Pivotal

PRODUCT DOCUMENTATION

Pivotal™ Greenplum Database®

Version 4.3

Security Configuration Guide

Rev: A03

Notice

Copyright

Revised February, 2016

Contents

Chapter 1: About This Guide.....5
About the Greenplum Database Documentation Set.....6
Document Conventions.....7
Text Conventions.....7
Command Syntax Conventions.....8
Getting Support.....9
Product Information and Technical Support.....9

Chapter 2: Securing the Database.....10

Chapter 3: Greenplum Database Ports and Protocols.....11

Chapter 4: Configuring Client Authentication.....15
Allowing Connections to Greenplum Database.....16
Editing the pg_hba.conf File.....18
Authentication Methods.....19
SSL Client Authentication.....22
PAM Based Authentication.....24
Radius Authentication.....25
Limiting Concurrent Connections.....26
Encrypting Client/Server Connections.....27

Chapter 5: Configuring Database Authorization.....28
Access Permissions and Roles.....29
Managing Object Privileges.....30
Using SSH-256 Encryption.....31
Setting Encryption Method System-wide.....31
Setting Encryption Method for an Individual Session.....32
Restricting Access by Time.....34
Dropping a Time-based Restriction.....36

Chapter 6: Greenplum Command Center Security.....37

Chapter 7: Auditing.....39

Chapter 8: Encrypting Data and Database Connections.....44
Encrypting gpfdist Connections.....45
Encrypting Data at Rest with pgcrypto.....46

Chapter 9: Configuring IPsec for Greenplum Database.....54
IPsec Overview.....55

Installing Openswan....57
Configuring Openswan Connections....60

Chapter 10: Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster....64
Prerequisites....65
Configuring the Greenplum Cluster....66
Creating and Installing Keytab Files....68
Configuring gphdfs for Kerberos....70
Testing Greenplum Database Access to HDFS....71
Troubleshooting HDFS with Kerberos....72

Chapter 11: Security Best Practices....74

Chapter 1

About This Guide

This guide describes how to secure a Greenplum Database system. The guide consists of the following sections:

- *Securing the Database* introduces Greenplum Database security topics.
- *Greenplum Database Ports and Protocols* lists network ports and protocols used within the Greenplum cluster.
- *Configuring Client Authentication* describes the available methods for authenticating Greenplum Database clients.
- *Configuring Database Authorization* describes how to restrict access to database data at the user level by using roles and permissions.
- *Auditing* describes Greenplum Database events that are logged and should be monitored to detect security threats.
- *Encrypting Data and Database Connections* describes how to encrypt data in the database or in transit over the network, to protect from evesdroppers or man-in-the-middle attacks.
- *Configuring IPsec for Greenplum Database* provides information about using IPsec with Greenplum Database.
- *Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster* provides steps for configuring Greenplum Database to access external tables in a Hadoop cluster secured with Kerberos.
- *Security Best Practices* provides steps for securing Greenplum Database hosts and the cluster.

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

Because Greenplum Database is based on PostgreSQL 8.2.15, this guide assumes some familiarity with PostgreSQL. References to *PostgreSQL documentation* are provided throughout this guide for features that are similar to those in Greenplum Database.

This guide provides information for system administrators responsible for administering a Greenplum Database system.

- *About the Greenplum Database Documentation Set*
- *Document Conventions*
- *Getting Support*

About the Greenplum Database Documentation Set

The Greenplum Database 4.3 documentation set consists of the following guides.

Table 1: Greenplum Database documentation set

Guide Name	Description
Greenplum Database Administrator Guide	Describes the Greenplum Database architecture and concepts such as parallel processing, and system administration and database administration tasks for Greenplum Database. System administraiton topics include configuring the server, monitoring system activity, enabling high-availability, backing up and restoring databases, and expanding the system. Database administration topics include creating databases and database objects, loading and manipulating data, writing queries, and monitoring and managing database performance.
Greenplum Database Reference Guide	Reference information for Greenplum Database systems: SQL commands, system catalogs, environment variables, character set support, datatypes, the Greenplum MapReduce specification, postGIS extension, server parameters, the gp_toolkit administrative schema, and SQL 2008 support.
Greenplum Database Utility Guide	Reference information for command-line utilities, client programs, and Oracle compatibility functions.
Greenplum Database Installation Guide	Information and instructions for installing and initializing a Greenplum Database system.
Greenplum Database Security Configuration Guide	Information about securing Greenplum Database systems.

Document Conventions

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

- *Text Conventions*
- *Command Syntax Conventions*

Text Conventions

Table 2: 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 column 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 3: 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 *https://support.pivotal.io/*.

Chapter 2

Securing the Database

The intent of security configuration is to configure the Greenplum Database server to eliminate as many security vulnerabilities as possible. This guide provides a baseline for minimum security requirements, and is supplemented by additional security documentation.

The essential security requirements fall into the following categories:

- *Authentication* covers the mechanisms that are supported and that can be used by the Greenplum database server to establish the identity of a client application.
- *Authorization* pertains to the privilege and permission models used by the database to authorize client access.
- *Auditing*, or log settings, covers the logging options available in Greenplum Database to track successful or failed user actions.
- *Data Encryption* addresses the encryption capabilities that are available for protecting data at rest and data in transit. This includes the security certifications that are relevant to the Greenplum Database.

Accessing a Kerberized Hadoop Cluster

Greenplum Database can read or write external tables in a Hadoop file system. If the Hadoop cluster is secured with Kerberos ("Kerberized"), Greenplum Database must be configured to allow external table owners to authenticate with Kerberos. See *Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster* for the steps to perform this setup.

Platform Hardening

Platform hardening involves assessing and minimizing system vulnerability by following best practices and enforcing federal security standards. Hardening the product is based on the US Department of Defense (DoD) guidelines Security Template Implementation Guides (STIG). Hardening removes unnecessary packages, disables services that are not required, sets up restrictive file and directory permissions, removes unowned files and directories, performs authentication for single-user mode, and provides options for end users to configure the package to be compliant to the latest STIGs.

Other Pivotal Security Documentation

This document may be supplemented by other Pivotal security documentation, including Hardening Guides and Security Advisories. Please consult your Pivotal representative to learn what other security documents may be applicable to your situation.

Chapter 3

Greenplum Database Ports and Protocols

Greenplum Database clients connect with TCP to the Greenplum master instance at the client connection port, 5432 by default. The listen port can be reconfigured in the `postgresql.conf` configuration file. Client connections use the PostgreSQL libpq API. The `psql` command-line interface, several Greenplum utilities, and language-specific programming APIs all either use the libpq library directly or implement the libpq protocol internally.

Each segment instance also has a client connection port, used solely by the master instance to coordinate database operations with the segments. The `gpstate -p` command, executed on the Greenplum master, lists the port assignments for the Greenplum master and the primary segments and mirrors. For example:

```
[gpadmin@mdw ~]$ gpstate -p
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-Starting gpstate with args: -p
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-local Greenplum Version:
 'postgres (Greenplum Database) 4.3.6.0 build 62994'
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-master Greenplum Version:
 'PostgreSQL 8.2.15 (Greenplum Database 4.3.6.0 build 62994) on x86_64-unknown-linux-
gnu, compiled by GCC gcc (GCC) 4.4.2 compiled on Jul 24 2015 11:35:08'
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-Obtaining Segment details from
 master...
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:--Master segment instance  /data/
master/gpseg-1  port = 5432
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:--Segment instance port
 assignments
20160126:15:40:22:028389 gpstate:mdw:gpadmin-
[INFO]:----------------------------------
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   Host   Datadir
 Port
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/primary/gpseg0
 40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/mirror/gpseg0
 50000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw2   /data/primary/gpseg1
 40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw1   /data/mirror/gpseg1
 50001
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/primary/gpseg2
 40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw4   /data/mirror/gpseg2
 50000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw4   /data/primary/gpseg3
 40000
20160126:15:40:22:028389 gpstate:mdw:gpadmin-[INFO]:-   sdw3   /data/mirror/gpseg3
 50001
```

Additional Greenplum Database network connections are created for features such as standby replication, segment mirroring, statistics collection, and data exchange between segments. Some persistent connections are established when the database starts up and other transient connections are created during operations such as query execution. Transient connections for query execution processes, data movement, and statistics collection use available ports in the range 1025 to 65535 with both TCP and UDP protocols.

Some add-on products and services that work with Greenplum Database have additional networking requirements. The following table lists ports and protocols used within the Greenplum cluster, and includes services and applications that integrate with Greenplum Database.

Table 4: Greenplum Database Ports and Protocols

Service	Protocol/Port	Description
Master SQL client connection	TCP 5432, libpq	SQL client connection port on the Greenplum master host. Supports clients using the PostgreSQL libpq API. Configurable.
Segment SQL client connection	varies, libpq	The SQL client connection port for a segment instance. Each primary and mirror segment on a host must have a unique port. Ports are assigned when the Greenplum system is initialized or expanded. The `gp_segment_configuration` system catalog records port numbers for each segment in the `port` column. Run `gpstate -p` to view the ports in use.
Segment mirroring port	varies, libpq	The port where a segment receives mirrored blocks from its primary. The port is assigned when the mirror is set up. The port number is stored in the `gp_segment_configuration` system catalog in the `mirror_port` column.
Greenplum Database Interconnect	UDP 1025-65535, dynamically allocated	The Interconnect transports database tuples between Greenplum segments during query execution.
Standby master client listener	TCP 5432, libpq	SQL client connection port on the standby master host. Usually the same as the master client connection port. Configure with the `gpinitstandby` utility `-P` option.
Standby master replicator	TCP 1025-65535, gpsyncmaster	The `gpsyncmaster` process on the master host establishes a connection to the secondary master host to replicate the master's log to the standby master.
Greenplum Control Center (GPCC)	TCP 28080, HTTP/ HTTPS	Default listen port for the GPCC console web server, which is usually installed on the master host. Configured in the `lighttpd.conf` file in the GPCC instance.
Greenplum database file load and transfer utilities: gpfdist, gpload, gptransfer	TCP 8080, HTTP TCP 9000, HTTPS	The gpfdist file serving utility can run on Greenplum hosts or external hosts. Specify the connection port with the `-p` option when starting the server. The gpload and gptransfer utilities run one or more instances of gpfdist with ports or port ranges specified in a configuration file.
GPCC agents	TCP 8888	Connection port for GPCC agents executing on each Greenplum host. Configure by setting the `gpperfmon_port` configuration variable in `postgresql.conf` on master and segment hosts.
Backup completion notification	TCP 25, TCP 587, SMTP	The gpcrondump backup utility can optionally send email to a list of email addresses at completion of a backup. The SMTP service must be enabled on the Greenplum master host.

Service	Protocol/Port	Description
Greenplum Database secure shell (SSH): gpssh, gpscp, gpssh-exkeys, gppkg, gpseginstall	TCP 22, SSH	Many Greenplum utilities use scp and ssh to transfer files between hosts and manage the Greenplum system within the cluster.
gpsnmpd (deprecated)	TCP 161, SNMP	The gpsnmpd utility daemon can run as a subagent of the system SNMP service, or standalone. When run standalone, an interface address and listen port are specified at startup time.
gphdfs	TCP 8020	The gphdfs protocol allows access to data in a Hadoop file system via Greenplum external tables. The URL in the `LOCATION` clause of the `CREATE EXTERNAL TABLE` command specifies the host address and port number for the Hadoop namenode service.
Greenplum Workload Manager (GP-WLM)	TCP 4369, epmd	Erlang port mapper (epmd) allows nodes in the cluster to resolve node names.
	TCP 25672	rabbitmq clustering
	TCP 7777	rabbitmq main port
EMC Data Domain and DD Boost	TCP/UDP 111, NFS portmapper	Used to assign a random port for the mountd service used by NFS and DD Boost. The mountd service port can be statically assigned on the Data Domain server.
	TCP 2052	Main port used by NFS mountd. This port can be set on the Data Domain system using the `nfs set mountd-port` command .
	TCP 2049, NFS	Main port used by NFS. This port can be configured using the `nfs set server-port` command on the Data Domain server.
	TCP 2051, replication	Used when replication is configured on the Data Domain system. This port can be configured using the `replication modify` command on the Data Domain server.
Symantec NetBackup	TCP/UDB 1556, veritas-pbx	The Symantec NetBackup client network port.
	TCP 13724, vnetd	Symantec NetBackup vnetd communication port.

Service	Protocol/Port	Description
Pgbouncer connection pooler	TCP, libpq	The pgbouncer connection pooler runs between libpq clients and Greenplum (or PostgreSQL) databases. It can be run on the Greenplum master host, but running it on a host outside of the Greenplum cluster is recommended. When it runs on a separate host, pgbouncer can act as a warm standby mechanism for the Greenplum master host, switching to the Greenplum standby host without requiring clients to reconfigure. Set the client connection port and the Greenplum master host address and port in the `pgbouncer.ini` configuration file.
stunnel SSL proxy	TCP, ssh, libpq	A stunnel SSL proxy can be used to add SSL support for database clients accessing the database through a pgbouncer connection pool. A secure tunnel can be set up by setting up stunnel on the client and the pgbouncer host. Newer versions of stunnel that support encrypted libpq connections only require stunnel on the pgbouncer host. The stunnel proxy's connection ports and the pgbouncer host and port are specified in the `stunnel.conf` configuration file.

Chapter 4

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 section explains how to configure client connections and authentication to Greenplum Database.

- *Allowing Connections to Greenplum Database*
- *Editing the pg_hba.conf File*
- *Authentication Methods*
- *Limiting Concurrent Connections*
- *Encrypting Client/Server Connections*

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 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 a # 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 this format:

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

Each UNIX-domain socket access record is in this format:

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

The meaning of the `pg_hba_conf` fields is as follows:

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* for more details.

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.

To edit `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
```

Authentication Methods

- *Basic Authentication*
- *Kerberos Authentication*
- *LDAP Authentication*
- *SSL Client Authentication*
- *PAM Based Authentication*
- *Radius Authentication*

Basic Authentication

The following basic authentication methods are supported:

Password or MD5

Requires clients to provide a password, one of either:

- Md5 – 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" in the *Greenplum Database Administrator Guide* 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 operating system user name. You should only use this for local connections.

Following are some sample `pg_hba.conf` basic authentication entries:

```
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).

The format for Kerberos authentication in the `pg_hba.conf` file is:

```
Servicename/hostname@realm
```

The following options may be added to the entry:

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.

Following is an example `pg_hba.conf` entry for Kerberos:

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

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-sensitivity. 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.

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`).

Specify the following parameter `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"
```

Following are sample `pg_hba.conf` file entries for LDAP authentication:

```
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 SSL handshake to the requested database user name. 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***

  mapping.

  This is specified in the `pg_ident.conf`, or in the file specified in the `ident_file` server setting.

  Following are sample `pg_hba.conf` entries for SSL client authentication:

```
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

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.

To create a self-signed certificate for the server:

1. Enter the following `openssl` command:

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

2. Enter the requested information at the prompts.

   Make sure you enter the local host name for the Common Name. The challenge password can be left blank.
3. The program generates a key that is passphrase-protected; it does not accept a passphrase that is less than four characters long. To remove the passphrase (and you must if you want automatic start-up of the server), run the following command:

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

4. 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.

5. Finally, run the following command:

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

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

Configuring postgresql.conf for SSL Authentication

The following Server settings need to be specified 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

SSL 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. The default is `~/.postgresql/postgresql.crt`.

Sslkey

The secret key used for the client certificate. The default is `~/.postgresql/postgresql.key`.

Sslrootcert

The name of a file containing SSL Certificate Authority certificate(s). The default is `~/.postgresql/root.crt`.

Sslcrl

The name of the SSL certificate revocation list. The default is `~/.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" (Pluggable Authentication Modules) is used to validate username/password pairs, similar to basic authentication. PAM authentication 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.

Following are sample `pg_hba.conf` entries for PAM client authentication:

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

Radius Authentication

RADIUS (Remote Authentication Dial In User Service) 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 the Network Access Server (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: RADIUS accounting is not supported.

RADIUS authentication 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.

RADIUS Authentication Options

Radiusserver

The ame of the RADIUS server.

Radiussecret

The RADIUS shared secret.

Radiusport

The port to connect to on the RADIUS server.

Radiusidentifier

NAS identifier in RADIUS requests.

Following are sample `pg_hba.conf` entries for RADIUS client authentication:

```
ostssl  all all 0.0.0.0/0 radius radiusserver=servername radiussecret=sharedsecret
```

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.

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
```

> Note: 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`.

To change the number of allowed connections:

1. Stop your Greenplum Database system:

   ```
   $ gpstop
   ```

2. On the 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 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
   ```

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 *Encrypting gpfdist Connections*.

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. See *Creating a Self-Signed Certificate* for steps to create a self-signed certificate.

Chapter 5

Configuring Database Authorization

Authorization governs access to Greenplum Database database objects.

Access Permissions and Roles

Greenplum Database manages database access permissions using *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 verifies their role and access privileges. The master then issues out commands to the segment instances behind the scenes using the currently logged in role.

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

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`. To create more roles you first must connect as this initial role.

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:

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

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;
```

Using SSH-256 Encryption

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.

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 Standard 140-2, 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 section 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 *Configuring the SSL Client Connection* 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.

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.

Setting Encryption Method System-wide

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

1. Log in to 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
   ```

Setting Encryption Method for an Individual Session

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

1. Log in to 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'
   ```

   or:

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

3. Verify the setting:

   ```
   # show password_hash_algorithm;
   ```

   You will see:

   ```
   SHA-256
   ```

   or:

   ```
   SHA-256-FIPS
   ```

Following is an example of how the new setting works:

1. Log 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. Log in to the database as the 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.

   Password hashes are stored in `pg_authid.rolpasswod`.
8. Log in as the super user.
9. Execute the following query:

   ```
   # SELECT rolpassword FROM pg_authid WHERE rolname = 'testdb';
   ```

```
    Rolpassword
    -----------
    sha256<64 hexidecimal characters>
```

Restricting Access by Time

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.

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

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.

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

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'.
```

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

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'
        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:

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

The correct specification uses two DENY clauses, as follows:

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

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 in the *Greenplum Database Reference 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'
  ...
```

Excample 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.

Chapter 6

Greenplum Command Center Security

Greenplum Command Center (GPCC) is a web-based application for monitoring and managing Greenplum clusters. GPCC works with data collected by agents running on the segment hosts and saved to the gpperfmon database. The gpperfmon database is created by running the `gpperfmon_install` utility, which also creates the `gpmon` database role that GPCC uses to access the gpperfmon database.

The gpmon User

The `gpperfmon_install` utility creates the `gpmon` database role and adds the role to the `pg_hba.conf` file with the following entries:

```
local    gpperfmon   gpmon                          md5
host     all         gpmon         127.0.0.1/28     md5
```

These entries allow `gpmon` to establish a local socket connection to the gpperfmon database and a TCP/IP connection to any database.

The `gpmon` database role is a superuser. In a secure or production environment, it may be desirable to restrict the `gpmon` user to just the gpperfmon database. Do this by editing the `gpmon` host entry in the `pg_hba.conf` file and changing `all` in the database field to `gpperfmon`:

```
local   gpperfmon   gpmon                         md5
host    gpperfmon   gpmon    127.0.0.1/28         md5
```

The password used to authenticate the `gpmon` user is set by the `gpperfmon_install` utility and is stored in the `gpadmin` home directory in the `~/.pgpass` file. The `~/.pgpass` file must be owned by the `gpadmin` user and RW-accessible only by the `gpadmin` user. To change the `gpmon` password, use the `ALTER ROLE` command to change the password in the database, change the password in the `~/.pgpass` file, and then restart GPCC with the `gpcmdr --restart` *`instance_name`* command.

> Note: The GPCC web server can be configured to encrypt connections with SSL. Two-way authentication with public keys can also be enabled for GPCC users. However, the `gpmon` user always uses md5 authentication with the password saved in the `~/.pgpass` file.

GPCC does not allow logins from any role configured with trust authentication, including the `gpadmin` user.

The `gpmon` user can log in to the Command Center Console and has access to all of the application's features. You can allow other database roles access to GPCC so that you can secure the `gpmon` user and restrict other users' access to GPCC features. Setting up other GPCC users is described in the next section.

Greenplum Command Center Users

GPCC has three types of users:

- *Regular* users can view their own queries in GPCC, but do not have access to administrative tasks in GPCC.
- *Operators* have access to more functionality in the Command Center Console than regular users. They can view and cancel all queries and they have limited access to administrative tasks.
- *Superusers* are Greenplum Database superusers. They can use all GPCC features, including viewing information for all database queries, system metrics, and performing potentially disruptive administrative tasks, such as stopping or restarting the database system.

To log in to the GPCC web application, a user must be allowed access to the gpperfmon database in `pg_hba.conf`. For example, to make `user1` a regular GPCC user, edit the `pg_hba.conf` file and either add or edit a line for the user so that the gpperfmon database is included in the database field. For example:

```
host     gpperfmon,accounts   user1     127.0.0.1/28    md5
```

To designate a user as an operator, create the `gpcc_operator` role, if it does not already exist:

```
=# CREATE ROLE gpcc_operator;
```

Then grant the `gpcc_operator` role to the user:

```
=# GRANT gpcc_operator TO user;
```

You can also grant `gpcc_operator` to a group role to make all members of the group GPCC operators.

See the `gpperfmon_install` reference in *Greenplum Database Utility Guide* and the *Greenplum Command Center Administrator Guide* for more information about managing the `gpperfmon` database and GPCC security.

Enabling SSL for Greenplum Command Center

The GPCC web server can be configured to support SSL so that client connections are encrypted. A server certificate can be generated when the Command Center instance is created or you can supply an existing certificate.

Two-way authentication with public key encryption can also be enabled for GPCC. See the *Greenplum Command Center Administration Guide* for instructions.

Chapter 7

Auditing

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, and all connection attempts and disconnections. 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. Greenplum Database records the username for all auditable events, when the event is initiated by a subject outside the Greenplum Database.

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. 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 in the *Greenplum Database Reference 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.

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`).

#	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")

#	Field Name	Data Type	Description
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

#	Field Name	Data Type	Description
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
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

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
```

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

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.

Field Name	Value Range	Default	Description
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.
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.

Field Name	Value Range	Default	Description
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.

Chapter 8

Encrypting Data and Database Connections

Encryption can be used to protect data in transit over the network between the database and clients, and data at rest in the database.

- Connections between clients and the master database can be encrypted with SSL. This is enabled with the `ssl` server configuration parameter, which is `off` by default. Setting the `ssl` parameter to `on` allows client communications with the master to be encrypted. The master database must be set up for SSL. See *OpenSSL Configuration* for more about encrypting client connections with SSL.
- Greenplum Database 4.2.1 and above allow SSL encryption of data in transit between the Greenplum parallel file distribution server, `gpfdist`, and segment hosts. See *Encrypting gpfdist Connections* for more information.
- The `pgcrypto` package of encryption/decryptions functions protect data at rest in the database. Encryption at the column level protects sensitive information, such as social security numbers or credit card numbers. See *Encrypting Data at Rest with pgcrypto* for more information.

Encrypting gpfdist Connections

The `gpfdists` protocol is a secure version of the `gpfdist` protocol that securely identifies the file server and the Greenplum Database and encrypts the communications between them. Using `gpfdists` protects against eavesdropping and man-in-the-middle attacks.

The `gpfdists` protocol implements client/server SSL security with the following notable features:

- Client certificates are required.
- Multilingual 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) or for the Greenplum Database (client.key).
- It is the user's responsibility to issue certificates that are appropriate for the operating system in use. Generally, converting certificates to the required format is supported, for example using the SSL Converter at *https://www.sslshopper.com/ssl-converter.html*.

A `gpfdist` server started with the `--ssl` option can only communicate with the `gpfdists` protocol. A `gpfdist` server started without the `--ssl` option can only communicate with the `gpfdist` protocol. For more detail about `gpfdist` refer to the *Greenplum Database Administrator Guide*.

There are two ways to enable the `gpfdists` protocol:

- 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.

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`

> 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 `gpload` with SSL you specify the location of the server certificates in the YAML control file. When using `gpfdist` with SSL, you specify the location of the server certificates with the --ssl option.

The following example shows how to securely load data into an external table. The example creates a readable external table named `ext_expenses` from all files with the `txt` extension, using the `gpfdists` protocol. The files are formatted with a pipe (`|`) as the column delimiter and an empty space as null.

1. Run `gpfdist` with the `--ssl` option on the segment hosts.
2. Log into the database and 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 ' ') ;
```

Encrypting Data at Rest with pgcrypto

The pgcrypto package for Greenplum Database provides functions for encrypting data at rest in the database. Administrators can encrypt columns with sensitive information, such as social security numbers or credit card numbers, to provide an extra layer of protection. Database data stored in encrypted form cannot be read by users who do not have the encryption key, and the data cannot be read directly from disk.

pgcrypto allows PGP encryption using symmetric and asymmetric encryption. Symmetric encryption encrypts and decrypts data using the same key and is faster than asymmetric encryption. It is the preferred method in an environment where exchanging secret keys is not an issue. With asymmetric encryption, a public key is used to encrypt data and a private key is used to decrypt data. This is slower then symmetric encryption and it requires a stronger key.

Using pgcrypto always comes at the cost of performance and maintainability. It is important to use encryption only with the data that requires it. Also, keep in mind that you cannot search encrypted data by indexing the data.

Before you implement in-database encryption, consider the following PGP limitations.

- No support for signing. That also means that it is not checked whether the encryption sub-key belongs to the master key.
- No support for encryption key as master key. This practice is generally discouraged, so this limitation should not be a problem.
- No support for several subkeys. This may seem like a problem, as this is common practice. On the other hand, you should not use your regular GPG/PGP keys with pgcrypto, but create new ones, as the usage scenario is rather different.

Greenplum Database is compiled with zlib by default; this allows PGP encryption functions to compress data before encrypting. When compiled with OpenSSL, more algorithms will be available.

Because pgcrypto functions run inside the database server, 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 and you should trust both the system and database administrators.

The pgcrypto package is not installed by default with Greenplum Database. You can download a pgcrypto package from *Pivotal Network* and use the Greenplum Package Manager (`gppkg`) to install `pgcrypto` across your cluster.

pgcrypto configures itself according to the findings of the main PostgreSQL configure script.

When compiled with `zlib`, pgcrypto encryption functions are able to compress data before encrypting.

You can enable support for Federal Information Processing Standards (FIPS) 140-2 in pgcrypto. FIPS 140-2 requires pgcrypto package version 1.2. The Greenplum Database `pgcrypto.fips` server configuration parameter controls the FIPS 140-2 support in pgcrypto. See "Server Configuration Parameters" in the *Greenplum Database Reference Guide*.

Pgcrypto has various levels of encryption ranging from basic to advanced built-in functions. The following table shows the supported encryption algorithms.

Table 5: Pgcrypto Supported Encryption Functions

Value Functionality	Built-in	With OpenSSL	OpenSSL with FIPS 140-2
MD5	yes	yes	no
SHA1	yes	yes	no

Value Functionality	Built-in	With OpenSSL	OpenSSL with FIPS 140-2
SHA224/256/384/512	yes	yes [1]	yes
Other digest algorithms	no	yes [2]	no
Blowfish	yes	yes	no
AES	yes	yes[3]	yes
DES/3DES/CAST5	no	yes	yes[4]
Raw Encryption	yes	yes	yes
PGP Symmetric-Key	yes	yes	yes
PGP Public Key	yes	yes	yes

Creating PGP Keys

To use PGP asymmetric encryption in Greenplum Database, you must first create public and private keys and install them.

This section assumes you are installing Greenplum Database on a Linux machine with the Gnu Privacy Guard (`gpg`) command line tool. Pivotal recommends using the latest version of GPG to create keys. Download and install Gnu Privacy Guard (GPG) for your operating system from *https://www.gnupg.org/download/*. On the GnuPG website you will find installers for popular Linux distributions and links for Windows and Mac OS X installers.

1. As root, execute the following command and choose option 1 from the menu:

```
# gpg --gen-key
gpg (GnuPG) 2.0.14; Copyright (C) 2009 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/root/.gnupg' created
gpg: new configuration file `/root/.gnupg/gpg.conf' created
gpg: WARNING: options in `/root/.gnupg/gpg.conf' are not yet active during this
 run
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
 (1) RSA and RSA (default)
 (2) DSA and Elgamal
 (3) DSA (sign only)
 (4) RSA (sign only)
Your selection? 1
```

2. Respond to the prompts and follow the instructions, as shown in this example:

```
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) Press enter to accept default key size
Requested keysize is 2048 bits
Please specify how long the key should be valid.
 0 = key does not expire
 <n> = key expires in n days
```

[1] SHA2 algorithms were added to OpenSSL in version 0.9.8. For older versions, pgcrypto will use built-in code.

[2] Any digest algorithm OpenSSL supports is automatically picked up. This is not possible with ciphers, which need to be supported explicitly.

[3] AES is included in OpenSSL since version 0.9.7. For older versions, pgcrypto will use built-in code.

[4] 3DES is supported, DES and CAST5 are not

```
 <n>w = key expires in n weeks
 <n>m = key expires in n months
 <n>y = key expires in n years
 Key is valid for? (0) 365
Key expires at Wed 13 Jan 2016 10:35:39 AM PST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: John Doe
Email address: jdoe@email.com
Comment:
You selected this USER-ID:
 "John Doe <jdoe@email.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.
(For this demo the passphrase is blank.)
can't connect to `/root/.gnupg/S.gpg-agent': No such file or directory
You don't want a passphrase - this is probably a *bad* idea!
I will do it anyway.  You can change your passphrase at any time,
using this program with the option "--edit-key".

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 2027CC30 marked as ultimately trusted
public and secret key created and signed.

gpg:  checking the trustdbgpg:
      3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg:  depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg:  next trustdb check due at 2016-01-13
pub   2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
      Key fingerprint = 7EDA 6AD0 F5E0 400F 4D45   3259 077D 725E 2027 CC30
uid                  John Doe <jdoe@email.com>
sub   2048R/4FD2EFBB 2015-01-13 [expires: 2016-01-13]
```

3. List the PGP keys by entering the following command:

```
gpg --list-secret-keys
/root/.gnupg/secring.gpg
------------------------
sec   2048R/2027CC30 2015-01-13 [expires: 2016-01-13]
uid                  John Doe <jdoe@email.com>
ssb   2048R/4FD2EFBB 2015-01-13
```

   2027CC30 is the public key and will be used to *encrypt* data in the database. 4FD2EFBB is the private (secret) key and will be used to *decrypt* data.
4. Export the keys using the following commands:

```
# gpg -a --export 4FD2EFBB > public.key
# gpg -a --export-secret-keys 2027CC30 > secret.key
```

See the *pgcrypto* documentation for for more information about PGP encryption functions.

Encrypting Data in Tables using PGP

This section shows how to encrypt data inserted into a column using the PGP keys you generated.

1. Dump the contents of the `public.key` file and then copy it to the clipboard:

```
# cat public.key
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----
```

2. Create a table called `userssn` and insert some sensitive data, social security numbers for Bob and Alice, in this example. Paste the public.key contents after "dearmor(".

```
CREATE TABLE userssn( ssn_id SERIAL PRIMARY KEY,
    username varchar(100), ssn bytea);

INSERT INTO userssn(username, ssn)
SELECT robotccs.username, pgp_pub_encrypt(robotccs.ssn, keys.pubkey) AS ssn
FROM (
        VALUES ('Alice', '123-45-6788'), ('Bob', '123-45-6799'))
            AS robotccs(username, ssn)
CROSS JOIN  (SELECT  dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
```

```
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----' AS pubkey) AS keys;
```

3. Verify that the `ssn` column is encrypted.

```
test_db=# select * from userssn;
ssn_id   | 1
username | Alice
ssn      | \301\300L\003\235M%_O
\322\357\273\001\010\000\272\227\010\341\216\360\217C\020\261)_\367
[\227\034\313:C\354d<\337\006Q\351('\2330\031lX\263Qf\341\262\200\3015\235\036AK
\242fL+\315g\322
7u\270*\304\361\355\220\021\330"\200%\264\274}R\213\377\363\235\366\030\023)\364!
\331\303\237t\277=
f \015\004\242\231\263\225%\032\271a\001\035\277\021\375X\232\304\305/
\340\334\0131\325\344[~\362\0
37-\251\336\303\340\377_\011\275\301/MY\334\343\245\244\372y\257S
\374\230\346\277\373W\346\230\276\
017fi\226Q\307\012\326\3646\000\326\005:E\364W\252=zz\010(:\343Y\237\257iqU
\0326\350=v0\362\327\350\
315G^\027:K_9\254\362\354\215<\001\304\357\331\355\323,\302\213Fe
\265\315\232\367\254\245%(\\\373
4\254\230\331\356\006B\257\333\326H\022\013\353\216F?\023\220\370\035vH5/\227\344b
\322\227\026\362=\
42\033\322<\001}\243\224;)\030zqX\214\340\221\035\275U\345\327\214\032\351\223c
\2442\345\304K\016\
011\214\307\227\237\270\026`R\205\205a~1\263\236[\037C
\260\031\205\374\245\317\033k|\366\253\037
---------
+-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------
ssn_id   | 2
username | Bob
ssn      | \301\300L\003\235M%_O\322\357\273\001\007\377t>\345\343,
\200\256\272\300\012\033M4\265\032L
L[v\262k\244\2435\264\232B\357\370d9\375\011\002\327\235<\246\210b\030\012\337@
\226Z\361\246\032\00
7`\012c\353]\355d7\360T\335\314\367\370;X\371\350*\231\212\260B
\010#RQ0\223\253c7\0132b\355\242\233\34
1\000\370\370\366\013\022\357\005i\202~\005\\z\301o\012\230Z\014\362\244\324&\243g
\351\362\325\375
\213\032\226$\2751\256XR\346k\266\030\234\267\201vUh\004\250\337A\231\223u
\247\366/i\022\275\276\350\2
20\316\306|\203+\010\261;\232\254tp\255\243\261\373Rq;\316w\357\006\207\374U
\333\365\365\245hg\031\005
\322\347ea\220\015l\212g\337\264\336b\263\004\311\210.4\340G+\221\274D
\035\375\2216\241`\346a0\273wE\2
12\342y^\202\262|A7\202t\240\333p\345G\373\253\243oCO
\011\360\247\211\014\024{\272\271\322<\001\267
\347\240\005\213\0078\036\210\307$\317\322\311\222\035\354\006<
\266\264\004\376\251q\256\220(+\030\
3270\013c\327\272\212%\363\033\252\322\337\354\276\225\232\201\212^\304\210\2269@
\3230\370{
```

4. Extract the public.key ID from the database:

```
SELECT pgp_key_id(dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)
```

```
mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAG0HHRlc3Qga2V5IDx0ZXN0
a2V5QGVtYWlsLmNvbT6JAT4EEwECACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUI
AgkKCwQWAgMBAh4BAheAAAoJEAd9cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY
7BfrvU52yk+PPZYoes9UpdL3CMRk8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8
JRoC3ooezTkmCBW8I1bU0qGetzVxopdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7
TAoccXLbyuZh9Rf5vLoQdKzcCyOHh5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0
DGoUAOanjDZ3KE8Qp7V74fhG1EZVzHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4
eTUXPSnwPi46qoAp9UQogsfGyB1XDOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBe5
AQ0EVLVl/QEIANabFdQ+8QMCADOipM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7
Zro2us99GlARqLWd8EqJcl/xmfcJiZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjw
gCWG/ZLu4wzxOFFzDkiPv9RDw6e5MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rE
iDDCWU4T0lhv3hSKCpke6LcwsX+7liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2J
bVLz3lLLouHRgpCzla+PzzbEKs16jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hO
v/8yAnkcAmowZrIBlyFg2KBzhunYmN2YvkUAEQEAAYkBJQQYAQIADwUCVLVl/QIb
DAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3rW9izrO48
WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPHQNPSvz62
WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8Hkk8qb5x/
HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlGWE8pvgEx
/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojDyC65KJci
Pv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=XZ8J
-----END PGP PUBLIC KEY BLOCK-----'));

pgp_key_id | 9D4D255F4FD2EFBB
```

This shows that the PGP key ID used to encrypt the `ssn` column is 9D4D255F4FD2EFBB. It is recommended to perform this step whenever a new key is created and then store the ID for tracking.

You can use this key to see which key pair was used to encrypt the data:

```
SELECT username, pgp_key_id(ssn) As key_used
FROM userssn;
username | Bob
key_used | 9D4D255F4FD2EFBB
---------+-----------------
username | Alice
key_used | 9D4D255F4FD2EFBB
```

> Note: Different keys may have the same ID. This is rare, but is a normal event. The client application should try to decrypt with each one to see which fits — like handling `ANYKEY`. See *pgp_key_id()* in the pgcrypto documentation.

5. Decrypt the data using the private key.

```
SELECT username, pgp_pub_decrypt(ssn, keys.privkey)
                 AS decrypted_ssn FROM userssn
                 CROSS JOIN
                 (SELECT dearmor('-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: GnuPG v2.0.14 (GNU/Linux)

lQOYBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6PqKyA05jtWiXZTh
2His1ojSP6LI0cSkIqMU9LAlncecZhRIhBhuVgKlGSgd9texg2nnSL9Admqik/yX
R5syVKG+qcdWuvyZg9oOOmeyjhc3n+kkbRTEMuM3flbMs8shOwzMvstCUVmuHU/V
vG5rJAe8PuYDSJCJ74I6w7SOH3RiRIc7IfL6xYddV42l3ctd44bl8/i71hq2UyN2
/Hbsjii2ymg7ttw3jsWAx2gP9nssDgoy8QDy/o9nNqC8EGlig96ZFnFnE6Pwbhn+
ic8MD0lK5/GAlR6Hc0ZIHf8KEcavruQlikjnABEBAAEAB/wNfjjvP1brRfjjIm/j
XwUNm+sI4v2Ur7qZC94VTukPGf67lvqcYZJuqXxvZrZ8bl6mvl65xEUiZYy7BNA8
fe0PaM4Wy+Xr94Cz2bPbWgawnRNN3GAQy4rlBTrvqQWy+kmpbd87iTjwZidZNNmx
02iSzraq41Rt0Zx21Jh4rkpF67ftmzOH0vlrS0bWOvHUeMY7tCwmdPe9HbQeDlPr
n9CllUqBn4/acTtCClWAjREZn0zXAsNixtTIPC1V+9nO9YmecMkVwNfIPkIhymAM
OPFnuZ/Dz1rCRHjNHb5j6ZyUM5zDqUVnnezktxqrOENSxm0gfMGcpxHQogUMzb7c
6UyBBADSCXHPfo/VPVtMm5p1yGrNOR2jR2rUj9+poZzD2gjkt5G/xIKRlkB4uoQl
emu27wr9dVEX7ms0nvDq58iutbQ4d0JIDlcHMeSRQZluErblB75Vj3HtImblPjpn
4Jx6SWRXPUJPGXGI87u0UoBH0Lwij7M2PW7l1ao+MLEA9jAjQwQA+sr9BKPL4Ya2
```

```
r5nE72gsbCCLowkC0rdldf1RGtobwYDMpmYZhOaRKjkOTMG6rCXJxrf6LqiN8w/L
/gNziTmch35MCq/MZzA/bN4VMPyeIlwzxVZkJLsQ7yyqX/A7ac7B7DH0KfXciEXW
MSOAJhMmklW1Q1RRNw3cnYi8w3q7X40EAL/w54FVvvPqp3+sCd86SAAapM4UO2R3
tIsuNVemMWdgNXwvK8AJsz7VreVU5yZ4B8hvCuQj1C7geaN/LXhiT8foRsJC5o71
Bf+iHC/VNEv4k4uDb4lOgnHJYYyifB1wC+nn/EnXCZYQINMia1a4M6Vqc/RIfTH4
nwkZt/89LsAiR/20HHRlc3Qga2V5IDx0ZXN0a2V5QGVtYWlsLmNvbT6JAT4EEwEC
ACgFAlS1Zf0CGwMFCQHhM4AGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJEAd9
cl4gJ8wwbfwH/3VyVsPkQl1owRJNxvXGt1bY7BfrvU52yk+PPZYoes9UpdL3CMRk
8gAM9bx5Sk08q2UXSZLC6fFOpEW4uWgmGYf8JRoC3ooezTkmCBW8I1bU0qGetzVx
opdXLuPGCE7hVWQe9HcSntiTLxGov1mJAwO7TAoccXLbyuZh9Rf5vLoQdKzcCyOH
h5IqXaQOT100TeFeEpb9TIiwcntg3WCSU5P0DGoUAOanjDZ3KE8Qp7V74fhG1EZV
zHb8FajR62CXSHFKqpBgiNxnTOk45NbXADn4eTUXPSnwPi46qoAp9UQogsfGyB1X
DOTB2UOqhutAMECaM7VtpePv79i0Z/NfnBedA5gEVLVl/QEIANabFdQ+8QMCADOi
pM1bF/JrQt3zUoc4BTqICaxdyzAfz0tUSf/7Zro2us99GlARqLWd8EqJcl/xmfcJ
iZyUam6ZAzzFXCgnH5Y1sdtMTJZdLp5WeOjwgCWG/ZLu4wzxOFFzDkiPv9RDw6e5
MNLtJrSp4hS5o2apKdbO4Ex83O4mJYnav/rEiDDCWU4T0lhv3hSKCpke6LcwsX+7
liozp+aNmP0Ypwfi4hR3UUMP70+V1beFqW2JbVLz3lLLouHRgpCzla+PzzbEKs16
jq77vG9kqZTCIzXoWaLljuitRlfJkO3vQ9hOv/8yAnkcAmowZrIBlyFg2KBzhunY
mN2YvkUAEQEAAQAH/A7r4hDrnmzX3QU6FAzePlRB7niJtE2IEN8AufF05Q2PzKU/
c1S72WjtqMAIAgYasDkOhfhcxanTneGuFVYggKT3eSDm1RFKpRjX22m0zKdwy67B
Mu95V2Oklul6OCm8dO6+2fmkGxGqc4ZsKy+jQxtxK3HG9YxMC0dvA2v2C5N4TWi3
Utc7zh//k6IbmaLd7F1d7DXt7Hn2Qsmo8I1rtgPE8grDToomTnRUodToyejEqKyI
ORwsp8n8g2CSFaXSrEyU6HbFYXSxZealhQJGYLFOZdR0MzVtZQCn/7n+IHjupndC
Nd2a8DVx3yQS3dAmvLzhFacZdjXi31wvj0moFOkEAOCz1E63SKNNksniQ11lRMJp
gaov6Ux/zGLMstwTzNouI+Kr8/db0GlSAy1Z3UoAB4tFQXEApoX9A4AJ2KqQjqOX
cZVULenfDZaxrbb9Lid7ZnTDXKVyGTWDF7ZHavHJ4981mCW17lU11zHBB9xMlx6p
dhFvb0gdy0jSLaFMFr/JBAD0fz3RrhP7e6Xll2zdBqGthjC5S/IoKwwBgw6ri2yx
LoxqBr2pl9PotJJ/JUMPhD/LxuTcOZtYjy8PKgm5jhnBDq3Ss0kNKAY1f5EkZG9a
6I4iAX/NekqSyF+OgBfC9aCgS5RG8hYoOCbp8na5R3bgiuS8IzmVmm5OhZ4MDEwg
nQP7BzmR0p5BahpZ8r3Ada7FcK+0ZLLRdLmOYF/yUrZ53SoYCZRzU/GmtQ7LkXBh
Gjqied9Bs1MHdNUolq7GaexcjZmOWHEf6w9+9M4+vxtQq1nkIWqtaphewEmd5/nf
EP3sIY0EAE3mmiLmHLqBju+UJKMNwFNeyMTqgcg50ISH8J9FRIkBJQQYAQIADwUC
VLVl/QIbDAUJAeEzgAAKCRAHfXJeICfMMOHYCACFhInZA9uAM3TC44l+MrgMUJ3r
W9izrO48WrdTsxR8WkSNbIxJoWnYxYuLyPb/shc9k65huw2SSDkj//0fRrI61FPH
QNPSvz62WH+N2lasoUaoJjb2kQGhLOnFbJuevkyBylRz+hI/+8rJKcZOjQkmmK8H
kk8qb5x/HMUc55H0g2qQAY0BpnJHgOOQ45Q6pk3G2/7Dbek5WJ6K1wUrFy51sNlG
WE8pvgEx/UUZB+dYqCwtvX0nnBu1KNCmk2AkEcFK3YoliCxomdOxhFOv9AKjjojD
yC65KJciPv2MikPS2fKOAg1R3LpMa8zDEtl4w3vckPQNrQNnYuUtfj6ZoCxv
=fa+6
-----END PGP PRIVATE KEY BLOCK-----') AS privkey) AS keys;

username | decrypted_ssn
----------+---------------
 Alice    | 123-45-6788
 Bob      | 123-45-6799
(2 rows)
```

If you created a key with passphrase, you may have to enter it here. However for the purpose of this example, the passphrase is blank.

Key Management

Whether you are using symmetric (single private key) or asymmetric (public and private key) cryptography, it is important to store the master or private key securely. There are many options for storing encryption keys, for example, on a file system, key vault, encrypted USB, trusted platform module (TPM), or hardware security module (HSM).

Consider the following questions when planning for key management:

- Where will the keys be stored?
- When should keys expire?
- How are keys protected?
- How are keys accessed?
- How can keys be recovered and revoked?

The Open Web Application Security Project (OWASP) provides a very comprehensive *guide to securing encryption keys*.

Chapter 9

Configuring IPsec for Greenplum Database

This topic describes how to set up and configure Internet Protocol Security (IPsec) for a Greenplum Database cluster.

- *IPsec Overview*
- *Installing Openswan*
- *Configuring Openswan Connections*

IPsec Overview

Internet Protocol Security (IPsec) is a suite of protocols that authenticate and encrypt communications at the IP network level (OSI layer 3). Using IPsec can help to prevent network attacks, such as packet sniffing, altering network packets, identity spoofing, and man-in-the-middle attacks in the Greenplum Database system.

When IPsec is enabled for Greenplum Database, a virtual private network (VPN), or tunnel, is established between every pair of hosts in the cluster and every packet exchanged between them is encrypted and sent through the tunnel. If you have *n* hosts in the cluster, *n*(*n*-1)/2 VPNs are needed to connect each host to every other host. You may choose to add other hosts on the network, for example ETL servers, to the IPsec configuration.

Encrypting IP traffic has a cost in network performance. To ensure suitable network bandwidth is available after IPsec configuration, use the Greenplum Database `gpcheckperf` utility. See the Greenplum Database Utility Guide for help with `gpcheckperf`. If network bandwidth is insufficient for performance and database workloads you may need to tune the configuration or use a higher bandwidth network medium.

This section describes how to set up and configure IPsec for a Greenplum cluster on Red Hat or CentOS hosts using Openswan, a popular IPsec implementation for Linux. Openswan provides user tools to enable IPsec on Linux. It uses the Internet Key Exchange (IKE) protocol and X.509 certificates to handshake and exchange session keys, and uses the netlink API to interface with the IPsec support built into the Linux kernel.

The IKE protocol allows two peers to negotiate the authentication and encryption algorithms the tunnels will use. The negotiation occurs in two phases. During phase 1, the peers perform a Diffie-Hellman key exchange to establish a secure, encrypted channel. Phase 1 must successfully complete before phase 2 begins.

During phase 2, the peers negotiate the authentication and encryption algorithms to be used with the IPsec tunnels. The result of the phase 2 negotiation is a *security association* (SA). It contains the source, the destination, and an instruction. The Linux kernel uses the SA to set up the connection.

The peers can authenticate each other using one of the following methods:

- RSA public key encryption. Each host has a public and private key. Public keys are distributed to all hosts in the cluster so that any host can authenticate any other host.
- Pre-shared keys. This method is easiest to configure, but is not as secure as using RSA public key encryption.
- X.509 certificates. A certificate is issued for each host by a certificate authority (CA). The host is authenticated based on trust conferred by the CA. This is most often used when many hosts connect to a central gateway.

RSA public key encryption is the preferred method.

There are two connection modes for a connection: tunnel or transport. In tunnel mode, the entire IP packet is encrypted, including the IP headers. In transport mode, the headers are exposed. The tunnel mode is preferred for greater security.

The following resources are recommended for additional information about IPsec and Openswan:

- *Internet Key Exchange* (IKE) - Wikipedia article that describes the IKE protocol used to set up a security association.
- *Security Association* - Wikipedia article that describes the attributes and purpose of a security association.
- *AES instruction set* - Wikipedia article that provides an overview of the Intel Advanced Encryption Standard (AES) instruction set and lists the CPU families that support it.
- *ipsec.conf(5)* - man page for the `ipsec.conf` configuration file.

- *setkey(8)* - man page for the `setkey` utility used to manage the Security Association Database (SAD) and Security Policy Database (SPD) in the Linux kernel.
- *Openswan* - Red Hat Openswan package overview; applies to CentOS also.
- *Host-to-Host VPN Using Openswan* - Red Hat guide for creating a host-to-host VPN using Openswan; can also be used with CentOS.

Installing Openswan

Openswan may be installed using the package manager on your system, by downloading an installable package from the Openswan Web site, or by downloading and compiling source code.

Pivotal recommends that you use Openswan version 2.6.43 or later. If your package manager has an earlier version, you can download an RPM of the latest Openswan release from the Openswan Web site. You can also download Openswan source code from the Openswan Web site.

The following instructions assume you are installing Openswan on hosts running 64-bit Red Hat 6.x or CentOS 6.x.

First, determine if Openswan is already installed and if so, which version:

```
$ sudo yum info installed openswan
```

If the recommended version is already installed, continue with *Configuring and Verifying the Openswan Installation*.

If an older version is installed, uninstall it before continuing:

```
$ sudo yum remove openswan
```

Installing Openswan with an RPM

Enter the following command to see which version of Openswan is available in the package repository:

```
$ sudo yum list available openswan
```

If the recommended version is available, install it on each host in the cluster:

```
$ sudo yum install -y openswan
```

If the recommended version is not in the repository, you can download it from the Openswan Web site at *https://download.openswan.org*. Browse to the `/rhel6/x86_64` directory to find the RPM.

Install the downloaded RPM with a command like the following:

```
$ sudo rpm -i openswanX-version.x86_64.rpm
```

Installing Openswan from Source

If you cannot install Openswan with an RPM you can download the source, compile it, and install it.

1. Download the Openswan source from the Openswan Web site at *Openswan Web site*.
2. Extract the archive and review the `README` file to ensure that the prerequisite packages are installed on your build machine. For example:

   ```
   sudo yum install gmp gmp-devel gawk flex bison \
       iproute2 iptables sed awk bash cut python
   ```

3. Build the Openswan tools by following the instructions in the `README` file. For example:

   ```
   $ make programs
   $ sudo make install
   ```

Configuring and Verifying the Openswan Installation

Follow the steps in this section to configure each host and verify the Openswan installation.

1. Edit `/etc/sysctl.conf` and modify or add the following variables:

   ```
   net.ipv4.conf.default.accept_redirects = 0
   net.ipv4.conf.all.accept_redirects = 0
   net.ipv4.conf.default.send_redirects = 0
   net.ipv4.conf.all.send_redirects = 0
   net.ipv4.ip_forward = 1
   ```

   Execute `sysctl -p` to reload the file.
2. Restore the default SELinux security contexts for the IPsec directory by running the following command:

   ```
   # restorecon -Rv /etc/ipsec.d
   ```
3. If a firewall is enabled, modify it to allow IPsec packets:
   - UDP port 500 for the Internet Key Exchange (IKE) protocol
   - UDP port 4500 for IKE NAT-Traversal
   - Protocol 50 for Encapsulated Security Payload (ESP) IPsec packets
   - *(not recommended)* Protocol 51 for Authenticated Header (AH) IPsec packets

   Here is an example of IPsec rules for iptables:

   ```
   iptables -A INPUT -p udp --sport 500 --dport 500 -j ACCEPT
   iptables -A OUTPUT -p udp --sport 500 --dport 500 -j ACCEPT
   iptables -A INPUT -p udp --sport 4500 --dport 4500 -j ACCEPT
   iptables -A OUTPUT -p udp --sport 4500 --dport 4500 -j ACCEPT
   iptables -A INPUT -p 50 -j ACCEPT
   iptables -A OUTPUT -p 50 -j ACCEPT
   ```
4. Edit the `/etc/ipsec.conf` file and make the following changes:
   - Change `protostack` from `auto` to `netkey`:

     ```
     protostack=netkey
     ```
   - Uncomment or add the following line:

     ```
     include /etc/ipsec.d/*.conf
     ```

     This allows you to create and install a separate configuration file for each host-to-host tunnel.
5. Start IPsec with the `service` command.

   ```
   # service start ipsec
   ```
6. Run `ipsec verify` to check the IPsec installation. Python must be installed to run this command.

   ```
   $ sudo ipsec verify
   ```

   The output looks like the following:

   ```
   Checking if IPsec got installed and started correctly:

   Version check and ipsec on-path                             [OK]
   Openswan U2.6.43/K2.6.32-504.16.2.el6.x86_64 (netkey)
   See `ipsec --copyright' for copyright information.
   Checking for IPsec support in kernel                        [OK]
    NETKEY: Testing XFRM related proc values
            ICMP default/send_redirects                        [OK]
            ICMP default/accept_redirects                      [OK]
            XFRM larval drop                                   [OK]
   Hardware random device check                               [N/A]
   ```

```
Two or more interfaces found, checking IP forwarding      [OK]
Checking rp_filter                                       [ENABLED]
 /proc/sys/net/ipv4/conf/all/rp_filter                    [ENABLED]
 /proc/sys/net/ipv4/conf/lo/rp_filter                     [ENABLED]
 /proc/sys/net/ipv4/conf/eth0/rp_filter                   [ENABLED]
 /proc/sys/net/ipv4/conf/pan0/rp_filter                   [ENABLED]
Checking that pluto is running                           [OK]
 Pluto listening for IKE on udp 500                      [OK]
 Pluto listening for IKE on tcp 500                      [NOT IMPLEMENTED]
 Pluto listening for IKE/NAT-T on udp 4500               [OK]
 Pluto listening for IKE/NAT-T on tcp 4500               [NOT IMPLEMENTED]
 Pluto listening for IKE on tcp 10000 (cisco)            [NOT IMPLEMENTED]
Checking NAT and MASQUERADEing                           [TEST INCOMPLETE]
Checking 'ip' command                                    [OK]
Checking 'iptables' command                              [OK]
```

Note: The result for `Checking 'ip' command` may be `[IP XFRM BROKEN]`, depending on the version of iproute on your system. This can be a misdiagnosis caused by a change in IP XFRM message output between iproute versions.

7. Enable starting IPsec on boot with the following command:

```
# chkconfig ipsec on
```

Configuring Openswan Connections

Set up an IPsec tunnel between each pair of hosts on the cluster by creating a connection. On each connection, one host is designated the "left" host and the other the "right" host. For example, if you have a master (mdw), a standby master (smdw), and three segment hosts (sdw1, sdw2, sdw3), you will need ten connections, as shown in the following table.

Table 6: IPsec connections for a five-host cluster

Connection Number	Left host	Right host
1	mdw	smdw
2	mdw	sdw1
3	mdw	sdw2
4	mdw	sdw3
5	smdw	sdw1
6	smdw	sdw2
7	smdw	sdw3
8	sdw1	sdw2
9	sdw1	sdw3
10	sdw2	sdw3

Complete these tasks to configure connections:

- *Create Host Keys*
- *Create a Connection Configuration File*
- *Test the IPsec Connection*

Create Host Keys

To enable RSA public key authentication for the tunnels, each host must have an RSA key pair. As the root user on each host, enter the following command to generate an authentication key.

```
# ipsec newhostkey --output /etc/ipsec.d/ipsec.secrets --bits 4096
```

The key is saved in the `/etc/ipsec.d/ipsec.secrets` file and its attributes are set to allow access to the root user only.

To view a host's public key, use the `ipsec showhostkey` command with the `--left` or `--right` option. The command outputs the public key in a format suitable for pasting into the connection configuration file. In this example, the keys are shortened for readability:

```
# ipsec showhostkey --left
 # rsakey AQOW+RwpL
 leftrsasigkey=0sAQOW+RwpLg7CGoyywCnv+vnasGJI7...
# ipsec showhostkey --right
 # rsakey AQOW+RwpL
 rightrsasigkey=0sAQOW+RwpLg7CGoyywCnv+vnasGJI7...
```

You will need to use this command as you configure the tunnel between each host pair.

Create a Connection Configuration File

IPsec tunnels are configured by creating a `conn` section in the `/etc/ipsec.conf` file. Because we added the `include /etc/ipsec.d/*.conf` directive to `/etc/ipsec.conf`, we can create a `.conf` file for each connection.

Follow these steps to configure a connection for each pair of hosts.

1. Log in to the host that will be on the "left" side of the tunnel.
2. Create a new configuration file in the `/etc/ipsec.d` directory. Choose a name that includes both host names and has a `.conf` extension. The following configuration file, named `mdw-sdw1.conf`, configures the connection between the hosts mdw and sdw1:

```
conn mdw-sdw1
    leftid=mdw
    left=192.1.2.214
    leftrsasigkey=0sAQOW+RwpLg7CGoyywCnv+vnasGJI7... # shortened for readability
    rightid=sdw1
    right=192.1.2.215
    rightrsasigkey=0sAQNfdDCoDte5bGaGLGkHTKa5GMRl... # shortened for readability
    type=tunnel
    authby=rsasig
    ike=aes192-sha2;dh20
    phase2alg=aes_gcm_c-160-null
    auto=start
```

   See the *ipsec.conf* man page for the complete list of available parameters and their default values.

   The connection name in the example is `mdw-sdw1`.

   For the `leftrsasigkey` use the output from running `ipsec showhostkey --left` on the "left" host. For `rightrsasigkey` use the output from running `ipsec showhostkey --right` on the "right" host.

   Following are recommendations for configuring parameters for Greenplum Database IPsec connections to obtain the best security and performance:

   type
   : Set to `tunnel`, the default mode.

   authby
   : Set to `rsasig`. This is more secure than using pre-shared keys (`psk`).

   auto
   : Set to `start` so that the tunnel is brought up when IPsec starts up.

   ike
   : The `ike` parameter is used during phase 1 to authenticate the peers and negotiate secure session keys for phase2. The parameter value is an entry in the format:

     ```
     cipher-hash;modpgroup, cipher-hash;modpgroup, ....
     ```

     - *cipher* is an encryption algorithm. AES is more secure than 3DES, which is more secure than DES. AES has length of 128, 192, or 256 bits. The more bits, the stronger the encryption, but more time is required for computation.
     - *hash* is the hash algorithm. SHA2 is stronger than SHA1, which is stronger than MD5. SHA2 is recommended, but if SHA2 is not supported on the device, use SHA1. SHA2 is a family of hash functions—SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256—not all of which are supported by Openswan. To find out which algorithms Openswan supports, run the following command after starting the `ipsec` service:

       ```
       # ipsec auto -status
       000 algorithm IKE hash: id=2, name=OAKLEY_SHA1, hashsize=20
       000 algorithm IKE hash: id=4, name=OAKLEY_SHA2_256, hashsize=32
       ```

```
000 algorithm IKE hash: id=6, name=OAKLEY_SHA2_512, hashsize=64
000 algorithm ESP encrypt: id=2, name=ESP_DES, ivlen=8,
 keysizemin=64, keysizemax=64
000 algorithm ESP encrypt: id=3, name=ESP_3DES, ivlen=8,
 keysizemin=192, keysizemax=192
000 algorithm ESP encrypt: id=18, name=ESP_AES_GCM_A, ivlen=8,
 keysizemin=160,
      keysizemax=288000 algorithm ESP encrypt: id=19,
 name=ESP_AES_GCM_B, ivlen=12, keysizemin=160,
      keysizemax=288
000 algorithm ESP encrypt: id=20, name=ESP_AES_GCM_C, ivlen=16,
 keysizemin=160,
      keysizemax=288
```

See *http://en.wikipedia.org/wiki/SHA-2* for information about SHA2.

- *modpgroup* is the Diffie-Hellman group. The peers negotiate a shared secret using the Diffie-Hellman protocol. The Diffie-Hellman group is a set of standardized parameters the peers agree to use as the basis for their calculations. The groups are numbered, and higher numbered groups are more secure (and more compute-intensive) than lower numbered groups. Avoid the lowest numbered groups: 1 (`modp768`), 3 (`modp1024`), and 5 (`modp1576`), which are not considered secure. Choose a higher level group, such as `dh14`, `dh15`, `dh19`, `dh20`, `dh21`, or `dh24`.

phase2

Set to `esp`, the default, to encrypt data. The `ah` setting creates a connection that authenticates, but does not encrypt IP packets.

phase2alg

The `phase2alg` parameter specifies algorithms to use for encrypting and authenticating data. The format and defaults are the same as for the `ike` parameter.

The AES cipher and SHA hash algorithm are more secure. For effective use of emerging 10-gigabit and 40-gigabit network devices, and to enable high speed communication channels, the AES_GCM algorithm is currently the recommended best option. To use AES_GCM, verify that the CPU supports the AES_NI instruction set. See *AES instruction set* for a list of CPUs that support AES_NI.

To see if the CPU supports AES-NI, see if the `aes` flag is set in `/proc/cpuinfo`:

```
grep aes /proc/cpuinfo
```

To see if AES-N1 has been *enabled*, search `/proc/crypto` for the module:

```
grep module /proc/crypto | sort -u
```

To see if the `aesni_intel` kernel module is loaded:

```
/sbin/modinfo aesni_intel
```

To specify the AES_GCM algorithm, use the following syntax:

```
phase2alg=aes_gcm_c-160-null
```

Openswan requires adding the salt size (32 bits) to the key size (128, 192, or 256 bits). In the example above, "160" is calculated by adding a 128-bit key size to the 32 bit salt size. The other valid values are 224 and 288.

3. Use `scp` to copy the configuration file to the "right" host. For example:

```
# scp /etc/ipsec.d/mdw-sdw1.conf sdw1:/etc/ipsec.d/
```

4. Ensure that IPsec is started by executing the following command on both the "left" and "right" hosts:

```
# ipsec service start
```

5. Load the tunnel on both left and right hosts with the following command:

```
# ipsec auto --add mdw-sdw
```

6. Bring up the tunnel on both left and right hosts with the following command:

```
# ipsec auto --up mdw-sdw
```

Test the IPsec Connection

To verify IPsec packets are flowing through a network interface, run the following `tcdump` command on one host and then ping that host from another host.

```
tcdump -n -i interface_name host hostname
```

For example, run the `tcpdump` command on sdw1 and then, on mdw, `ping sdw2`:

```
# tcpdump -n -i eth0 host mdw
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
08:22:10.186743 IP 192.168.1.214 > 192.168.1.215: ESP(spi=0xe56f19ea,seq=0x1), length
 132
08:22:10.186808 IP 192.168.1.214 > 192.168.1.215: ICMP echo request, id 30987, seq 1,
 length 64
08:22:10.186863 IP 192.168.1.215 > 192.168.1.214: ESP(spi=0x4e55824c,seq=0x1), length
 132
08:22:11.189663 IP 192.168.1.214 > 192.168.1.215: ESP(spi=0xe56f19ea,seq=0x2), length
 132
08:22:11.189707 IP 192.168.1.214 > 192.168.1.215: ICMP echo request, id 30987, seq 2,
 length 64
```

The ESP packets verify that the IP packets are encrypted and encapsulated.

When you have connections set up between all of the hosts in the cluster and Greenplum Database is running, you can run the `tcpdump` command on segment hosts to observe database activity in the IPsec tunnels.

Chapter 10

Enabling gphdfs Authentication with a Kerberos-secured Hadoop Cluster

Using external tables and the `gphdfs` protocol, Greenplum Database can read files from and write files to a Hadoop File System (HDFS). Greenplum segments read and write files in parallel from HDFS for fast performance.

When a Hadoop cluster is secured with Kerberos ("Kerberized"), Greenplum Database must be configured to allow the Greenplum Database gpadmin role, which owns external tables in HDFS, to authenticate through Kerberos. This topic provides the steps for configuring Greenplum Database to work with a Kerberized HDFS, including verifying and troubleshooting the configuration.

- *Prerequisites*
- *Configuring the Greenplum Cluster*
- *Creating and Installing Keytab Files*
- *Configuring gphdfs for Kerberos*
- *Testing Greenplum Database Access to HDFS*
- *Troubleshooting HDFS with Kerberos*

Prerequisites

Make sure the following components are functioning and accessible on the network:

- Greenplum Database cluster—either a Pivotal Greenplum Database software-only cluster, or an EMC Data Computing Appliance (DCA).
- Kerberos-secured Hadoop cluster. See the *Greenplum Database Release Notes* for supported Hadoop versions.
- Kerberos Key Distribution Center (KDC) server.

Configuring the Greenplum Cluster

The hosts in the Greenplum Cluster must have a Java JRE, Hadoop client files, and Kerberos clients installed.

Follow these steps to prepare the Greenplum Cluster.

1. Install a Java 1.6 or later JRE on all Greenplum cluster hosts.

   Match the JRE version the Hadoop cluster is running. You can find the JRE version by running `java --version` on a Hadoop node.
2. *(Optional)* Confirm that Java Cryptography Extension (JCE) is present.

   The default location of the JCE libraries is *`JAVA_HOME`*`/lib/security`. If a JDK is installed, the directory is *`JAVA_HOME`*`/jre/lib/security`. The files `local_policy.jar` and `US_export_policy.jar` should be present in the JCE directory.

   The Greenplum cluster and the Kerberos server should, preferably, use the same version of the JCE libraries. You can copy the JCE files from the Kerberos server to the Greenplum cluster, if needed.
3. Set the `JAVA_HOME` environment variable to the location of the JRE in the `.bashrc` or `.bash_profile` file for the `gpadmin` account. For example:

   ```
   export JAVA_HOME=/usr/java/default
   ```
4. Source the `.bashrc` or `.bash_profile` file to apply the change to your environment. For example:

   ```
   $ source ~/.bashrc
   ```
5. Install the Kerberos client utilities on all cluster hosts. Ensure the libraries match the version on the KDC server before you install them.

   For example, the following command installs the Kerberos client files on Red Hat or CentOS Linux:

   ```
   $ sudo yum install krb5-libs krb5-workstation
   ```

   Use the `kinit` command to confirm the Kerberos client is installed and correctly configured.
6. Install Hadoop client files on all hosts in the Greenplum Cluster. Refer to the documentation for your Hadoop distribution for instructions.
7. Set the Greenplum Database server configuration parameters for Hadoop. The `gp_hadoop_target_version` parameter specifies the version of the Hadoop cluster. See the *Greenplum Database Release Notes* for the target version value that corresponds to your Hadoop distribution. The `gp_hadoop_home` parameter specifies the Hadoop installation directory.

   ```
   $ gpconfig -c gp_hadoop_target_version -v "hdp2"
   $ gpconfig -c gp_hadoop_home -v "/usr/lib/hadoop"
   ```

   See the *Greenplum Database Reference Guide* for more information.
8. Reload the updated `postgresql.conf` files for master and segments:

   ```
   gpstop -u
   ```

   You can confirm the changes with the following commands:

   ```
   $ gpconfig -s gp_hadoop_target_version
   $ gpconfig -s gp_hadoop_home
   ```

9. Grant Greenplum Database gphdfs protocol privileges to roles that own external tables in HDFS, including `gpadmin` and other superuser roles. Grant `SELECT` privileges to enable creating readable external tables in HDFS. Grant `INSERT` privileges to enable creating writable exeternal tables on HDFS.

```
#= GRANT SELECT ON PROTOCOL gphdfs TO gpadmin;
#= GRANT INSERT ON PROTOCOL gphdfs TO gpadmin;
```

10. Grant Greenplum Database external table privileges to external table owner roles:

```
ALTER ROLE HDFS_USER CREATEEXTTABLE (type='readable');
ALTER ROLE HDFS_USER CREATEEXTTABLE (type='writable');
```

Note: It is best practice to review database privileges, including gphdfs external table privileges, at least annually.

Creating and Installing Keytab Files

1. Log in to the KDC server as root.
2. Use the `kadmin.local` command to create a new principal for the `gpadmin` user:

   ```
   # kadmin.local -q "addprinc -randkey gpadmin@LOCAL.DOMAIN"
   ```

3. Use `kadmin.local` to generate a Kerberos service principal for each host in the Greenplum Database cluster. The service principal should be of the form *name*/*role*@*REALM*, where:

   - *name* is the gphdfs service user name. This example uses `gphdfs`.
   - *role* is the DNS-resolvable host name of a Greenplum cluster host (the output of the `hostname -f` command).
   - *REALM* is the Kerberos realm, for example `LOCAL.DOMAIN`.

   For example, the following commands add service principals for four Greenplum Database hosts, mdw.example.com, smdw.example.com, sdw1.example.com, and sdw2.example.com:

   ```
   # kadmin.local -q "addprinc -randkey gphdfs/mdw.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "addprinc -randkey gphdfs/smdw.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "addprinc -randkey gphdfs/sdw1.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "addprinc -randkey gphdfs/sdw2.example.com@LOCAL.DOMAIN"
   ```

   Create a principal for each Greenplum cluster host. Use the same principal name and realm, substituting the fully-qualified domain name for each host.
4. Generate a keytab file for each principal that you created (`gpadmin` and each `gphdfs` service principal). You can store the keytab files in any convenient location (this example uses the directory `/etc/security/keytabs`). You will deploy the service principal keytab files to their respective Greenplum host machines in a later step:

   ```
   # kadmin.local -q "xst -k /etc/security/keytabs/gphdfs.service.keytab
    gpadmin@LOCAL.DOMAIN"
   # kadmin.local -q "xst -k /etc/security/keytabs/mdw.service.keytab gpadmin/mdw
    gphdfs/mdw.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "xst -k /etc/security/keytabs/smdw.service.keytab gpadmin/smdw
    gphdfs/smdw.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "xst -k /etc/security/keytabs/sdw1.service.keytab gpadmin/sdw1
    gphdfs/sdw1.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "xst -k /etc/security/keytabs/sdw2.service.keytab gpadmin/sdw2
    gphdfs/sdw2.example.com@LOCAL.DOMAIN"
   # kadmin.local -q "listprincs"
   ```

5. Change the ownership and permissions on `gphdfs.service.keytab` as follows:

   ```
   # chown gpadmin:gpadmin /etc/security/keytabs/gphdfs.service.keytab
   # chmod 440 /etc/security/keytabs/gphdfs.service.keytab
   ```

6. Copy the keytab file for `gpadmin@LOCAL.DOMAIN` to the Greenplum master host:

   ```
   # scp /etc/security/keytabs/gphdfs.service.keytab mdw_fqdn:/home/gpadmin/
   gphdfs.service.keytab
   ```

7. Copy the keytab file for each service principal to its respective Greenplum host:

   ```
   # scp /etc/security/keytabs/mdw.service.keytab mdw_fqdn:/home/gpadmin/
   mdw.service.keytab
   # scp /etc/security/keytabs/smdw.service.keytab smdw_fqdn:/home/gpadmin/
   smdw.service.keytab
   # scp /etc/security/keytabs/sdw1.service.keytab sdw1_fqdn:/home/gpadmin/
   sdw1.service.keytab
   ```

```
# scp /etc/security/keytabs/sdw2.service.keytab sdw2_fqdn:/home/gpadmin/
sdw2.service.keytab
```

Configuring gphdfs for Kerberos

1. Edit the Hadoop `core-site.xml` client configuration file on all Greenplum cluster hosts. Enable service-level authorization for Hadoop by setting the `hadoop.security.authorization` property to `true`. For example:

```
<property>
    <name>hadoop.security.authorization</name>
    <value>true</value>
</property>
```

2. Edit the `yarn-site.xml` client configuration file on all cluster hosts. Set the resource manager address and yarn Kerberos service principle. For example:

```
<property>
    <name>yarn.resourcemanager.address</name>
    <value>hostname:8032</value>
</property>
<property>
    <name>yarn.resourcemanager.principal</name>
    <value>yarn/hostname@DOMAIN</value>
</property>
```

3. Edit the `hdfs-site.xml` client configuration file on all cluster hosts. Set properties to identify the NameNode Kerberos principals, the location of the Kerberos keytab file, and the principal it is for:

   - `dfs.namenode.kerberos.principal` - the Kerberos principal name the gphdfs protocol will use for the NameNode, for example `gpadmin@LOCAL.DOMAIN`.
   - `dfs.namenode.https.principal` - the Kerberos principal name the gphdfs protocol will use for the NameNode's secure HTTP server, for example `gpadmin@LOCAL.DOMAIN`.
   - `com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file` - the path to the keytab file for the Kerberos HDFS service, for example `/home/gpadmin/mdw.service.keytab.`.
   - `com.emc.greenplum.gpdb.hdfsconnector.security.user.name` - the gphdfs service principal for the host, for example `gphdfs/mdw.example.com@LOCAL.DOMAIN`.

   For example:

```
<property>
    <name>dfs.namenode.kerberos.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
    <name>dfs.namenode.https.principal</name>
    <value>gphdfs/gpadmin@LOCAL.DOMAIN</value>
</property>
<property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.keytab.file</name>
    <value>/home/gpadmin/gpadmin.hdfs.keytab</value>
</property>
<property>
    <name>com.emc.greenplum.gpdb.hdfsconnector.security.user.name</name>
    <value>gpadmin/@LOCAL.DOMAIN</value>
</property>
```

Testing Greenplum Database Access to HDFS

Confirm that HDFS is accessible via Kerberos authentication on all hosts in the Greenplum cluster. For example, enter the following command to list an HDFS directory:

```
hdfs dfs -ls hdfs://namenode:8020
```

Create a Readable External Table in HDFS

Follow these steps to verify that you can create a readable external table in a Kerberized Hadoop cluser.

1. Create a comma-delimited text file, `test1.txt`, with contents such as the following:

   ```
   25, Bill
   19, Anne
   32, Greg
   27, Gloria
   ```

2. Persist the sample text file in HDFS:

   ```
   hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
   ```

3. Log in to Greenplum Database and create a readable external table that points to the `test1.txt` file in Hadoop:

   ```
   CREATE EXTERNAL TABLE test_hdfs (age int, name text)
   LOCATION('gphdfs://namenode:8020/tmp/test1.txt')
   FORMAT 'text' (delimiter ',');
   ```

4. Read data from the external table:

   ```
   SELECT * FROM test_hdfs;
   ```

Create a Writable External Table in HDFS

Follow these steps to verify that you can create a writable external table in a Kerberized Hadoop cluster. The steps use the `test_hdfs` readable external table created previously.

1. Log in to Greenplum Database and create a writable external table pointing to a text file in HDFS:

   ```
   CREATE WRITABLE EXTERNAL TABLE test_hdfs2 (LIKE test_hdfs)
   LOCATION ('gphdfs://namenode:8020/tmp/test2.txt'
   FORMAT 'text' (DELIMITER ',');
   ```

2. Load data into the writable external table:

   ```
   INSERT INTO test_hdfs2
   SELECT * FROM test_hdfs;
   ```

3. Check that the file exists in HDFS:

   ```
   hdfs dfs -ls hdfs://namenode:8020/tmp/test2.txt
   ```

4. Verify the contents of the external file:

   ```
   hdfs dfs -cat hdfs://namenode:8020/tmp/test2.txt
   ```

Troubleshooting HDFS with Kerberos

Forcing Classpaths

If you encounter "class not found" errors when executing `SELECT` statements from `gphdfs` external tables, edit the `$GPHOME/lib/hadoop-env.sh` file and add the following lines towards the end of the file, before the `JAVA_LIBRARY_PATH` is set. Update the script on all of the cluster hosts.

```
if [ -d "/usr/hdp/current" ]; then
for f in /usr/hdp/current/**/*.jar; do
    CLASSPATH=${CLASSPATH}:$f;
done
fi
```

Enabling Kerberos Client Debug Messages

To see debug messages from the Kerberos client, edit the `$GPHOME/lib/hadoop-env.sh` client shell script on all cluster hosts and set the `HADOOP_OPTS` variable as follows:

```
export HADOOP_OPTS="-Djava.net.prefIPv4Stack=true -Dsun.security.krb5.debug=true
 ${HADOOP_OPTS}"
```

Adjusting JVM Process Memory on Segment Hosts

Each segment launches a JVM process when reading or writing an external table in HDFS. To change the amount of memory allocated to each JVM process, configure the `GP_JAVA_OPT` environment variable.

Edit the `$GPHOME/lib/hadoop-env.sh` client shell script on all cluster hosts.

For example:

```
export GP_JAVA_OPT=-Xmx1000m
```

Verify Kerberos Security Settings

Review the `/etc/krb5.conf` file:

- If AES256 encryption is not disabled, ensure that all cluster hosts have the JCE Unlimited Strength Jurisdiction Policy Files installed.
- Ensure all encryption types in the Kerberos keytab file match definitions in the `krb5.conf` file.

  ```
  cat /etc/krb5.conf | egrep supported_enctypes
  ```

Test Connectivity on an Individual Segment Host

Follow these steps to test that a single Greenplum Database host can read HDFS data. This test method executes the Greenplum `HDFSReader` Java class at the command-line, and can help to troubleshoot connectivity problems outside of the database.

1. Save a sample data file in HDFS.

   ```
   hdfs dfs -put test1.txt hdfs://namenode:8020/tmp
   ```

2. On the segment host to be tested, create an environment script, `env.sh`, like the following:

   ```
   export JAVA_HOME=/usr/java/default
   export HADOOP_HOME=/usr/lib/hadoop
   export GP_HADOOP_CON_VERSION=hdp2
   ```

```
export GP_HADOOP_CON_JARDIR=/usr/lib/hadoop
```

3. Source all environment scripts:

```
source /usr/local/greenplum-db/greenplum_path.sh
source env.sh
source $GPHOME/lib/hadoop-env.sh
```

4. Test the Greenplum Database HDFS reader:

```
java com.emc.greenplum.gpdb.hdfsconnector.HDFSReader 0 32 TEXT hdp2
 gphdfs://namenode:8020/tmp/test1.txt
```

Chapter 11

Security Best Practices

This chapter describes basic security best practices that Pivotal recommends you follow to ensure the highest level of system security.

Greenplum Database default security configuration:

- Only local connections are allowed.
- Basic authentication is configured for the superuser (`gpadmin`).
- The superuser is authorized to do anything.
- Only database role passwords are encrypted.

System User (gpadmin)

Secure and limit access to 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. The `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. The 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 the `gpadmin` user id.

The `gpadmin` user can bypass all security features of Greenplum Database. Anyone who logs on to a Greenplum host with 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 allow essential system administrators access to it.

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`.

Superusers

Roles granted the `SUPERUSER` attribute are superusers. Superusers bypass all access privilege checks and resource queues. Only system administrators should be given superuser rights.

See "Altering Role Attributes" in the *Greenplum Database Administrator Guide*.

Login Users

Assign a distinct role to each user who logs in and set the `LOGIN` attribute.

For logging and auditing purposes, each user who 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*.

Each login role should be assigned to a single, non-default resource queue.

Groups

Use groups to manage access privileges.

Create a group for each logical grouping of object/access permissions.

Every login user should belong to one or more roles. Use the `GRANT` statement to add group access to a role. Use the `REVOKE` statement to remove group access from a role.

The `LOGIN` attribute should not be set for group roles.

See "Creating Groups (Role Membership)" in the *Greenplum Database Administrator Guide*.

Object Privileges

Only the owner and superusers have full permissions to new objects. Permission must be granted to allow other rules (users or groups) to access objects. Each type of database object has different privileges that may be granted. Use the `GRANT` statement to add a permission to a role and the `REVOKE` statement to remove the permission.

You can change the owner of an object using the `REASIGN OWNED BY` statement. For example, to prepare to drop a role, change the owner of the objects that belong to the role. Use the `DROP OWNED BY` to drop objects, including dependent objects, that are owned by a role.

Schemas can be used to enforce an additional layer of object permissions checking, but schema permissions do not override object privileges set on objects contained within the schema.

Operating System Users and File System

To protect the network from intrusion, system administrators should verify the passwords used within an organization are sufficently strong. The following recommendations can strengthen a password:

- Minimum password length recommendation: At least 9 characters. MD5 passwords should be 15 characters or longer.
- Mix upper and lower case letters.
- Mix letters and numbers.
- Include non-alphanumeric characters.
- Pick a password you can remember.

The following are recommendations for password cracker software that you can use to determine the strength of a password.

- John The Ripper. A fast and flexible password cracking program. It allows the use of multiple word lists and is capable of brute-force password cracking. It is available online at *http://www.openwall.com/john/*.
- Crack. Perhaps the most well-known password cracking software, Crack is also very fast, though not as easy to use as John The Ripper. It can be found online at *http://www.crypticide.com/alecm/security/crack/c50-faq.html*.

The security of the entire system depends on the strength of the root password. This password should be at least 12 characters long and include a mix of capitalized letters, lowercase letters, special characters, and numbers. It should not be based on any dictionary word.

Password expiration parameters should be configured.

Ensure the following line exists within the file `/etc/libuser.conf` under the `[import]` section.

```
login_defs = /etc/login.defs
```

Ensure no lines in the `[userdefaults]` section begin with the following text, as these words override settings from `/etc/login.defs`:

- `LU_SHADOWMAX`
- `LU_SHADOWMIN`
- `LU_SHADOWWARNING`

Ensure the following command produces no output. Any accounts listed by running this command should be locked.

```
grep "^+:" /etc/passwd /etc/shadow /etc/group
```

Note: We strongly recommend that customers change their passwords after initial setup.

```
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```

Find all the files that are world-writable and that do not have their sticky bits set.

```
find / -xdev -type d \( -perm -0002 -a ! -perm -1000 \) -print
```

Set the sticky bit (`# chmod +t {dir}`) for all the directories that result from running the previous command.

Find all the files that are world-writable and fix each file listed.

```
find / -xdev -type f -perm -0002 -print
```

Set the right permissions (`# chmod o-w {file}`) for all the files generated by running the aforementioned command.

Find all the files that do not belong to a valid user or group and either assign an owner or remove the file, as appropriate.

```
find / -xdev \( -nouser -o -nogroup \) -print
```

Find all the directories that are world-writable and ensure they are owned by either root or a system account (assuming only system accounts have a User ID lower than 500). If the command generates any output, verify the assignment is correct or reassign it to root.

```
find / -xdev -type d -perm -0002 -uid +500 -print
```

Authentication settings such as password quality, password expiration policy, password reuse, password retry attempts, and more can be configured using the Pluggable Authentication Modules (PAM) framework. PAM looks in the directory `/etc/pam.d` for application-specific configuration information. Running `authconfig` or `system-config-authentication` will re-write the PAM configuration files, destroying any manually made changes and replacing them with system defaults.

The default `pam_cracklib` PAM module provides strength checking for passwords. To configure `pam_cracklib` to require at least one uppercase character, lowercase character, digit, and special character, as recommended by the U.S. Department of Defense guidelines, edit the file `/etc/pam.d/system-auth` to include the following parameters in the line corresponding to password requisite `pam_cracklib.so try_first_pass`.

```
retry=3:
dcredit=-1. Require at least one digit
ucredit=-1. Require at least one upper case character
ocredit=-1. Require at least one special character
lcredit=-1. Require at least one lower case character
minlen-14. Require a minimum password length of 14.
```

For example:

```
password required pam_cracklib.so try_first_pass retry=3\minlen=14 dcredit=-1
 ucredit=-1 ocredit=-1 lcredit=-1
```

These parameters can be set to reflect your security policy requirements. Note that the password restrictions are not applicable to the root password.

The `pam_tally2` PAM module provides the capability to lock out user accounts after a specified number of failed login attempts. To enforce password lockout, edit the file `/etc/pam.d/system-auth` to include the following lines:

- The first of the auth lines should include:

  ```
  auth required pam_tally2.so deny=5 onerr=fail unlock_time=900
  ```

- The first of the account lines should include:

  ```
  account required pam_tally2.so
  ```

Here, the deny parameter is set to limit the number of retries to 5 and the `unlock_time` has been set to 900 seconds to keep the account locked for 900 seconds before it is unlocked. These parameters may be configured appropriately to reflect your security policy requirements. A locked account can be manually unlocked using the `pam_tally2` utility:

```
/sbin/pam_tally2 --user {username} -reset
```

You can use PAM to limit the reuse of recent passwords. The remember option for the `pam_ unix` module can be set to remember the recent passwords and prevent their reuse. To accomplish this, edit the appropriate line in `/etc/pam.d/system-auth` to include the remember option.

For example:

```
password sufficient pam_unix.so [ … existing_options …]
remember=5
```

You can set the number of previous passwords to remember to appropriately reflect your security policy requirements.

```
cd /etc
chown root:root passwd shadow group gshadow
chmod 644 passwd group
chmod 400 shadow gshadow
```