a number of fields which are separated by spaces and/or tabs. Fields can contain white space if the field value is quoted. Records cannot be continued across lines. Each remote client access record is in the format of:

```
host    database    role    CIDR-address    authentication-method
```

Each UNIX-domain socket access record is in the format of:

```
local    database    role    authentication-method
```

The meaning of the fields is as follows:

Table 2.1 pg_hba.conf Fields

Field	Description
local	Matches connection attempts using UNIX-domain sockets. Without a record of this type, UNIX-domain socket connections are disallowed.
host	Matches connection attempts made using TCP/IP. Remote TCP/IP connections will not be possible unless the server is started with an appropriate value for the `listen_addresses` server configuration parameter.
hostssl	Matches connection attempts made using TCP/IP, but only when the connection is made with SSL encryption. SSL must be enabled at server start time by setting the `ssl` configuration parameter. Requires SSL authentication be configured in `postgresql.conf`. See Configuring postgresql.conf for SSL Authentication.
hostnossl	Matches connection attempts made over TCP/IP that do not use SSL. Requires SSL authentication be configured in `postgresql.conf`. See Configuring postgresql.conf for SSL Authentication
database	Specifies which database names this record matches. The value `all` specifies that it matches all databases. Multiple database names can be supplied by separating them with commas. A separate file containing database names can be specified by preceding the file name with @.
role	Specifies which database role names this record matches. The value `all` specifies that it matches all roles. If the specified role is a group and you want all members of that group to be included, precede the role name with a +. Multiple role names can be supplied by separating them with commas. A separate file containing role names can be specified by preceding the file name with @.
CIDR-address	Specifies the client machine IP address range that this record matches. It contains an IP address in standard dotted decimal notation and a CIDR mask length. IP addresses can only be specified numerically, not as domain or host names. The mask length indicates the number of high-order bits of the client IP address that must match. Bits to the right of this must be zero in the given IP address. There must not be any white space between the IP address, the /, and the CIDR mask length. Typical examples of a CIDR-address are 172.20.143.89/32 for a single host, or 172.20.143.0/24 for a small network, or 10.6.0.0/16 for a larger one. To specify a single host, use a CIDR mask of 32 for IPv4 or 128 for IPv6. In a network address, do not omit trailing zeroes.
IP-address IP-mask	These fields can be used as an alternative to the CIDR-address notation. Instead of specifying the mask length, the actual mask is specified in a separate column. For example, 255.0.0.0 represents an IPv4 CIDR mask length of 8, and 255.255.255.255 represents a CIDR mask length of 32. These fields only apply to host, hostssl, and hostnossl records.
authentication-method	Specifies the authentication method to use when connecting. See “Authentication Methods” on page 7 for more details.

Authentication Methods

Basic authentication Methods

The following basic authentication methods are supported:

- **Password or MD5**
  Requires clients to provide a password, one of either:
  - **Md5**. A password transmitted as an MD5 hash
  - **Password**. A password transmitted in clear text

  Always use SSL connections to prevent password sniffing during transit.

  This is configurable, see Encrypting Passwords for more information.
- **Reject**
  Reject the connections with the matching parameters. You should typically use this to restrict access from specific hosts or insecure connections.
- **Ident**
  Authenticates based on the client's OS user name. You should only use this for local connections.

Configuration File Format

Sample entries in the `pg_hba.conf` file appear as follows:

```
Hostnossl     all    all         0.0.0.0    reject
hostssl       all    testuser    0.0.0.0/0  md5
Local         all    gpuser                 ident
```

Kerberos Authentication

You can authenticate against a Kerberos server (RFC 2743, 1964)

Format

The format of the Kerberos authentication is:

```
Servicename/hostname@realm
```

Options

- `Map`. Map system and database users
- `Include_realm`. Option to specify realm name included in the system-user name in the ident map file.
- `Krb_realm`. Specify the realm name for matching the principals.
- `Krb_server_hostname`. The hostname of the service principal.

Configuration File Format

Sample entries in the `pg_hba.conf` file appear as follows:

```
host    all all 0.0.0.0/0   krb5
Hostssl all all 0.0.0.0/0   krb5 map=krbmap
```

Kerberos Authentication Configuration

The following Kerberos server settings are specified in `postgresql.conf`:

- `krb_server_key <file>`. Sets the location of the Kerberos server key file
- `krb_srvname <string>`. Kerberos service name
- `krb_caseins_users <boolean>`. Case-insensitively

The default is off

The following client setting is specified as a connection parameter:

- `Krbsrvname`. The Kerberos service name to use for authentication

LDAP Authentication

You can authenticate against an LDAP directory.

- LDAPS and LDAP over TLS options encrypt the connection to the LDAP server.
- The connection from the client to the server is not encrypted unless SSL is enabled. Configure client connections to use SSL to encrypt connections from the client.
- To configure or customize LDAP settings, set the `LDAPCONF` environment variable with the path to the `ldap.conf` file and add this to the `greenplum_path.sh` script.

Configuring LDAP Authentication

Following are the recommended steps for configuring your system for LDAP authentication:

1. Set up the LDAP server with the database users/roles to be authenticated via LDAP.
2. On the database:
   a. Verify that the database users to be authenticated via LDAP exist on the database. LDAP is only used for verifying username/password pairs, so the roles should exist in the database.
   b. Update the `pg_hba.conf` file in the `$MASTER_DATA_DIRECTORY` to use LDAP as the authentication method for the respective users. Note that the first entry to match the user/role in the `pg_hba.conf` file will be used as the authentication mechanism, so the position of the entry in the file is important.
   c. Reload the server for the `pg_hba.conf` configuration settings to take effect (`gpstop -u`).

LDAP Authentication Parameters

Specify the following parameters `auth-options`

- `Ldapserver`. Names or IP addresses of LDAP servers to connect to. Multiple servers may be specified, separated by spaces.
- `Ldapprefix`. String to prepend to the user name when forming the DN to bind as, when doing simple bind authentication.
- `Ldapsuffix`. String to append to the user name when forming the DN to bind as, when doing simple bind authentication.
- `Ldapport`. Port number on LDAP server to connect to. If no port is specified, the LDAP library's default port setting will be used
- `Ldaptls`. Set to 1 to make the connection between PostgreSQL and the LDAP server use TLS encryption. Note that this only encrypts the traffic to the LDAP server — the connection to the client will still be unencrypted unless SSL is used.
- `ldapbasedn`. Root DN to begin the search for the user in, when doing search+bind authentication.
- `ldapbinddn`. DN of user to bind to the directory with to perform the search when doing search+bind authentication.
- `ldapbindpasswd`. Password for user to bind to the directory with to perform the search when doing search+bind authentication.
- `ldapsearchattribute`. Attribute to match against the user name in the search when doing search+bind authentication.

Example:

```
Ldapserver=ldap.greenplum.com prefix="cn=" suffix=",
dc=greenplum, dc=com"
```

Configuration File Format

Sample entries in the `pg_hba.conf` file appear as follows:

```
host all testuser 0.0.0.0/0 ldap ldap
ldapserver=ldapserver.greenplum.com ldapport=389
ldapprefix="cn=" ldapsuffix=",ou=people,dc=greenplum,dc=com"
```

```
hostssl     all         ldaprole      0.0.0.0/0     ldap
ldapserver=ldapserver.greenplum.com ldaptls=1
ldapprefix="cn=" ldapsuffix=",ou=people,dc=greenplum,dc=com"
```

SSL Client Authentication

SSL authentication compares the Common Name (cn) attribute of an SSL certificate provided by the connecting client during the SSSL handshake, to the requested database user name, hence the Database user should exist in the database. A map file can be used for mapping between system and database user names.

SSL Authentication Parameters

- Authentication method:
  - Cert
- Authentication options:
  - `Hostssl`. Connection type must be `hostssl`
  - `map=<mapping>`. User-name map
    This is specified in the `pg_ident.conf`, or in file specified in the `ident_file` server setting .

Configuration File Format

Sample entries in the `pg_hba.conf` file appear as follows:

```
Hostssl testdb certuser 192.168.0.0/16 cert
Hostssl testdb all 192.168.0.0/16 cert map=gpuser
```

OpenSSL Configuration

Greenplum Database reads the OpenSSL configuration file specified in `$GP_HOME/etc/openssl.cnf` by default. You can make changes to the default configuration for OpenSSL by modifying or updating this file and restarting the server.

Creating a Self-Signed Certificate

To create a quick self-signed certificate for the server, use the following OpenSSL command:

```
openssl req -new -text -out server.req
```

Fill out the information that openssl prompts for. Make sure you enter the local host name as `Common Name`; the challenge password can be left blank. The program will generate a key that is passphrase protected; it will not accept a passphrase that is less than four characters long. To remove the passphrase (as you must if you want automatic start-up of the server), run the following commands:

```
openssl rsa -in privkey.pem -out server.key rm privkey.pem
```

Enter the old passphrase to unlock the existing key. Then run the following command:

```
openssl req -x509 -in server.req -text -key server.key -out
server.crt
```

This turns the certificate into a self-signed certificate and copies the key and certificate to where the server will look for them. Finally, run the following command:

```
chmod og-rwx server.key
```

This sets the permissions on the file because the server will reject the file if its permissions are more liberal than this.

For more details on how to create your server private key and certificate, refer to the OpenSSL documentation.

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA) (either one of the global CAs or a local one) should be used in production so that clients can verify the server's identity. If all the clients are local to the organization, using a local CA is recommended.

Configuring postgresql.conf for SSL Authentication

The following Server settings need specifying in the `postgresql.conf` configuration file:

- `ssl <boolean>`. Enables SSL connections
- `ssl_renegotiation_limit <integer>`. Specifies the data limit before key renegotiation
- `ssl_ciphers` (string). Lists SSL ciphers that are allowed

The following SSL server files can be found in the Master Data Directory

- `server.crt`. Server certificate
- `server.key`. Server private key
- `root.crt`. Trusted certificate authorities
- `root.crl`. Certificates revoked by certificate authorities

Configuring the SSL Client Connection

- `sslmode <options>`
  - `Require`. Only use SSL connection.
    If a root CA file is present, verify the certificate in the same way as if verify-ca was specified
  - `verify-ca`. Only use an SSL connection.
    Verify that the server certificate is issued by a trusted CA
  - `verify-full`. Only use an SSL connection.
    Verify that the server certificate is issued by a trusted CA and that the server host name matches that in the certificate
- `Sslcert`. The file name of the client SSL certificate. Default= `~/.postgresql/postgresql.crt`
- `Sslkey`. The secret key used for the client certificate. `Default=~/.postgresql/postgresql.key`

- `Sslrootcert`. The name of a file containing SSL Certificate Authority certificate(s). `Default=~/.postgresql/root.crt`
- `Sslcrl`. The name of the SSL certificate revocation list. `Default=~/.postgresql/root.crl`

The client connection parameters can be set using the following environment variables.

```
Sslmode – PGSSLMODE
Sslkey – PGSSLKEY
Sslrootcert – PGSSLROOTCERT
Sslcert – PGSSLCERT
Sslcrl – PGSSLCRL
```

PAM Based Authentication

PAM are Pluggable Authentication Modules. It is similar to password-based authentication and is used to validate username/password pairs. PAM authenication only works if the users already exist in the database.

Parameters

- `pamservice`. The default PAM service is postgresql

Note that if PAM is set up to read `/etc/shadow`, authentication will fail because the PostgreSQL server is started by a non-root user.

Configuration File Format

Sample entries in the pg_hba.conf file appear as follows:

```
local    all gpuser am pamservice=postgresql
```

Radius Authentication

RADIUS authentication works by sending an Access Request message of type ‘Authenticate Only’ to a configured RADIUS server. It includes parameters for user name, password (encrypted) and NAS Identifier. The request is encrypted using the shared secret specified in the radiussecret option. The RADIUS server responds with either Access Accept or Access Reject.

Note that Radius accounting is not supported.

Radius authenication only works if the users already exist in the database.

The Radius encryption vector requires SSL to be enabled in order to be cryptographically strong.

Authentication Options

- `Radiusserver`. Name of the radius server
- `Radiussecret`. Radius shared secret
- `Radiusport`. Port to connect to on the Radius server
- `Radiusidentifier`. NAS identifier in Radius requests

Configuration File Format

Sample entries in the `pg_hba.conf` file appear as follows:

```
Hostssl  all all 0.0.0.0/0 radius radiusserver=servername
radiussecret=<sharedsecret>
```

Editing the pg_hba.conf File

This example shows how to edit the `pg_hba.conf` file of the master to allow remote client access to all databases from all roles using encrypted password authentication.

Note: For a more secure system, consider removing all connections that use trust authentication from your master `pg_hba.conf`. Trust authentication means the role is granted access without any authentication, therefore bypassing all security. Replace trust entries with ident authentication if your system has an ident service available.

Editing pg_hba.conf

1. Open the file `$MASTER_DATA_DIRECTORY/pg_hba.conf` in a text editor.
2. Add a line to the file for each type of connection you want to allow. Records are read sequentially, so the order of the records is significant. Typically, earlier records will have tight connection match parameters and weaker authentication methods, while later records will have looser match parameters and stronger authentication methods. For example:

```
# allow the gpadmin user local access to all databases
# using ident authentication
local   all   gpadmin   ident         sameuser
host    all   gpadmin   127.0.0.1/32  ident
host    all   gpadmin   ::1/128       ident
# allow the 'dba' role access to any database from any
# host with IP address 192.168.x.x and use md5 encrypted
# passwords to authenticate the user
# Note that to use SHA-256 encryption, replace md5 with
# password in the line below
host    all   dba   192.168.0.0/32  md5
# allow all roles access to any database from any
# host and use ldap to authenticate the user. Greenplum role
# names must match the LDAP common name.
host    all   all   192.168.0.0/32  ldap ldapserver=usldap1
ldapport=1389 ldapprefix="cn="
ldapsuffix=",ou=People,dc=company,dc=com"
```

3. Save and close the file.
4. Reload the `pg_hba.conf` configuration file for your changes to take effect:

```
$ gpstop -u
```

Note: Note that you can also control database access by setting object privileges as described in "Managing Object Privileges" on page 25. The `pg_hba.conf` file just controls who can initiate a database session and how those connections are authenticated.

Limiting Concurrent Connections

To limit the number of active concurrent sessions to your Greenplum Database system, you can configure the `max_connections` server configuration parameter. This is a *local* parameter, meaning that you must set it in the `postgresql.conf` file of the master, the standby master, and each segment instance (primary and mirror). The value of `max_connections` on segments must be 5-10 times the value on the master.

When you set `max_connections`, you must also set the dependent parameter `max_prepared_transactions`. This value must be at least as large as the value of `max_connections` on the master, and segment instances should be set to the same value as the master.

For example:

In `$MASTER_DATA_DIRECTORY/postgresql.conf` (including standby master):

```
max_connections=100
max_prepared_transactions=100
```

In *`SEGMENT_DATA_DIRECTORY`*`/postgresql.conf` for all segment instances:

```
max_connections=500
max_prepared_transactions=100
```

To change the number of allowed connections

1. Stop your Greenplum Database system:

   ```
   $ gpstop
   ```

2. On your master host, edit `$MASTER_DATA_DIRECTORY/postgresql.conf` and change the following two parameters:

   `max_connections` (the number of active user sessions you want to allow plus the number of `superuser_reserved_connections`)

   `max_prepared_transactions` (must be greater than or equal to `max_connections`)

3. On each segment instance, edit *`SEGMENT_DATA_DIRECTORY`*`/postgresql.conf` and and change the following two parameters:

   `max_connections` (must be 5-10 times the value on the master)

   `max_prepared_transactions` (must be equal to the value on the master)

4. Restart your Greenplum Database system:

```
$ gpstart
```

Note: Raising the values of these parameters may cause Greenplum Database to request more shared memory. To mitigate this effect, consider decreasing other memory-related parameters such as `gp_cached_segworkers_threshold`.

Encrypting Client/Server Connections

Greenplum Database has native support for SSL connections between the client and the master server. SSL connections prevent third parties from snooping on the packets, and also prevent man-in-the-middle attacks. SSL should be used whenever the client connection goes through an insecure link, and must be used whenever client certificate authentication is used.

Note: For information about encrypting data between the `gpfdist` server and Greenplum Database segment hosts, see Chapter 4, "Data Encryption".

To enable SSL requires that OpenSSL be installed on both the client and the master server systems. Greenplum can be started with SSL enabled by setting the server configuration parameter `ssl=on` in the master `postgresql.conf`. When starting in SSL mode, the server will look for the files `server.key` (server private key) and `server.crt` (server certificate) in the master data directory. These files must be set up correctly before an SSL-enabled Greenplum system can start.

Important: Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and the database startup fails with an error if one is required.

A self-signed certificate can be used for testing, but a certificate signed by a certificate authority (CA) should be used in production, so the client can verify the identity of the server. Either a global or local CA can be used. If all the clients are local to the organization, a local CA is recommended.

Creating a Self-signed Certificate without a Passphrase for Testing Only

To create a quick self-signed certificate for the server for testing, use the following OpenSSL command:

```
# openssl req -new -text -out server.req
```

Fill out the information that openssl asks for. Be sure to enter the local host name as *Common Name*. The challenge password can be left blank.

The program will generate a key that is passphrase protected, and does not accept a passphrase that is less than four characters long.

To use this certificate with Greenplum Database, remove the passphrase with the following commands:

```
# openssl rsa -in privkey.pem -out server.key
# rm privkey.pem
```

Enter the old passphrase when prompted to unlock the existing key.

Then, enter the following command to turn the certificate into a self-signed certificate and to copy the key and certificate to a location where the server will look for them.

```
# openssl req -x509 -in server.req -text -key server.key -out server.crt
```

Finally, change the permissions on the key with the following command. The server will reject the file if the permissions are less restrictive than these.

```
# chmod og-rwx server.key
```

For more details on how to create your server private key and certificate, refer to the OpenSSL documentation.

3. Auditing

The Greenplum Database is capable of auditing a variety of events, including startup and shutdown of the system, segment database failures, SQL statements that result in an error, all connection attempts and disconnections. The Greenplum Database also logs SQL statements and information regarding SQL statements, and can be configured in a variety of ways to record audit information with more or less detail. The `log_error_verbosity` configuration parameter controls the amount of detail written in the server log for each message that is logged. Similarly, the `log_min_error_statement` parameter allows administrators to configure the level of detail recorded specifically for SQL statements, and the `log_statement` parameter determines the kind of SQL statements that are audited. The Greenplum Database records the username for all auditable events, when the event is initiated by a subject outside the Greenplum Database.

The Greenplum Database prevents unauthorized modification and deletion of audit records by only allowing administrators with an appropriate role to perform any operations on log files. Logs are stored in a proprietary format using comma-separated values (CSV). Each segment and the master stores its own log files, although these can be accessed remotely by an administrator. The Greenplum Database also authorizes overwriting of old log files via the `log_truncate_on_rotation` parameter. This is a local parameter and must be set on each segment and master configuration file.

Greenplum provides an administrative schema called gp_toolkit that you can use to query log files, as well as system catalogs and operating enviroment for system status information. For more information, including usage, refer to *The gp_tookit Administrative Schema* appendix of the *Greenplum Administrator Guide*.

Viewing the Database Server Log Files

Every database instance in Greenplum Database (master and segments) is a running a PostgreSQL database server with its own server log file. Daily log files are created in the `pg_log` directory of the master and each segment data directory.

Log File Format

The server log files are written in comma-separated values (CSV) format. Not all log entries will have values for all of the log fields. For example, only log entries associated with a query worker process will have the `slice_id` populated. Related log entries of a particular query can be identified by its session identifier (`gp_session_id`) and command identifier (`gp_command_count`).

The following fields are written to the log:

Table 3.1 Greenplum Database Server Log Format

#	Field Name	Data Type	Description
1	event_time	timestamp with time zone	Time that the log entry was written to the log
2	user_name	varchar(100)	The database user name
3	database_name	varchar(100)	The database name
4	process_id	varchar(10)	The system process id (prefixed with "p")
5	thread_id	varchar(50)	The thread count (prefixed with "th")
6	remote_host	varchar(100)	On the master, the hostname/address of the client machine. On the segment, the hostname/address of the master.
7	remote_port	varchar(10)	The segment or master port number
8	session_start_time	timestamp with time zone	Time session connection was opened
9	transaction_id	int	Top-level transaction ID on the master. This ID is the parent of any subtransactions.
10	gp_session_id	text	Session identifier number (prefixed with "con")
11	gp_command_count	text	The command number within a session (prefixed with "cmd")
12	gp_segment	text	The segment content identifier (prefixed with "seg" for primaries or "mir" for mirrors). The master always has a content id of -1.
13	slice_id	text	The slice id (portion of the query plan being executed)
14	distr_tranx_id	text	Distributed transaction ID
15	local_tranx_id	text	Local transaction ID
16	sub_tranx_id	text	Subtransaction ID
17	event_severity	varchar(10)	Values include: LOG, ERROR, FATAL, PANIC, DEBUG1, DEBUG2
18	sql_state_code	varchar(10)	SQL state code associated with the log message
19	event_message	text	Log or error message text
20	event_detail	text	Detail message text associated with an error or warning message
21	event_hint	text	Hint message text associated with an error or warning message
22	internal_query	text	The internally-generated query text
23	internal_query_pos	int	The cursor index into the internally-generated query text
24	event_context	text	The context in which this message gets generated
25	debug_query_string	text	User-supplied query string with full detail for debugging. This string can be modified for internal use.
26	error_cursor_pos	int	The cursor index into the query string

Table 3.1 Greenplum Database Server Log Format

#	Field Name	Data Type	Description
27	func_name	text	The function in which this message is generated
28	file_name	text	The internal code file where the message originated
29	file_line	int	The line of the code file where the message originated
30	stack_trace	text	Stack trace text associated with this message

Searching the Greenplum Database Server Log Files

Greenplum provides a utility called `gplogfilter` that can be used to search through a Greenplum Database log file for entries matching the specified criteria. By default, this utility searches through the Greenplum master log file in the default logging location. For example, to display the last three lines of the master log file:

```
$ gplogfilter -n 3
```

You can also use `gplogfilter` to search through all segment log files at once by running it through the `gpssh` utility. For example, to display the last three lines of each segment log file:

```
$ gpssh -f seg_host_file
   => source /usr/local/greenplum-db/greenplum_path.sh
   => gplogfilter -n 3 /gpdata/gp*/pg_log/gpdb*.log
```

Security-Related Audit Settings

The following are the Greenplum security-related audit (or logging) server configuration parameters that are set in the `postgresql.conf` configuration file:

Table 3.2 Greenplum Database Security-Related Server Configuration Parameters

Field Name	Value Range	Default	Description
`Log_connections`	Boolean	off	This outputs a line to the server log detailing each successful connection. Some client programs, like psql, attempt to connect twice while determining if a password is required, so duplicate “connection received” messages do not always indicate a problem.
`Log_disconnections`	Boolean	off	This outputs a line in the server log at termination of a client session, and includes the duration of the session.

`Log_statement`	`NONE`
	`DDL`
	`MOD`
	`ALL`
		`ALL`	Controls which SQL statements are logged. DDL logs all data definition commands like `CREATE`, `ALTER`, and `DROP` commands. MOD logs all DDL statements, plus `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, and `COPY FROM`. `PREPARE` and `EXPLAIN ANALYZE` statements are also logged if their contained command is of an appropriate type.

Table 3.2 Greenplum Database Security-Related Server Configuration Parameters

Field Name	Value Range	Default	Description
Log_hostname	Boolean	off	By default, connection log messages only show the IP address of the connecting host. Turning on this option causes logging of the host name as well. Note that depending on your host name resolution setup this might impose a non-negligible performance penalty.
Log_duration	Boolean	off	Causes the duration of every completed statement which satisfies log_statement to be logged.
Log_error_verbosity	TERSE DEFAULT VERBOSE	DEFAULT	Controls the amount of detail written in the server log for each message that is logged.
log_min_duration_statement	number of milliseconds, 0, -1	-1	Logs the statement and its duration on a single log line if its duration is greater than or equal to the specified number of milliseconds. Setting this to 0 will print all statements and their durations. -1 disables the feature. For example, if you set it to 250 then all SQL statements that run 250ms or longer will be logged. Enabling this option can be useful in tracking down unoptimized queries in your applications.
log_min_messages	DEBUG5 DEBUG4 DEBUG3 DEBUG2 DEBUG1 INFO NOTICE WARNING ERROR LOG FATAL PANIC	NOTICE	Controls which message levels are written to the server log. Each level includes all the levels that follow it. The later the level, the fewer messages are sent to the log.
log_rotation_age	Any valid time expression (number and unit)	1d	Determines the maximum lifetime of an individual log file. After this time has elapsed, a new log file will be created. Set to zero to disable time-based creation of new log files.
log_statement_stats	Boolean	off	For each query, write total performance statistics of the query parser, planner, and executor to the server log. This is a crude profiling instrument.
log_truncate_on_rotation	Boolean	off	Truncates (overwrites), rather than appends to, any existing log file of the same name. Truncation will occur only when a new file is being opened due to time-based rotation. For example, using this setting in combination with a log_filename such as gpseg#-%H.log would result in generating twenty-four hourly log files and then cyclically overwriting them. When off, pre-existing files will be appended to in all cases.

4. Data Encryption

Data can be encrypted at various points:

- Encryption of Data in Transit.
  - Encryption of data between the `gpfdist` server and Greenplum Database segment hosts.
  - Encryption of client communications with the Master server.

    Greenplum Database supports SSL by default.

    This behavior is controlled by the server configuration parameter `ssl`. This parameter is `off` by default. Setting this to `on` allows the client communications to the master to be encrypted. Note that this requires a manual set up of SSL on the master database.

    For more information, see "Encrypting Client/Server Connections" on page 15
- Encryption of Data at Rest. Encryption of data stored in the database.
  - Greenplum provides column-level encryption using the pgcrypto package of encryption/decryptions functions. See "pgcrypto" on page 23.

Encryption of Data in Transit

Greenplum Database 4.2.1 and above supports SSL for encryption of data in transit between the `gpfdist` server and Greenplum Database segment hosts.

gpfdists

The `gpfdists` protocol is a secure version of `gpfdist`, which enables encrypted communication and secure identification of the file server and the Greenplum Database to protect against attacks such as eavesdropping and man-in-the-middle attacks.

Note: For more detail about gpfdist, refer to the *Greenplum Database Administrator Guide*.

The protocol implements SSL security in a client/server scheme, with the following notable features:

- Client certificates are required.
- Multi-lingual certificates are not supported.
- A Certificate Revocation List (CRL) is not supported.
- The TLSv1 protocol is used with the `TLS_RSA_WITH_AES_128_CBC_SHA` encryption algorithm. These SSL parameters cannot be changed.
- SSL renegotiation is supported.
- The SSL ignore host mismatch parameter is set to false.

- Private keys containing a passphrase are not supported for the `gpfdist` file server (`server.key`) and for the Greenplum Database (client.key).
- Issuing certificates that are appropriate for the operating system in use is the user's responsibility. Generally, converting certificates as shown in `https://www.sslshopper.com/ssl-converter.html` is supported.

Note: A server that was started with the `gpfdist --ssl` option can only communicate with the `gpfdists` protocol. A server that was started with `gpfdist` without the `--ssl` option can only communicate with the `gpfdist` protocol.

There are two ways to use the `gpfdists` protocol, as follows:

- Run `gpfdist` with the `--ssl` option and then use the `gpfdists` protocol in the `LOCATION` clause of a `CREATE EXTERNAL TABLE` statement.
- Use a YAML Control File with the SSL option set to true and run gpload.. Running `gpload` starts the `gpfdist` server with the `--ssl` option and then uses the `gpfdists` protocol.

Important: Do not protect the private key with a passphrase. The server does not prompt for a passphrase for the private key, and loading data fails with an error if one is required.

When using `gpfdists`, the following client certificates must be located in the `$PGDATA/gpfdists` directory on each segment.

- The client certificate file, `client.crt`
- The client private key file, `client.key`
- The trusted certificate authorities, `root.crt`

Note: When using `gpload` or `gpfdist` with SSL, you must specify the location of the server certificates location in the YAML file for `gpload` or in the `--ssl` option for `gpfdist`.

Below is an example of loading data into an external table securely:

This example creates a readable external table named `ext_expenses` using the `gpfdists` protocol from all files with the txt extension. The files are formatted with a pipe (|) as the column delimiter and an empty space as null.

First, run `gpfdist` with the `--ssl` option. Then, execute the following command.

```
=# CREATE EXTERNAL TABLE ext_expenses ( name text,
date date, amount float4, category text, desc1 text )
LOCATION ('gpfdists://etlhost-1:8081/*.txt',
'gpfdists://etlhost-2:8082/*.txt')
FORMAT 'TEXT' ( DELIMITER '|' NULL ' ') ;
```

Encryption of Data at Rest

pgcrypto

PostgreSQL provides an optional package of encryption/decryption functions called pgcrypto, which can be installed and used in Greenplum Database. The pgcrypto package is not installed by default with Greenplum Database, however you can download a pgcrypto package from the EMC Download Center, then use the Greenplum Package Manager (`gppkg`) to install pgcrypto across your entire cluster.

The pgcrypto functions allow database administrators to store certain columns of data in encrypted form. This adds an extra layer of protection for sensitive data, as data stored in Greenplum Database in encrypted form cannot be read by users who do not have the encryption key, nor be read directly from the disks.

It is important to note that the pgcrypto functions run inside database server. That means that all the data and passwords move between pgcrypto and the client application in clear-text. For optimal security, you should:

- Connect locally or use SSL connections.
- Trust both system and database administrator.

When compiled with `zlib`, pgcrypto encryption functions are able to compress data before encrypting. All pgcrypto functions run inside the database server. That means that all the data and passwords move between pgcrypto and client applications in clear text.

PgCrypto has various levels of encryption ranging from basic to advanced built-in functions.

Table 4.1 Pgcrypto Supported Encryption Functions:

Value Functionality	Built-in	With OpenSSL
MD5	yes	yes
SHA1	yes	yes
SHA224/256/384/512	yes	yes (note 1)
Other digest algorithms	no	yes (note 2)
Blowfish	yes	yes
AES	yes	yes (note 3)
DES/3DES/CAST5	no	yes
Raw Encryption	yes	yes
PGP Symmetric-Key	yes	yes
PGP Public Key	yes	yes

5. Managing Roles and Privileges

Greenplum Database manages database access permissions using the concept of *roles*. The concept of roles subsumes the concepts of *users* and *groups*. A role can be a database user, a group, or both. Roles can own database objects (for example, tables) and can assign privileges on those objects to other roles to control access to the objects. Roles can be members of other roles, thus a member role can inherit the object privileges of its parent role.

Every Greenplum Database system contains a set of database roles (users and groups). Those roles are separate from the users and groups managed by the operating system on which the server runs. However, for convenience you may want to maintain a relationship between operating system user names and Greenplum Database role names, since many of the client applications use the current operating system user name as the default.

In Greenplum Database, users log in and connect through the master instance, which then verifies their role and access privileges. The master then issues out commands to the segment instances behind the scenes as the currently logged in role.

Roles are defined at the system level, meaning they are valid for all databases in the system.

In order to bootstrap the Greenplum Database system, a freshly initialized system always contains one predefined *superuser* role (also referred to as the system user). This role will have the same name as the operating system user that initialized the Greenplum Database system. Customarily, this role is named `gpadmin`. In order to create more roles you first have to connect as this initial role.

Security Best Practices for Roles and Privileges

- **Secure the gpadmin system user.** Greenplum requires a UNIX user id to install and initialize the Greenplum Database system. This system user is referred to as `gpadmin` in the Greenplum documentation. This `gpadmin` user is the default database superuser in Greenplum Database, as well as the file system owner of the Greenplum installation and its underlying data files. This default administrator account is fundamental to the design of Greenplum Database. The system cannot run without it, and there is no way to limit the access of this `gpadmin` user id. This `gpadmin` user can bypass all security features of Greenplum Database. Anyone who logs on to a Greenplum host as this user id can read, alter or delete any data; including system catalog data and database access rights. Therefore, it is very important to secure the `gpadmin` user id and only provide access to essential system administrators. Administrators should only log in to Greenplum as `gpadmin` when performing certain system maintenance tasks (such as upgrade or expansion). Database users should never log on as `gpadmin`, and ETL or production workloads should never run as `gpadmin`.

- **Assign a distinct role to each user that logs in.** For logging and auditing purposes, each user that is allowed to log in to Greenplum Database should be given their own database role. For applications or web services, consider creating a distinct role for each application or service. See “Creating New Roles (Users)” in the *Greenplum Database Administrator Guide*.
- **Use groups to manage access privileges.** See “Creating Groups (Role Membership)” in the *Greenplum Database Administrator Guide*.
- **Limit users who have the SUPERUSER role attribute.** Roles that are superusers bypass all access privilege checks in Greenplum Database, as well as resource queuing. Only system administrators should be given superuser rights. See “Altering Role Attributes” in the *Greenplum Database Administrator Guide*.

Managing Object Privileges

When an object (table, view, sequence, database, function, language, schema, or tablespace) is created, it is assigned an owner. The owner is normally the role that executed the creation statement. For most kinds of objects, the initial state is that only the owner (or a superuser) can do anything with the object. To allow other roles to use it, privileges must be granted. Greenplum Database supports the following privileges for each object type:

Table 5.1 Object Privileges

Object Type	Privileges
Tables, Views, Sequences	SELECT INSERT UPDATE DELETE RULE ALL
External Tables	SELECT RULE ALL
Databases	CONNECT CREATE TEMPORARY | TEMP ALL
Functions	EXECUTE
Procedural Languages	USAGE
Schemas	CREATE USAGE ALL

Note: Privileges must be granted for each object individually. For example, granting ALL on a database does not grant full access to the objects within that database. It only grants all of the database-level privileges (CONNECT, CREATE, TEMPORARY) to the database itself.

Use the `GRANT` SQL command to give a specified role privileges on an object. For example:

```
=# GRANT INSERT ON mytable TO jsmith;
```

To revoke privileges, use the `REVOKE` command. For example:

```
=# REVOKE ALL PRIVILEGES ON mytable FROM jsmith;
```

You can also use the `DROP OWNED` and `REASSIGN OWNED` commands for managing objects owned by deprecated roles (Note: only an object's owner or a superuser can drop an object or reassign ownership). For example:

```
=# REASSIGN OWNED BY sally TO bob;
=# DROP OWNED BY visitor;
```

Simulating Row and Column Level Access Control

Greenplum Database access control corresponds roughly to the Orange Book 'C2' level of security, not the 'B1' level. Greenplum Database currently supports access privileges at the object level. Row-level or column-level access is not supported, nor is labeled security.

Row-level and column-level access can be simulated using views to restrict the columns and/or rows that are selected. Row-level labels can be simulated by adding an extra column to the table to store sensitivity information, and then using views to control row-level access based on this column. Roles can then be granted access to the views rather than the base table. While these workarounds do not provide the same as "B1" level security, they may still be a viable alternative for many organizations that

Encrypting Passwords

In Greenplum Database versions before 4.2.1, passwords were encrypted using MD5 hashing by default. Since some customers require cryptographic algorithms that meet the Federal Information Processing Standard140-2, as of version 4.2.1, Greenplum Database features RSA's BSAFE implementation that lets customers store passwords hashed using SHA-256 encryption.

To use SHA-256 encryption, you must set a parameter either at the system or the session level. This technical note outlines how to use a server parameter to implement SHA-256 encrypted password storage. Note that in order to use SHA-256 encryption for storage, the client authentication method must be set to `password` rather than the default, `MD5`. (See "Encrypting Client/Server Connections" on page 15 for more details.) This means that the password is transmitted in clear text over the network, so we highly recommend that you set up SSL to encrypt the client server communication channel.

Enabling SHA-256 Encryption

You can set your chosen encryption method system-wide or on a per-session basis. There are three encryption methods available: `SHA-256`, `SHA-256-FIPS`, and `MD5` (for backward compatibility). The `SHA-256-FIPS` method requires that FIPS compliant libraries are used.

System-wide

To set the `password_hash_algorithm` server parameter on a complete Greenplum system (master and its segments):

1. Log into your Greenplum Database instance as a superuser.
2. Execute `gpconfig` with the `password_hash_algorithm` set to `SHA-256` (or `SHA-256-FIPS` to use the FIPS-compliant libraries for SHA-256)

   ```
   $ gpconfig -c password_hash_algorithm -v 'SHA-256'
   ```

   or:

   ```
   $ gpconfig -c password_hash_algorithm -v 'SHA-256-FIPS'
   ```

3. Verify the setting:

   ```
   $ gpconfig -s
   ```

   You will see:

   ```
   Master value: SHA-256
   Segment value: SHA-256
   ```

   or:

   ```
   Master  value: SHA-256-FIPS
   Segment value: SHA-256-FIPS
   ```

Individual Session

To set the `password_hash_algorithm` server parameter for an individual session:

1. Log into your Greenplum Database instance as a superuser.
2. Set the `password_hash_algorithm` to `SHA-256` (or `SHA-256-FIPS` to use the FIPS-compliant libraries for SHA-256):

   ```
   # set password_hash_algorithm = 'SHA-256'
   SET
   ```

or:

   ```
   # set password_hash_algorithm = 'SHA-256-FIPS'
   SET
   ```

3. Verify the setting:

   ```
   # show password_hash_algorithm;
   password_hash_algorithm
   ```

   You will see:

   ```
   SHA-256
   ```

   or:

```
SHA-256-FIPS
```

Example

Following is an example of how the new setting works:

1. Login in as a super user and verify the password hash algorithm setting:
   ```
   # show password_hash_algorithm
   password_hash_algorithm
   ------------------------------
   SHA-256-FIPS
   ```
2. Create a new role with password that has login privileges.
   ```
   create role testdb with password 'testdb12345#' LOGIN;
   ```
3. Change the client authentication method to allow for storage of SHA-256 encrypted passwords:
   Open the `pg_hba.conf` file on the master and add the following line:
   ```
   host all testdb 0.0.0.0/0 password
   ```
4. Restart the cluster.
5. Login to the database as user just created `testdb`.
   ```
   psql -U testdb
   ```
6. Enter the correct password at the prompt.
7. Verify that the password is stored as a SHA-256 hash.
   Note that password hashes are stored in `pg_authid.rolpasswod`
   a. Login as the super user.
   b. Execute the following:
      ```
      # select rolpassword from pg_authid where rolname =
      'testdb';
      Rolpassword
      -----------
      sha256<64 hexidecimal characters>
      ```

Time-based Authentication

Greenplum Database enables the administrator to restrict access to certain times by role. Use the `CREATE ROLE` or `ALTER ROLE` commands to specify time-based constraints.

Access can be restricted by day or by day and time. The constraints are removable, without deleting and recreating the role.

Time-based constraints only apply to the role to which they are assigned. If a role is a member of another role that contains a time constraint, the time constraint is not inherited.

Time-based constraints are enforced only during login. The `SET ROLE` and `SET SESSION AUTHORIZATION` commands are not affected by any time-based constraints.

Required Permissions

`Superuser` or `CREATEROLE` privileges are required to set time-based constraints for a role. No one can add time-based constraints to a superuser.

How to Add Time-based Constraints

There are two ways to add time-based constraints. Use the keyword `DENY` in the `CREATE ROLE` or `ALTER ROLE` command followed by one of the following.

- A day and optionally a time when access is restricted. For example, no access on Wednesdays.
- An interval, that is a beginning and ending day and optional time, when access is restricted. For example, no access from Wednesday 10 p.m.through Thursday at 8 a.m.

You can specify more than one restriction; for example, no access Wednesdays at any time and no access on Fridays between 3:00 p.m. and 5:00 p.m.

Specifying the Day and Time

There are two ways to specify a day. Use the word `DAY` followed either the English term for the weekdays, in single quotation marks or a number between 0 and 6, as shown in the table below.

Table 5.2 Values for `DAY`

English Term	Number
`DAY 'Sunday'`	`DAY 0`
`DAY 'Monday'`	`DAY 1`
`DAY 'Tuesday'`	`DAY 2`
`DAY 'Wednesday'`	`DAY 3`
`DAY 'Thursday'`	`DAY 4`
`DAY 'Friday'`	`DAY 5`
`DAY 'Saturday'`	`DAY 6`

A time of day is specified in either 12- or 24-hour format. The word `TIME` is followed by the specification in single quotation marks. Only hours and minutes are specified and are separated by a colon (`:`). If using a 12-hour format, add `AM` or `PM` at the end. The following examples show various time specifications.

`TIME '14:00'` (24-hour time implied)

`TIME '02:00 PM'` (12-hour time specified by PM)

`TIME '02:00'` (24-hour time implied) This is equivalent to `TIME '02:00 AM'`.

Note: Time-based authentication is enforced with the server time. Timezones are disregarded.

Specifying an Interval

To specify an interval of time during which access is denied, use two day/time specifications with the words `BETWEEN` and `AND`, as shown. `DAY` is always required.

```
BETWEEN DAY 'Monday' AND DAY 'Tuesday'
BETWEEN DAY 'Monday' TIME '00:00' AND
        DAY 'Monday' TIME '01:00'
BETWEEN DAY 'Monday' TIME '12:00 AM' AND
        DAY 'Tuesday' TIME '02:00 AM'
BETWEEN DAY 'Monday' TIME '00:00' AND
        DAY 'Tuesday' TIME '02:00'
BETWEEN DAY 1 TIME '00:00' AND
        DAY 2 TIME '02:00'
```

The last three statements are equivalent.

Note: Intervals of days cannot wrap past Saturday.

The following syntax is not correct:

Incorrect: `DENY BETWEEN DAY 'Saturday' AND DAY 'Sunday'`

The correct specification uses two `DENY` clauses, as follows:

```
DENY DAY 'Saturday'
DENY DAY 'Sunday'
```

Examples

The following examples demonstrate creating a role with time-based constraints and modifying a role to add time-based constraints.Only the statements needed for time-based constraints are shown. For more details on creating and altering roles see the descriptions of `CREATE ROLE` and `ALTER ROLE` in Appendix A in the *Greenplum Database Administrator Guide*.

Example 1 - Create a New Role with Time-based Constraints

No access is allowed on weekends.

```
CREATE ROLE generaluser
DENY DAY 'Saturday'
DENY DAY 'Sunday'
...
```

Example 2 - Alter a Role to add Time-based Constraints

No access is allowed every night between 2:00 a.m. and 4:00 a.m.

```
ALTER ROLE generaluser
DENY BETWEEN DAY 'Monday' TIME '02:00' AND DAY 'Monday' TIME '04:00'
DENY BETWEEN DAY 'Tuesday' TIME '02:00' AND DAY 'Tuesday' TIME '04:00'
DENY BETWEEN DAY 'Wednesday' TIME '02:00' AND DAY 'Wednesday' TIME '04:00'
DENY BETWEEN DAY 'Thursday' TIME '02:00' AND DAY 'Thursday' TIME '04:00'
DENY BETWEEN DAY 'Friday' TIME '02:00' AND DAY 'Friday' TIME '04:00'
DENY BETWEEN DAY 'Saturday' TIME '02:00' AND DAY 'Saturday' TIME '04:00'
DENY BETWEEN DAY 'Sunday' TIME '02:00' AND DAY 'Sunday' TIME '04:00'
...
```

Example 3 - Alter a Role to add Time-based Constraints

No access is allowed Wednesdays or Fridays between 3:00 p.m. and 5:00 p.m.

```
ALTER ROLE generaluser
DENY DAY 'Wednesday'
DENY BETWEEN DAY 'Friday' TIME '15:00' AND DAY 'Friday' TIME '17:00'
```

Dropping a Time-based Restriction

To remove a time-based restriction, use the `ALTER ROLE` command. Enter the keywords `DROP DENY FOR` followed by a day/time specification to drop.

```
DROP DENY FOR DAY 'Sunday'
```

Any constraint containing all or part of the conditions in a `DROP` clause is removed. For example, if an existing constraint denies access on Mondays and Tuesdays, and the `DROP` clause removes constraints for Mondays, the existing constraint is completely dropped. The `DROP` clause completely removes all constraints that overlap with the contraint in the drop clause. The overlapping constraints are completely removed even if they contain more restrictions that the restrictions mentioned in the `DROP` clause.

Example 1 - Remove a Time-based Restriction from a Role

```
ALTER ROLE generaluser
DROP DENY FOR DAY 'Monday'
...
```

This statement would remove all constraints that overlap with a Monday constraint for the role *generaluser* in Example 2, even if there are additional constraints.

A. Common Criteria Certification

The Greenplum Database software release version 4.2 has been certified to meet the security requirements as defined in the Common Criteria Certification at an assurance level of EAL 2+ with conformance to the US Government Database Protection Profile.

Common Criteria is a globally recognized standard for evaluating the security featuresand declared assurance claims of information technology products.

- The Common Criteria Portal listing certified products is available here: http://www.commoncriteriaportal.org/products/?expand#ALL
- The Canadian scheme that the Greenplum Database has been certified under: http://www.commoncriteriaportal.org/products/?expand#ALL