PingDirectory™ and PingData™ Security Guide

Version:7.3

PingDirectory™ and PingData™ Security Guide

Trademarks

Disclaimer

The information provided in these documents is provided "as is" without warranty of any kind. Ping Identity disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall Ping Identity or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if Ping Identity or its suppliers have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of liability for consequential or incidental damages so the foregoing limitation may not apply.

Support

https://support.pingidentity.com/

Table of Contents

-

-

-

Chapter 1: Introduction

Storing and handling consumer identity data requires taking appropriate steps to safeguard it, while continuing to provide fast, real-time, and highly-available services for the consumers who consent to its use.

Topics include:

Security in an identity environment

PingData security features

Security risks in an identity environment

The PingData Platform serves as the authentication repository for a wide variety of network applications, sensitive user information, and application data. Hackers that obtain user credentials can cause extensive damage to individuals, systems, and businesses. News of companies suffering from a data breach is more common and more concerning for consumers.

Financial and reputation costs

Business costs to secure and monitor identity deployments can be large, but the total cost is small compared to the cost of a security breach. A security breach requires resources to investigate the incident, assess the scope of damage, and identify and fix any compromised data. Businesses face compensating users for downtime and for costs incurred from the exposure of their personal data. But, the damage to a company's reputation is the most costly result.

Common attack models

Directories are the central component within identity management systems. They streamline the authentication, authorization, and privilege granting across system boundaries. Whether for user, account, or subscriber provisioning, directory services systems must be properly secured so that sensitive information is not accessible by unauthorized individuals externally or internally.

This guide describes procedures to mitigate three broad classes of security threats.

Common Attack Models

Attack Model	Description	Ease of Detection	Potential Loss
Man-in-the-Middle Attacks	The communication between systems is compromised, allowing the attacker to insert himself in the conversation, undetected by the legitimate systems.	Difficult to detect.	Generally results in the loss of specific data.
Denial-of-Service Attacks	An attacker (or a poorly coded client application) swamps the system with requests that cripple its ability to operate and serve legitimate clients.	Easy to detect.	Limited data loss, but severe disruption to normal business operations.
Data Breach and Data Trawling Attacks	The attacker gains access to data they should not have.	Difficult to detect.	Often results in the loss of a large amount of data in a single event.

PingData security features

The PingData Platform provides the following security and monitoring components:

Network Encryption with SSL and StartTLS. PingData servers support SSL and StartTLS to encrypt communication with clients. Administrators can configure different certificates for each connection handler, or use the same certificate for all connection handlers. SSL or StartTLS can also be configured to secure communication between server components. Replication between PingDirectory Servers uses SSL. The server also enables fine-grained control of the key material used in connecting peers in SSL handshakes and trust material for storing certificates.

SASL Authentication Mechanism Support. PingData servers support SASL mechanisms including Anonymous, Cram-MD5, Digest-MD5, External, Plain, and GSSAPI. Servers using Cram-MD5 and Digest-MD5 require access to the clear-text password for a user. In this case, the PingDirectory Server supports reversible encryption to store passwords with more secure encoding. The PingDirectory Server server also supports two types of one-time password (OTP) mechanisms for multi-factor authentication: UnboundID-TOTP SASL and UnboundID-Delivered-OTP SASL. The proprietary UnboundID-TOTP SASL mechanism allows multi-factor authentication to the server using time-based one-time password (TOTP) code. The proprietary UnboundID-Delivered-OTP SASL mechanism allows multi-factor authentication to the server by delivering a one-time password to the end user through some out-of-band channel, such as email or SMS.

Another component is the UnboundID Certificate Plus Password SASL mechanism, which is used to perform multifactor authentication against the PingDirectory Server using both a client certificate, presented during SSL/TLS negotiation, and a static password.

Certificate-based Authentication. PingData servers support client-based authentication, which uses a client certificate as the set of credentials for LDAP authentication during SSL or StartTLS negotiation.

Password Policies. PingData servers provide extensive password policy support including:

- maximum password age
- maximum password reset age
- configurable password warning intervals
- grace logins
- ability to disallow password changes after expiration
- forcing passwords to be changed when accounts are added or reset
- preventing password reuse by time or number of old passwords
- account lock-outs based on failed login attempts
- preset account expiration time (for temporary accounts)
- idle time log-outs
- a password generator
- multiple default password storage schemes
- account expiration

Password Storage Schemes. PingData servers support password storage schemes such as one-way digests (CRYPT, SCRYPT, BCRYPT, PBKDF2, MD5, SMD5, SHA, SSHA, SSHA256, SSHA384, SSHA512) and reversible encryption (BASE64, 3DES, AES, RC4, BLOWFISH). Password Policies can also require a specific authentication mechanisms for users associated with the policy.

Message Digests & Encryption Algorithms for Passwords. PingData servers support the use of one-way message digests (CRYPT, SCRYPT, BCRYPT, PBKDF2, 128-bit MD5, 160-bit SHA-1, and 256-bit, 384-bit, and 512-bit SHA-2), and a number of reversible encryption algorithms (BASE64, 3DES, AES, RC4, and Blowfish) for storing passwords. Even if passwords are encoded using reversible encryption, that encryption is intended for use only within the server. Passwords are not made available to administrators in unencrypted form. Encrypted password storage should only be used if using an authentication mechanism that requires the server to have access to clear-text passwords, like CRAM-MD5 or DIGEST-MD5.

Client Connection Policies. PingData servers can control which clients get connected to the server, how they are connected, and what resources or operations are available to them. For example, client connection criteria can be defined to block specific IP addresses or domains.

When a client establishes a connection to the server, the server assigns a policy for that connection. If the client performs a bind, which can change the identity of that connection, or uses the StartTLS extended operation, which can change an insecure connection to a secure one, the server re-evaluates the connection and assigns it a different policy.

Full-Featured Access Control System. PingData servers provide an access control subsystem that determines whether a given operation is allowed based on specified criteria. The access control system is used to grant or restrict access to data, restrict the use of specific types of controls and extended operations and provides strong validation for access control rules before accepting them.

Privileges. PingData servers include a privilege subsystem that works with the access control subsystem. Operations are only allowed if both privilege and access control criteria are met.

Encrypted Backups. PingData servers protect the integrity of backup contents using cryptographic digests and encryption. When restoring the backup, servers verify that the digest matches the content of the backup and generates an error if the backup has been changed.

Global Settings. Key security features are configured globally, and apply to the service as a whole. These features include schema validation, authentication and authorization constraint policies, limiting resource consumption (to defend against denial-of-service attacks), data protection, and encryption controls.

Lock-Down Mode. PingData servers can automatically enter lockdown mode when certain events are triggered, and only allow requests from users who have the lockdown-mode privilege.

Sensitive Attributes. PingData servers support sensitive attributes used to prevent access or restrict access to secure connections. Sensitive attributes can also restrict access to those users who are not subject to access control processing (those users with the `bypass-acl` or `bypass-read-acl` privilege), but for which access should still be restricted.

Operational Attributes. The PingDirectory Server provides a number of operational attributes that can be added to user entries in order to restrict the way those users can authenticate and the circumstances under which they can be used for proxied authorization.

System and Audit Logging. PingData servers provide extensive logging and auditing capabilities that can detect attacks and assess potential damage.

Plug-Ins and SDK Extensions. PingData servers provide plug-in points and extensions for custom certificate mappers, trust manager providers, post-connect and post-disconnect for client connections and many others.

Chapter 2: Client access

Mitigating the risk of data exposure requires understanding the expected uses of the directory service, the nature of the data stored, and the clients that can access it. Knowing what data clients need to access and the ways they need to interact with it can help define security policies.

Topics include:

Identifying potential clients

The capabilities of the clients using PingData services will determine the security features used. For example, if some clients do not support SSL or StartTLS, a less secure type of communication maybe required. If some clients can only perform simple binds, SASL authentication may not be an option.

If the set of clients is known ahead of time, the server can be configured based on their capabilities. Even if some clients are not known in advance, the client connection policies enable separating and restricting unknown or insecure clients. Things to consider when tuning security options for clients include:

- Is the set of clients for the directory service well defined, or an arbitrary or diverse set of clients? If the answer is not known, is it possible to enforce a minimum set capabilities for all clients to use?
- Do all clients support the use of SSL and/or StartTLS? If so, the server can be configured to accept only secure requests.
- Do any clients require unauthenticated access to the server? If not, configure the server to accept only authenticated requests. If some unauthenticated requests are required, create a client connection policy specifically for those clients that limits the kinds of operations they can request.
- If the set of clients that will interact with the server fit into well defined groups, create separate client connection policies for each group.

Clients requiring privileged ports

Many operating systems consider ports 1 through 1024 to be “privileged” ports. By default only processes owned by (or initially started by) the root user can listen on them. Though PingData servers can be configured to listen on any set of ports, running the PingDirectory Server (or any network process) under a root user account is not recommended.

The vast majority of LDAP client applications make it possible to configure the ports that they should use to communicate with an LDAP server. If all applications in this environment provide this support, run all server instances on unprivileged ports (such as 1389 and 1636).

If there are client applications that can only use privileged ports (such as 389 and 636), configure the operating system to allow servers to use those ports.

Identifying data security

Understanding how to secure data requires knowing what data will be stored and how it will be accessed. Even if all of the data in the directory environment is considered sensitive to some extent, some elements are more sensitive, or may have different requirements for client interaction. For example, although passwords are critical for authentication and must be changed over LDAP, well-designed LDAP clients should not need to retrieve them from the server.

Answer the following questions for attributes that will be stored in the server:

- What do clients need to retrieve the attribute from the server? Do any clients need to access it over insecure connections?
- What attributes do clients need to be able to use in search filters for searches with a `baseObject` scope? Searches with a `baseObject` scope do not require any attribute indexes.
- What attributes do clients need to be able to use in search filters for searches with a scope other than `baseObject`?

Most of the attributes in an entry will fit into the same category. It is not necessary (and generally not recommended) to specify different access control rules or sensitive attribute definitions for each individual attribute. Create one rule for each class of similar attributes. Attributes that exist in multiple classes can be governed by the most restrictive of those classes.

Answer the following questions for the client applications that will access attributes:

- What kinds of operations do they perform?
- What indexes are required?
- Is there any need for insecure client access?
- Does the application need to perform any updates?
- Does the application access or store sensitive data?

Chapter 3: Mitigating system attacks

There are three main system attack types:

Denial of Service Attacks – Malicious clients or rogue programs that continuously perform expensive operations that exhaust the available resources of the server. The primary goal of this attack is to take the system down and impede user access.

Data Breach Attacks – An attacker accesses and steals private data. Data breaches often lead to data trawling attacks and unauthorized bulk downloads of data.

Man-in-the-Middle Attacks – A connection is established between the Identity server and a client by an intermediary host that relays messages between them. The client and target server are unaware of this eavesdropping attack, which can be used to intercept sensitive data, manipulate data transmission, or inject malicious code to compromise security.

Topics include:

Denial of service prevention

Data breach prevention

Man-in-the-middle attack prevention

Denial of service prevention

The PingData Platform provides a number of features that can help avoid denial of service attacks:

- Server's monitoring tools detect attacks.
- Resource limits enforced on all clients to prevent a denial of service attack.
- Restrictions on the types of operations that clients can request.
- Cllient type access restrictions.

The PingDirectory Server and the PingData Platform can restrict the type of operations that clients can request, the rate at which clients can issue requests, the number of concurrent requests per connection, and the number of concurrent connections per client. For example, access to expensive unindexed searches can be limited, including restricting access to specific users and limiting the number of concurrent unindexed searches. Search requests can be limited by the number of entries returned, the length of time they are allowed to take, and the number of entries that can be examined during processing. The PingDirectoryProxy Server also provides health checking and load balancing capabilities that can detect servers that are slow or unresponsive and route requests to healthy servers.

If a malicious client is discovered, associated connection(s) can be terminated and future connections prevented. If one or more clients are able to consume all available worker threads, work queue monitoring can immediately notify administrators, and the servers can provide additional worker threads that are reserved for processing administrative requests.

Monitoring tools

There are three methods of monitoring the performance of PingData servers. Each of these can be used to examine server performance and compare suspicious levels of activity with normal patterns. In conjunction with other tools, they can provide alerts and alarms:

- **PingDataMetrics Server** – A data collection and aggregation server that collects performance and event data from a set of PingData servers. It can report the overall performance of the entire directory service, as well as report on individual servers. The PingDataMetrics Server normalizes and aggregates this data and makes it available through a REST API and chart output. Both historical and current data is available.

- **cn=monitor Entry** – The entry used by each PingData server to expose monitoring information. The `cn=monitor` data is also available through SNMP Management Information Bases (MIBs), including the Processing Time MIB, the System Status MIB, and the LDAP Statistics MIB. Data can be accessed with tools like the servers' Administrative Console, JConsole, LDAP command-line tools, and JMX.
- **Stats Logger** – A built-in tool for all PingData servers that is useful for profiling server performance for a given configuration. When enabled, the Stats Logger writes server statistics to a log file in a comma separated format (.csv) at a specified interval. The logger has a negligible impact on server performance unless the log-interval property is set to a very small value (less than 1 second).

System alerts

The system also supports a number of alert handlers:

- **Error Log Alert Handler** – Sends administrative alerts to the configured server error logger(s).
- **Exec Alert Handler** – Executes a specified command on the local system if an administrative alert matching the criteria for this alert handler is generated by the server. Information about the administrative alert will be made available to the executed application as arguments provided by the command.
- **SNMP, JMX, and SMTP (mail) Alert Handlers** – Send administrative alerts via their respective protocols.
- **Groovy Scripted Alert Handler** – Provides alert handler implementations defined in a dynamically-loaded Groovy script that implements the `ScriptedAlertHandler` class defined in the Server SDK.
- **Third Party Alert Handler** – Provides alert handler implementations created in third-party code using the Server SDK.

System alarms and gauges

Each PingData server installs a set of gauges that are specific to the product and that can be cloned or configured through the `dsconfig` tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server available' or ‘server unavailable’ (Indicator Gauge). Gauges generate alarms, when the gauge’s severity changes due to changes in the monitored value. Like alerts, alarms have a severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a `Condition` property, and may have a `Specific Problem` or `Resource` property. If surfaced through SNMP, a `Probable Cause` property and `Alarm Type` property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool. As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

Enforce resource limits

The PingData product family provides methods to enforce resource limits to protect against denial-of-service attacks. These include setting global configuration properties, configuring the limits in Client Connection Policies, or configuring operational attributes. For details about these properties, see the administration guide for the specific server.

Enforce resource limits with global configuration options

These properties of the Global Configuration are relevant for protection against denial of service attacks:

- **Limit the Max Number of Connections** – Includes a number of properties that can be used to control the maximum number of connections established with the server. This includes `maximum-concurrent-connections`, `maximum-concurrent-connections-per-ip-address`, and `maximum-concurrent-connections-per-bind-dn`. If any connection limit is reached, any subsequent connections are terminated.
- **allowed-task** – Specifies the task classes that the server can run. Tasks allow LDAP clients to request operations, including shutting down or restarting the server, importing

data from LDIF, restoring data from a backup, rebuilding a database index, or forcing a JVM garbage collection. The Server SDK also supports custom Java-based or Groovy-based tasks.

- **disabled-privilege** – Specifies privileges that should be disabled. If a privilege is disabled, it is assumed that all users have that privilege. The user will still be required to satisfy any other requirements (such as access control permissions) that the server has in place for that operation.
- **size-limit** – Specifies the maximum number of entries that a user can retrieve in a single search operation. This limit can be overridden on a per-user basis with the `ds-rlim-size-limit` operational attribute in the user's entry (in a real or virtual attribute), which is reserved for users that need to retrieve many entries.
- **time-limit** – Specifies the maximum length of time that the server is allowed to spend on any user-requested search operation. This limit can be overridden on a per-user basis using the `ds-rlim-time-limit` operational attribute (in a real or virtual attribute) in the user's entry.
- **lookthrough-limit** – Specifies the maximum number of entries that the server can examine while processing a single search. This count can include entries that don't match the search criteria or that the user doesn't have permission to access. The `ds-rlim-lookthrough-limit` operational attribute (as a real or virtual attribute) can be used to set an alternate limit.
- **idle-time-limit** – Specifies the maximum length of time that a client can maintain a connection without any active operations. This is useful for dealing with applications that establish connections, and then fail to close those connections when no longer needed. This can be overridden on a per-user basis with the `ds-rlim-idle-time-limit` operational attribute in a user's entry (as a real or virtual attribute).
- **maximum-concurrent-connections** – Specifies the maximum number of connections that can be established with the server at one time. If the limit is reached, new connection attempts are rejected until existing ones are closed. However, the maximum number of connections is ultimately determined by the number of file descriptors available to the JVM (minus the number of descriptors needed for interacting with local files).
- **maximum-concurrent-connections-per-ip-address** – Specifies the maximum number of connections that can be established with the server at one time from a single IP address. If a client has the maximum connections established, additional attempts from that client are rejected until existing connections are closed.
- **maximum-concurrent-connections-per-bind-dn** – Specifies the maximum number of connections that can be established concurrently while authenticated as a given user.

If this limit is reached, any connection attempts to authenticate as that user will be terminated.

- **maximum-concurrent-unindexed-searches** – Specifies the maximum number of unindexed searches that can be processed at one time. Unindexed searches can tie up worker threads for a significant length of time.
- **duplicate-error-log-limit** – Specifies the maximum number of duplicate messages that can be written to the server error log within a specified time period (defined by the `duplicate-error-log-time-limit` property). This prevents a frequently-encountered problem from filling the server error log. If this limit is exceeded, a message is recorded at the end of that time period stating the number of messages that were suppressed.
- **duplicate-error-log-time-limit** – Specifies the duration for which the `duplicate-errorlog-limit` property will be in effect.
- **duplicate-alert-limit** – Specifies the maximum number of administrative alerts of the same type that can be generated within a specified time period (defined by the `duplicate-alert-time-limit` property). This limits the number of administrative alerts generated if a recurring problem exists within the server. If this limit is exceeded, an alert is generated at the end of the specified time period stating the number of alerts that were suppressed.
- **duplicate-alert-time-limit** – Specifies the duration for which the `duplicate-alert-limit` property will be in effect.

Enforce resource limits with client connection policies

Configuration properties in client connection policy objects enforce restrictions on the resources that clients can consume. A policy is associated with each connection to the server. If multiple policies exist, they are evaluated in ascending order of the assigned evaluation order index. Policies with a lower index number are evaluated first. The first policy that the server finds whose criteria match the client connection will be associated with that connection. If no client connection policy is found with criteria matching the connection, then the connection is terminated.

Properties include:

- **maximum-concurrent-connections** – Specifies the maximum number of client connections allowed at one time per policy. If the maximum number of connections have already been assigned through the policy, the new connection is terminated.
- **maximum-connection-duration** – Specifies the maximum length of time that a client connection is allowed to remain, regardless of the level of activity on that client

connection. If a connection associated with this policy exceeds the value, it is terminated.

- **maximum-idle-connection-duration** – Specifies the maximum length of time that a client connection can remain established without any active requests in progress. If the connection associated with this policy exceeds this value, it is terminated.
- **maximum-operation-count-per-connection** – Specifies the maximum number of operations that a client connection can request over the life of that connection. If the client submits more than the value, the connection is terminated.
- **maximum-concurrent-operations-per-connection** – Specifies the maximum number of operations a connection can have in progress at one time. If the client reaches the limit, any additional request will either be rejected or delayed before a timeout (specified through the `maximum-concurrent-operation-wait-time-before-rejecting` property).
- **maximum-concurrent-operation-wait-time-before-rejecting** – Specifies the maximum length of time that the server should allow a client request to wait for an operation to complete, before it can be processed within the `maximum-concurrent-operations-per-connection` limit.
- **maximum-connection-operation-rate** – Specifies the maximum rate at which a client connection associated with the policy can submit operation requests. Values are specified as "100/s" for a limit of one hundred operations in a one-second period, or "1000/5m" for a limit of one thousand operations in a five-minute period. Multiple operation rate limits can be specified. For example, specifying values of "100/s" and "50000/h" will allow clients to burst up to 100 operations per second, but not more than 50,000 operations in an hour.
- **connection-operation-rate-exceeded-behavior** – Specifies the action that the server should take if a client exceeds any of the `maximum-connection-operation-rate` values. By default, the server will reject the operation with a result code of 51 (busy). An option to terminate the client connection is also available.
- **maximum-policy-operation-rate** – Specifies the maximum operation rate across all connections associated with the client connection policy. This is useful in cases where a policy is dedicated to clients associated with a particular application for the purpose of limiting the aggregate request rate for that application.
- **policy-operation-rate-exceeded-behavior** – Specifies the behavior that the server should exhibit if the `maximum-policy-operation-rate` is exceeded. This has the same set of options as the `connection-operation-rate-exceeded-behavior` property.

Enforce search limits with client connection policies

The following settings are used to limit the search parameters of clients for which the client connection policy applies:

- **maximum-search-size-limit** – Specifies the maximum search size limit (the maximum number of entries that can be returned by a search operation). This can be used to enforce a smaller limit for clients, but will not increase a client's size limit.
- **maximum-search-time-limit** – Specifies the maximum length of time that the server can spend processing a client search operation. This can be used to enforce a smaller time limit for clients than they would otherwise have, but will never increase a client's time limit.
- **maximum-search-lookthrough-limit** – Specifies the maximum search lookthrough limit (the maximum number of entries that the server can examine during the course of processing a search, regardless of whether those entries are actually returned to the client). This can be used to enforce a smaller lookthrough limit, but will not increase a client's lookthrough limit.
- **allow-unindexed-searches** – Allows clients to request unindexed search operations. Unindexed searches can occupy worker threads for long periods of time. They can also be used to retrieve large amounts of data from the server. It is generally recommended that access to request unindexed searches be limited to administrators, or operations requested from a specific set of systems.

Restrict request types with client connection policies

Client Connection Policies provide a number of properties that can be used to restrict the type of requests that clients are allowed to issue. They include:

- **allowed-operation** – Specifies the operations that are allowed for clients associated with the policy. Allowed values include `abandon`, `add`, `bind`, `compare`, `delete`, `extended`, `modify`, `modify-dn`, and `search`.
- **allowed-request-control** – Specifies object IDs of controls that clients are allowed to use in requests. Any request containing one or more controls not in this list is rejected. If no `allowed-request-control` values and no `denied-request-control` values are specified, clients can request any controls.
- **denied-request-control** – Specifies object IDs of controls that clients are not allowed to use in requests. If a client request includes a control with an object ID that matches a denied value, that request is rejected. If no `allowed-request-control` values and `no-denied-request-control` values are specified, clients are allowed to request any controls.

- **allowed-extended-operation** – Specifies object IDs of extended requests that clients are allowed to send. If one or more values are specified, any extended request not contained in this list is rejected. If no `allowed-extended-operation` and no `denied-extended-operation` values are specified, clients are allowed to submit any extended request.
- **denied-extended-operation** – Specifies object IDs of extended requests that clients are not allowed to send. If a client sends an extended request listed as denied, that request is rejected. If no `allowed-extended-operation` and no `denied-extended-operation` values are specified, clients can submit any extended request.
- **required-operation-request-criteria** – Specifies a request criteria object that is required to match any request submitted by the client. If a value is specified, and the client submits a request that does not match that criteria, the request is rejected.
- **prohibited-operation-request-criteria** – Specifies a request criteria object that must not match requests submitted by the client. If a value is specified and the client submits a request that matches that criteria, the request is rejected.

Allow and deny client IP addresses

The PingDirectory Server provides several means to limit client access using connection handlers, client connection policies, or operational attributes.

Allow and deny client IP addresses using connection handlers

Limit the client IP addresses using the LDAP or LDAPS connection handlers. The connection handlers provide two properties that can be used to mitigate denial-of-service attacks:

- **allowed-client** – Specifies the set of allowable address masks that can establish connections to the handler.
- **denied-client** – Specifies the set of address masks that are not allowed to establish connections to the handler.

Allow and deny client IP addresses using client connection policies

Access can be restricted by configuring a new client connection policy, then creating a new connection criteria and associating it with the connection policy. Connection criteria define sets of criteria for grouping and describing client connections based on a number of properties, including the protocol, client address, connection security, and authentication state for the connection.

Limit client IP addresses by specifying the following properties in client connection policies:

- **included-client-address** – Specifies an address mask that identifies a set of clients that should be included in the connection criteria.
- **excluded-client-address** – Specifies an address mask that can be used to specify a set of clients that should be excluded from the connection criteria.

Allow and deny client IP addresses using an operational attribute

Specified address masks can be limited using the following operational attribute:

ds-auth-allowed-address – Specifies the set of addresses from which a user is allowed to authenticate. Values can be address masks, which can include individual IP addresses or resolvable names, addresses with wildcards, CIDR address ranges, or IP addresses with subnet masks. This attribute can also be used to ensure that accounts used by external systems are only used by those external systems.

Data Breach Prevention

The PingData Platform provides features that mitigate data breach and trawling attacks such as on-disk encryption, sensitive attributes, password storage schemes, access control rules, and client connection policies. Flexible logging capabilities make it possible to record operations that involve large amounts of data, which can be investigated after a breach.

Client connection policies are effective against trawling attacks, limiting the resource capabilities for certain clients using connection criteria. For example, limits can be enforced on the number of requests a client can issue, the rate at which the client can make requests, the types of filters clients are allowed to issue, and on substring length. Server-wide or per-user constraints can be defined on the number of entries that can be examined and/or returned per search, the length of time the server spends processing a search, and whether to process expensive unindexed searches.

Global configuration options for on-disk encryption

A number of Global Configuration properties can set on-disk encryption to protect against data breach and trawling attacks. Data encryption is only applied to the on-disk storage for a PingDirectory Server instance. It does not automatically protect information accessed or replicated between servers, although other mechanisms provide that protection (SSL, StartTLS, SASL). Client communication using either SSL or StartTLS encryption ensures that the data is protected from individuals or applications able to eavesdrop on network communication. This communication security can be enabled independently of data encryption.

The global configuration properties designed to set up on-disk encryption include the following:

- **encrypt-data** – Specifies whether data encryption should be enabled in the server for all components that support it, including certain backends, like the LDAP changelog backends, and the replication database. If this is enabled, the server must be configured with at least one encryption setting definition.
- **encryption-settings-cipher-stream-provider** – Specifies the cipher stream provider used to read from and write to the encryption settings database, which is also encrypted. If no cipher stream provider is configured, the server uses a hard-coded algorithm for accessing encrypted data. If data encryption is enabled, a custom cipher stream provider should be defined so that it uses a non-default mechanism for accessing the contents of the encryption settings database.
- **verify-entry-digests** – Specifies whether the server should automatically verify any cryptographic digests that can exist in the encoded representation of entries during the course of decoding them. The generation of entry digests is controlled by the hash-entries configuration property in backends that support this capability. The process of generating these digests can be controlled independently of their verification. Verification can be enabled only if database corruption is suspected.

Implement sensitive attributes

Some attributes contained in user data need additional protection beyond what access controls provide. This is important for those users who are not subject to access control processing (those users with the `bypass-acl` or `bypass-read-acl` privilege), but for which access to certain attributes should still be restricted. The PingDirectory Server's sensitive attribute mechanism can be used to accomplish this.

Sensitive attributes are used to restrict certain kinds of access to a specified set of attributes, or to ensure that they can only be accessed over a secure connection. Sensitive attributes can be defined as part of the global configuration or in client connection policies.

- **Global sensitive attributes** – Are applied across all client connection policies, except those that explicitly exclude them using the `sensitive-attribute` property. Administrators can configure this setting using the `dsconfig` tool.
- **Sensitive attributes** – Are configured using the `sensitive-attribute` property in the client connection policy configuration object. The `exclude-global-sensitive-attribute` property can be used to indicate that certain global sensitive attributes should

not be in effect for clients associated with that client connection policy. It is possible for the same attribute type to be referenced in multiple sensitive attribute definitions. In this case, the server enforces the most restrictive combination of these sensitive attribute definitions during processing.

Global configuration for sensitive attributes

The global configuration property available for use with sensitive attributes include the following:

sensitive-attribute – In the global configuration, specifies the set of sensitive attribute definitions, which is automatically applied across all client connection policies. However, individual client connection policies can exclude one or more global sensitive attribute definitions if desired. See the section on sensitive attributes for more information.

Client connection policy properties for sensitive attributes

The client connection policy properties available for sensitive attributes include the following:

- **sensitive-attribute** – Specifies the set of sensitive attribute definitions that are in effect for clients associated with the client connection policy. Sensitive attributes can be used to prevent access to specified attributes, or to restrict them so that they can only be accessed over secure connections.
- **exclude-global-sensitive-attribute** – Specifies the set of global sensitive attribute definitions that is excluded for clients associated with the policy. For example, if most clients should be prevented from retrieving passwords, but the Data Sync Server needs to be able to retrieve encoded passwords over a secure connection, a global sensitive attribute can prevent password access, and the policy used by the Data Sync Server can exclude that global policy.

Configuration properties for sensitive attribute definitions

There are a number of properties used to configure sensitive attributes with the `dsconfig` command-line tool:

- **attribute-type** – Specifies the names of the attributes targeted by this definition.
- **include-default-sensitive-operational-attributes** – Specifies whether the server should consider certain operational attributes to be sensitive. This includes the `ds-sync-hist` attribute, which is used for holding information for replication conflict resolution processing. Since this attribute can include previous values for attributes, it could

contain values for sensitive attributes, and therefore it needs to provide the same level of protection as explicitly-defined sensitive attributes.

- **allow-in-returned-entries** – Specifies whether the server should allow sensitive attribute values to be returned to the client in search result entries. The value for this property can be one of the following:
  - **allow** – Include this attribute in search result entries if it is permitted by access control and other parameters.
  - **suppress** – Exclude this attribute in search result entries, regardless of whether the user has permission to access it in other ways.
  - **secure-only** – Include this attribute in search result entries, but only for clients communicating with the server over a secure connection.
- **allow-in-filter** – Specifies whether the server should allow clients to request search operations with a filter that targets any of the sensitive attributes. The value for this property can be one of the following:
  - **allow** – Allow any search containing a filter targeting a sensitive attribute.
  - **reject** – Reject any search containing a filter targeting a sensitive attribute.
  - **secure-only** – Allow any search containing a filter targeting a sensitive attribute, but only for clients communicating with the server over a secure connection.
- **allow-in-add** – Specifies whether the server should allow clients to attempt to create entries that include the sensitive attribute. The value can be one of the following:
  - **allow** – Allow any add requests.
  - **reject** – Reject any add requests.
  - **secure-only** – Allow any add requests for clients communicating with the server over a secure connection.
- **allow-in-compare** – Specifies whether the server should allow clients to attempt to perform a compare operation which targets the sensitive attribute. The value can be one of the following:
  - **allow** – Allow any compare requests.
  - **reject** – Reject any compare requests.
  - **secure-only** – Allow any compare requests for clients communicating with the server over a secure connection.
- **allow-in-modify** – Specifies whether the server should allow clients to attempt to modify sensitive attributes. The value can be one of the following:
  - **allow** – Allow any modify operations.
  - **reject** – Reject any modify.
  - **secure-only** – Allow any modify operations for clients communicating with the server over a secure connection.

Password storage schemes

Many news-worthy security breaches center around stealing large numbers of stored, encoded passwords. To protect passwords, the PingDirectory Server enables a variety of password storage schemes. Password storage schemes are used to perform encoding, and to verify that clear-text passwords provided in a bind request match the encoded representation stored in a user's entry.

There are a number of different password storage scheme implementations to obscure user passwords. Many of them use one-way digests that encode passwords in a manner that cannot be reversed, so that even if someone discovers the encoded representation of a password, they cannot easily determine the clear-text value used to generate it. Many of them use salts to provide better resistance to attacks using pre-encoded dictionaries. Others use reversible encryption that makes it possible for the server to determine the clear-text value, but it is still difficult for users to determine the clear-text version of that password.

Some implementations use trivial encodings that do not offer any real protection and are only supported for compatibility with third-party applications. Schemes that use reversible encryption should be avoided unless clients need to perform SASL authentication with the DIGEST-MD5 or CRAM-MD5 mechanisms. Storage schemes using one-way digests are recommended for best security.

The password storage schemes supported by the PingDirectory Serverinclude:

- **AES** – Uses the AES reversible encryption algorithm.
- **Base64** – Uses base64 encoding, which obscures password values but does not provide any real level of protection.
- **Blowfish** – Uses the Blowfish reversible encryption algorithm.
- **Bcrypt and scrypt** – Uses thousands of cryptographic computations in the course of encoding a password to make the process of encoding a password relatively expensive. These require the free, open source Bouncy Castle cryptographic library available at `https://www.bouncycastle.org/download/bcprov-jdk15on-157.jar`.
- **Clear** – Stores the clear-text representation of the password with no encoding or obfuscation.
- **Crypt** – Uses the UNIX crypt mechanism. The server supports three variants of this mechanism: the "classic" crypt digest which is weak, and two stronger 256-bit and 512-bit SHA-2 digests, which are extremely resistant to attacks.

- **MD5** – Uses an unsalted form of the MD5 message digest algorithm.
- **PBKDF2** – Uses PBKDF2 key derivation function as described in the PKCS#5 specification (RFC2898). PBKDF2 is the preferred option for a strong password storage scheme that involves multiple rounds of cryptographic processing, and does not require third-party components (like Bcrypt and scrypt).
- **RC4** – Uses the RC4 reversible encryption algorithm.
- **Salted MD5** – Uses a salted form of the MD5 message digest algorithm.
- **Salted SHA1** – Uses a salted form of the SHA-1 message digest algorithm.
- **Salted SHA256** – Uses a salted form of the 256-bit SHA-2 message digest algorithm.
- **Salted SHA384** – Uses a salted form of the 384-bit SHA-2 message digest algorithm.
- **Salted SHA512** – Uses a salted form of the 512-bit SHA-2 message digest algorithm.
- **SHA-1** – Uses an unsalted form of the SHA-1 message digest algorithm.
- **Third-Party Enhanced** – Created in third-party code using the Server SDK. These storage schemes may have access to the user entry so that content from that entry can be used in the password encoding and/or validation process if needed.
- **Third-Party** – Created in third-party code using the Server SDK.
- **3DES** – Uses the 3DES reversible encryption algorithm.

Strongest supported password storage schemes

The strongest of the supported storage schemes are the PBKDF2, Bcrypt, and scrypt schemes. Each of these schemes performs thousands of cryptographic computations in the course of encoding a password (and the scrypt scheme also relies on memory consumption and memory access latency) to make the process of encoding a password relatively expensive. In most cases, this expense is not significant for normal authentication processing, but it is very effective at impeding brute force and dictionary attacks, even if the attacker has access to the encoded representation of a password. The Bcrypt and scrypt password storage schemes requires the free and open source Bouncy Castle cryptographic library (`https://www.bouncycastle.org/download/bcprov-jdk15on-157.jar`), which is not included with the server. The PBKDF2 scheme does not require any additional library to be installed.

The SHA-2 variants of the crypt password storage scheme use similar techniques for encoding passwords that are resistant to attack. However, the crypt scheme also supports substantially weaker variants that may permit the inadvertent use of weakly-encoded passwords. It is recommended that the crypt scheme only be used if it is necessary for compatibility with other systems. If that compatibility is only needed for migrating data, the crypt scheme can be marked as deprecated so that passwords encoded with it are automatically re-encoded with a stronger scheme the first time the user authenticates with that password.

The password storage schemes that use salted variants of the 256-bit, 384-bit, and 512-bit SHA-2 digests are also considered strong, although these schemes only apply the digest one time when encoding a password. Brute force and dictionary attacks against passwords encoded with these schemes can be conducted much more quickly than with the PBKDF2, Bcrypt, or scrypt schemes. However, with strong passwords, these attacks are still very expensive to conduct.

Formats for encoded passwords

The PingDirectory Server supports two different formats for representing encoded passwords: `userPassword` and `authPassword`. The `userPassword` syntax is widely supported by directory servers. It contains the name of the scheme in curly braces followed by an encoded representation of a password, like `{SSHA}7z9Lzvdk3ACw9ITe/yEV5iES5ADcbdZcj3PFZQ==`. The `authPassword` syntax, as described in RFC 3112, looks like `SHA1$wrOlEecRfV0=$3as1EB+TkA85WWcOugSLsoghU90=`, and is not supported by all servers.

Each password policy must have at least one default password storage scheme. When the server is asked to store a password provided in clear-text, it encodes it using each of the storage schemes before actually storing it in the database. Multiple default schemes can be used, but this should only be done for cases in which clients need to retrieve the encoded password from the server and verify it themselves rather than using an LDAP bind operation, or if there are multiple clients that require conflicting schemes for offline verification.

Deprecated password storage schemes

Password Policies can also be configured with zero or more deprecated password storage schemes. Deprecated storage schemes provide a mechanism for retiring old password schemes that had previously been used but are no longer needed and are not considered secure. When a user authenticates to the server with a password encoded in any of the deprecated storage schemes, the deprecated encodings are removed and the password is re-encoded with the current password storage scheme.

Limit search results

Data trawling attacks are characterized by broad searches that attempt to retrieve as much data in one operation as possible. Allowed searches can be limited and the server can be prevented from returning suspiciously large result sets.

Global configuration property that limits search results

The PingDirectory Server provides a global configuration property that limits search results to mitigate against data trawling attacks:

size-limit – Specifies the maximum number of entries returned in a single search operation. This limit can be overridden on a per-user basis by including the `ds-rlim-size-limit` operational attribute in the user's entry (in a real or virtual attribute), and should be reserved for users that have a legitimate need to retrieve large numbers of entries.

Client connection policies that limit search results

The PingDirectory Server provides Client Connection Policies that limit search results to mitigate against data trawling attacks:

- **allowed-filter-type** – Specifies the kinds of filter components that clients are allowed to use in search operations. If a client sends a search filter containing one or more components with a filter type that is not allowed, the search is rejected.
- **minimum-substring-length** – Specifies the minimum number of consecutive non-wildcard characters that must be present in a substring search filter component in order for a search request to be allowed. If a search request contains a filter with a substring component that does not have at least this many consecutive non-wildcard characters, the search is rejected.
- **maximum-search-size-limit** – Specifies the maximum search size limit (the maximum number of entries that can be returned by any search operation) for clients using the policy. This property can be used to enforce a smaller limit for clients than they already have, but will never increase a client's size limit.
- **maximum-search-time-limit** – Specifies the maximum length of time that the server can spend processing any search operation for clients using the policy. This property can be used to enforce a smaller time limit for clients than they already have, but will never increase a client's time limit.
- **maximum-search-lookthrough-limit** – Specifies the maximum search lookthrough limit (the maximum number of entries that the server can examine during the course of

processing a search, regardless of whether those entries are actually returned to the client) for clients using the policy. This property can be used to enforce a smaller lookthrough limit for clients than they already have, but will never increase a client's lookthrough limit.

Operational attributes that limit search results

The PingDirectory Server provides operational attributes that limit search results to mitigate against data trawling attacks:

- **ds-rlim-size-limit** – Specifies the search size limit that should be enforced for the user. If a value is not specified, the user inherits the default size limit specified in the global configuration.
- **ds-rlim-time-limit** – Specifies the search time limit (in seconds) that should be enforced for the user. If a value is not specified, the user inherits the default time limit specified in the global configuration.
- **ds-rlim-lookthrough-limit** – Specifies the search lookthrough limit that should be enforced for the user. A value of zero indicates that no lookthrough limit should be enforced for that user. If this is not specified, the user inherits the default lookthrough limit specified in the global configuration.

Searches involving sensitive attributes

The PingDirectory Server supports additional restrictions that can be applied to specific attributes. Several of these are useful in the context of making search requests more secure, such as `allow-in-returned-entries` and `allow-in-filter`.

Restrict access to certain controls

Another way to protect against a data breach is to limit large searches spanning multiple requests, which is a risk if an attacker has access to certain controls, such as the simple paged results and virtual list view. The simple paged results control can be used with a search operation to iterate sequentially through the search results a page at a time. The virtual list view control is similar, except that the results are presented in sort order that enables the server to return a subset of entries.

Using client connection policies to restrict access to controls

By default, these controls are granted to users specifically through access control rules or for those users who have the `bypass-acl` privilege. Access to these controls can be limited by configuring restrictions on what controls are allowed through the Client Connection Policy. The properties that limit access to controls are as follows:

- **allowed-request-control** – Specifies the object IDs of the controls that clients can include in requests. Only specified controls can be included in the requests.
- **denied-request-control** – Specifies the object IDs of the controls that clients are not allowed to include in requests.
- **required-operation-request-criteria** – Specifies a request criteria object that is required to match all requests submitted by clients. If a client submits a request that does not satisfy this request criteria object, that request is rejected.
- **prohibited-operation-request-criteria** – Specifies a request criteria object that must not match requests submitted by clients. If a client submits a request that satisfies this request criteria object, that request is rejected.

Note

These properties do not grant a user the ability to use these controls, which is done using access control rules. They do provide more granular control over circumstances in which they can be used, and also enforce these restrictions for users with the `bypass-acl` privilege.

Restrict access to the directory information tree with client connection policies

Client connection policies can control the portions of the Directory Information Tree (DIT) that clients can access. This is configured through the set of subtree views associated with the policy. Subtree views associate a base DN with the logic used to process requests within that portion of the DIT. Some of these subtree views can be automatically created by the server for those associated with local backends, but some of them can be manually created by administrators, especially in the PingDirectoryProxy Server, for views that pass through operations to backend servers.

The configuration properties used to limit access to portions of the DIT include:

- **include-backend-subtree-views** – Indicates whether the policy should automatically include subtree views for local backends defined within the server. This should only be set to `false` in the PingDirectoryProxy Server, so that it only allows access to proxied data but prevents access to local content like the server root DSE, schema, configuration, and monitor data.
- **included-backend-base-dn** – Specifies the base DNs for backends whose information should be made available to clients. If base DNs are specified, clients associated with the

policy are only allowed to access data within those DNs. If no backend base DNs are specified as included or excluded, clients can access all content in all backends.

- **excluded-backend-base-dn** – Specifies the backend base DNs for content that clients should be prevented from accessing. If no backend base DNs are specified as included or excluded, clients can access all content in all backends.
- **included-backend-server-pass-through-subtree-views** – Used in the PingDirectoryProxy Server to expose access to each of the backend servers. This can be useful for administrators who can only access the backend servers through the PingDirectoryProxy Server but need to interact with a specific server without worrying about how requests are routed through load balancing and entry balancing. If enabled, each backend server is available through `ds-backend-server={serverID}`. For example, for a server with ID `ds1.example.com:389`, the `cn=monitor` entry could be accessed with a DN of `cn=monitor,ds-backendserver= ds1.example.com:389`.
- **subtree-view** – Used in the PingDirectoryProxy Server to control access to proxied data sets for clients. It can be useful for accessing data sets through proxying, entry balancing, and failover request processors.

LDAP injection attacks

LDAP Injection attacks are used to manipulate the search filters from a client application to gain access to an underlying directory database. Where a SQL query can be used to destroy or alter data, LDAP injection only offers the possibility of providing unexpected read access to the data. Also, the LDAP syntax used for expressing search filters prevents many kinds of injection attacks so that an attacker cannot increase the scope of data returned.

To prevent injection attacks, make sure that all clients sanitize the user input that can be included in search requests. Many LDAP client APIs (including the LDAP SDK for Java) provide ways of creating search filters that do not require using the string representation, and therefore do not allow unexpected input to turn into one or more additional search filter components.

If a client succeeds in performing an LDAP injection, the intended result will be either to get the server to reveal a large amount of data, or to reveal specific sensitive information. The first scenario is considered a data trawling attack, which can be prevented with any of the previously listed configuration options. The second scenario can be addressed by processing requests with an appropriate authorization identity, and by ensuring that the server is configured (through the use of access controls, sensitive attributes, client connection policy restrictions) to only return information that users have a right to retrieve.

Man-in-the-middle attack prevention

The man-in-the-middle attack works by establishing connections between the PingDirectory Server and the client and relaying messages between them as an intermediary host. The client and target PingDirectory Server are unaware of this eavesdropping attack, as each believes it is communicating with the other. This attack can intercept sensitive data, manipulate data transmission, and inject malicious code.

PingData servers can mitigate these attacks by ensuring that all connections are secure. For example, clients can connect to the PingDirectory Server over SSL or StartTLS, which enables determining if the certificate presented can be trusted. Secure naming services like DNSSEC can also help prevent the kinds of DNS hijacking attacks that are frequently used to trick clients into establishing connections to the wrong systems.

Secure system-to-system network connections

Another critical aspect of network security lies in making sure that communication occurs between the intended systems. If a client can be tricked into establishing a connection with an untrusted system, then it could compromise the client’s credentials or enable a man-in-the-middle attack, in which the untrusted system could alter traffic between the client and server or inject completely new requests.

Consider using the following external tools (they are not PingData Platform features) to mitigate these problems:

- **Use DNSSEC** – If available, DNSSEC should be used to prevent DNS hijacking in a way that could cause clients to receive the wrong addresses for servers.
- **Strong TCP sequence numbers** – Use strong TCP sequence numbers to avoid existing sessions from being hijacked.
- **Reject source-routed packets** – Though rarely used, source routing allows the sender of a packet to specify which route the packet should take to its destination.
- **Reject ICMP redirects** – Internet Control Message Protocol (ICMP) redirects are used by routers to notify host systems that a better path is available to its destination. Reject ICMP redirects avoid traffic routed through untrusted systems.

- **Prevent Eavesdropping** – Any inter-system communication should be encrypted to ensure data integrity and confidentiality. For PingData servers, internal configuration options address this concern. However, communication to remote filesystems can also include sensitive data that needs to be protected. Encryption for these services can be configured on an individual basis, or IPsec can be configured to ensure that all communication between systems is encrypted. IPsec can also encrypt communication for services that do not provide their own encryption support.

Features that reduce the risk of network address-spoofing

The Ping Identity product family provides a number of features that reduce the risk of network address-spoofing:

- **Use a Global Configuration Property** – The global configuration property, `networkaddress-cache-ttl`, specifies the maximum length of time that the JVM should cache the IP address for a resolved hostname.
- **Use Custom Post-Connect and Post-Disconnect Plug-ins** – The Server SDK can be used to develop custom post-connect and post-disconnect plug-ins. Post-connect plug-ins are invoked when the server accepts a new client connection and are used to terminate that connection if it is determined that it should not be allowed. Post-disconnect plug-ins are invoked just after an existing connection is closed, whether that closure is initiated by a client or by the server.
- **Set Up Credentials for External Servers** – If the directory must access content on an external server, credentials for that server must be supplied. These credentials can be in the form of a password, or a certificate. Because these credentials are often for accounts with elevated privileges, they need to be protected. The PingDirectory Server encrypts the passwords it uses to access external systems and the PINs used to interact with a certificate keystore. Access to the configuration file, archived configurations, and the configuration audit log should be carefully protected.

Credentials used to authenticate to external servers should not be shared by other applications. If possible, the target server should also be configured to accept those credentials only from PingData servers. This ensures that even if those credentials are compromised, they can only be usable from PingData servers.

Chapter 4: Host system protection

Securing a directory environment requires securing the systems and networks on which the servers are running. Even if a server is locked down, someone who has access to the system on which the server is running may still be able to obtain sensitive data.

There is much information available about how to secure systems and networks. Work with operating system vendors to understand the security features and best practices specific to those platforms.

Most PingData server deployments exist on UNIX-based operating systems. The principles addressed in this guide are suitable for any operating system, but some content is more relevant for UNIX-based operating systems.

Topics include:

The PingData environment on multiple operating systems

In many environments, selecting an operating system is a relatively straightforward process. Most modern operating systems can be configured securely. PingData servers are well-suited to deployments installed across multiple operating systems because they do not include system-specific dependencies. The server software itself is pure Java. The data that it stores does not depend on whether the CPU is bigendian or little-endian, the operating system specifics, or what end-of-line sequence is used. A backup taken on one operating system can be restored on another operating system, and in most cases, an entire server instance can be moved from one operating system to another.

Some operating systems include extensions that add additional security capabilities. These can provide advanced security features, including mandatory access control, role-based access control, and labeled security. In addition, many of the security concepts from these operating systems have been incorporated into the PingData server products.

Minimize software and running services

Operating systems often come with a large amount of software installed. Each application or command on a system is potentially a security hole that could provide unauthorized users a way to get into the system. Software that is not needed to run the JVM or PingData server software can be safely removed. Consult the operating system(s) documentation to determine the applications that can be removed.

For software that cannot be removed from the system, reduce the likelihood that it can be exploited. Any unnecessary network services should be disabled, and any network daemons which must run, but are not needed outside the system, should be configured so that they are not accessible to external clients. When possible, services should be configured to run as a non-root user with as few rights as possible.

All nonessential network services should be disabled, and firewall software should be used to ensure that a service that is disabled cannot be accessed.

Keep systems patched

For software that cannot be removed from the system, it should be updated regularly so that vulnerabilities are fixed as quickly as possible. However, there have been instances in which security patches caused unforeseen problems. It is strongly recommended that a testing environment be used to test patches and updates prior to production.

Monitor security-related or operating system mailing lists. These offer timely security information based on a wide range of use. It can take a significant length of time for an operating system vendor to prepare and test a patch for a problem, leaving systems vulnerable during the duration. The sooner these problems are known, the sooner corrective action can be taken. After fixes are released, review industry reports for any problems introduced by the patches.

Virtualization best practices

In environments that run multiple network services on the same system, it may be useful to use virtualization to separate those services. In some cases, it can be useful to isolate a service from the rest of the operating system.

The primary advantage of separation is that it limits the effects of a vulnerability in one of the services. A second advantage is that if an attacker does gain control over a service, the compartmentalization can prevent the breach from escaping the boundaries of the container.

A third advantage to virtualization is that it can separate security monitoring tools from the containers in which the servers are running. If an attacker gains access to the container in which a PingData server is running, the attacker will not have access to the monitoring process. This can also be useful for server logging. If a `syslog` daemon is run in one container and the PingDirectory Server in another, the server log can be run over a private network available between those containers.

For a heavy virtualization option like VMware or hardware partitions like LDOMs, each container will be required to have its own operating system installed, which must be secured and maintained. Lightweight virtualization options that use only one operating system instance may be more efficient.

Maintain the Java Virtual Machine

A recent version of the JVM should be used to ensure that known security holes have been patched. However, there are known problems with some recent JVM versions, and it is best to contact an authorized support provider for assistance in selecting the best version.

Just as with operating systems, consider running JVMs from different vendors to mitigate the risk of bugs and security holes that can be discovered. On some operating systems, there may not be many options.

Configure strong authentication for administrators

The mechanism for authenticating to the system should be as secure as possible. Consider who really needs access to the system to minimize the number of credentials that could be compromised. Each authorized user should have a distinct set of credentials, so that the actions of each can be audited. If one leaves or no longer needs access to the system, that account access can be easily revoked without impacting others. This also makes it easier to enforce policies that require individuals to change their credentials on a regular basis without having to coordinate the change among multiple users.

If possible, consider credentials either instead of, or in addition to passwords. SSH keys work well. They are relatively straightforward to set up and use, do not need to be remembered, and cannot be guessed. The keys themselves can be protected with passwords, adding another layer of security. If passwords alone must be used, configure the system to require strong passwords and ensure that passwords are encoded with a mechanism that is resistant to attacks.

If a user account is compromised or its owner leaves, the account should be terminated as soon as possible. Normally, this is best left to a centralized naming service like LDAP, but this is not recommended for the systems responsible for providing the directory service itself. Local file-based accounts are most reliable. Keep a current list of all users with access to these systems, and all systems they can access with those credentials, so they can all be quickly accessed and revoked if necessary.

Ping IdentityMinimize administrative account capabilities

Each system account should be as limited as possible while still allowing its owner to accomplish necessary tasks. System administrators need full access to the system, but administrative accounts should not be allowed to directly authenticate to the system.

The accounts of users authorized to log into the system and assume the administrative role should be restricted to authentication, and assuming the administrative identity. Ideally, they should have limited access to the server filesystem, and should not be able to see processes owned by other users. Restrict the set of commands they can execute using profiles or a restricted shell.

Use system logging and auditing

Most operating systems provide an audit mechanism that records detailed information about system events. This can include coarse information, like recording user login and logout events, or more detailed information like recording each time a user opens or closes a file. Auditing can provide vital information to diagnose problems or investigate security breaches. Make sure auditing and logging are tuned properly to avoid saving too much data, which can hamper problem solving and reduce system performance.

Chapter 5: Filesystem security

If an attacker does gain access to a server system, the next line of defense is the restrictions the system enforces for access to data on the server. This includes database content, configuration files, log files, backups, LDIF exports, and other information.

Topics include:

Filesystem protections

The most basic forms of filesystem protection are file permissions and filesystem encryption. Any portion of the filesystem containing sensitive data should be accessible only to the account used to run the server. In the default installation, all components of the server reside in server root. When the server archive is unzipped, which should be done with the account used to run the server, the server root directory will have 0700 permission. The content below it cannot be accessed by any other account on the system, except those not subject to filesystem access restrictions, like root. Further, directories used to hold database files have permissions of 0700, and access and error log files are given permissions of 0600. If some components of the server are moved to other filesystems, then permissions and ownership should be set on those paths to ensure that it is appropriately protected.

Another form of security is filesystem encryption. Although PingData servers provide the ability to encrypt some content, an additional level of protection may be obtained by encrypting the entire filesystem. Encryption generally does not add much value for a mounted filesystem, since it appears unencrypted to applications that use it.

Filesystem auditing software can also be used to identify questionable use of file permissions and SUID/SGID bits, and keep a record of all filesystem content changes. This is valuable for files that are part of PingData server installations. Though log and database files change frequently, changes to jar and configuration files are less frequent. Any change in operating system binaries and configuration files should be tracked.

Remove Java encryption security restrictions

Although the Java runtime environment includes the Java Cryptography Extensions (JCE) library for performing encryption, hashing, signing, and other kinds of cryptographic operations, the strength of the encryption that can be used is limited by default. This restriction is enforced for legal reasons, because U.S. law forbids exporting strong encryption capabilities to some countries. If possible, update the installation to remove these restrictions.

To do this, search for and download the "Java Cryptography Extension (JCE) Unlimited Strength Policy Files 6." Follow the instructions in the `README.txt` file included with the package.

Manage the encryption settings database

Encryption can be enabled during setup for all PingData and PingDirectory servers. Encryption for directory data, backups, LDIF exports, and log files can be enabled during setup by providing or generating an encryption key with a passphrase. Using a passphrase (obtained interactively or read from a file) is the recommended approach for new deployments, and you should use the same encryption passphrase when setting up each server in the topology.

If not enabled during setup, data encryption can be configured by creating an encryption-settings definition to specify the cipher transformation that should be used to encrypt the data, and encapsulate the encryption key. The `encryption-settings` command-line tool can be used to manage the encryption-settings database.

Because the encryption-settings database contains the encryption keys used to protect server data, the encryption-settings database is itself encrypted. By default, the server will derive a key to use for this purpose, but the logic used to access the encryption-settings database with a cipher stream provider should be customized. The Server SDK provides an API that can be used to create custom cipher stream provider implementations, but the server also provides one that will obtain the key from a custom PIN file.

Each server in a replicated environment will maintain its own encryption-settings database. If data encryption is enabled, each replica uses its own encryption settings to encrypt updated entries. Though it is not necessary for servers to share the same encryption-settings definitions, it is necessary if the server needs to be able to restore a backup containing encrypted data on a different instance than the server from which it was originally created. It is recommended that an encryption-settings definition be created on one server, exported, and imported on all other servers.

Supported cipher stream providers

PingDirectory Server supports four types of cipher stream providers, which are used to obtain cipher input and output streams for reading and writing encrypted data. These are advanced configuration properties, listed in the *PingData PingDirectory Server Reference*.

Cipher Stream Providers

Cipher Stream Providers	Description
Amazon	Amazon Web Services (AWS) Key Management Service (KMS) cipher stream provider.
Default	Default cipher stream provider that uses a hard-coded default key.
File-based	Reads a specified file to obtain a password that is used to generate cipher streams for reading and writing encrypted data.
Third-party	Provides cipher stream provider implementations that are created in third-party code by using the Server SDK.
Vault	Vault cipher stream provider.
Wait-for-paraphrase	Causes the server to wait for an administrator to enter a passphrase that is used to derive the key for cipher streams. To supply the passphrase to the server, run the `encryption-settings supply-passphrase` command.

If you use AWS KMS as a cipher stream provider, the system performs the following actions after the key Amazon Resource Name (ARN) is used:

- Saves the key ARN to the cipher stream provider configuration
- Embeds the key ARN in the encrypted passphrase file

If the same passphrase file is used subsequently to configure a new cipher stream provider, the system uses the embedded ARN, even if a new one is provided.

Configure data encryption

By default, the server stores information in the database in a compact encoded form, intended to minimize the amount of space required to hold that data on disk and in memory. Although this encoding makes the data harder to extract, it is still possible for an attacker to get the data, and potentially decode the database files on a different server.

To address this problem, the PingDirectory Server enables encrypting the data after it has been encoded, so that only an individual with access to both the database files and the encryption key can determine the content. As long as the encryption keys are carefully protected, the database content remains secure.

The PingDirectory Server relies on data encryption rather than attribute encryption. Instead of indicating which attributes should be encrypted, the server is enabled to encrypt all data. This has several advantages over encrypting individual attributes, including:

- **Simplicity** – Enable data encryption in the global configuration, and it will be applied where the server supports it. This includes user data, the replication database, and LDAP changelog.
- **Better Protection** – The problem of selecting attributes and omitting something that can contain sensitive information is eliminated. This is especially true for operational attributes, which can contain portions of user data in non-obvious ways. For example, attributes used for replication conflict resolution can have data from any attribute in the entry.
- **Smaller Database Size** – Encrypting everything makes the encoded representation smaller than if the database contained a mix of encrypted and unencrypted content.
- **Efficiency** – Encrypting all of the data as a single encryption operation is more efficient than performing multiple encryption operations for individual attributes.

Once the server has been configured with at least one encryption settings definition, data encryption can be enabled with the following:

```
$ dsconfig set-global-configuration-prop \
  --set encrypt-data:true
```

Data encryption can be enabled during setup, or after. Any writes performed after configuration will be encrypted. Existing entries remain unencrypted until they are updated. To ensure that all data is immediately encrypted, it is recommended that the server is stopped, the data is exported to LDIF (optionally encrypting that LDIF file as described below), and the data is re-imported into the server. This process can also be used to re-encrypt the data if the server is updated with a new preferred encryption settings definition.

Encrypt passphrase files

Using an encrypted passphrase or `tools.properties` file enables you to have credentials available for use by the server and command-line tools, but not stored in the clear. These files should be encrypted with the `encrypt-file` tool, including:

- Certificate keystore and truststore PIN files. When setting up an instance with encryption enabled and at least one of SSL or StartTLS enabled, the installer will automatically encrypt the PIN files for the config/keystore, config/truststore, and config/ads-truststore certificate databases.
- Passphrase files specified by command-line arguments. For example, most LDAP tools offer `--bindPasswordFile`, `--keystorePasswordFile`, and `--truststorePasswordFile` arguments.

- The `config/tools.properties` file, which will automatically be used to obtain a default set of arguments for most command-line tools. Alternately, the `--propertiesFilePath` argument can be used to specify an alternate properties file.

These files should be encrypted with the following considerations:

- If the file is encrypted with a key obtained from the server’s encryption settings database, then the server and associated command-line tools should be able to retrieve the appropriate key from the encryption settings database so that the clear-text contents of the file can be accessed without any interaction (although if the cipher stream provider configured to protect the contents of the encryption settings database requires interaction, like the wait for passphrase cipher stream provider, then command-line tools may require interaction to unlock the encryption settings database).
- If the file is encrypted with a passphrase that the user specifies rather than one obtained from the encryption settings database, then the user will be interactively prompted for that passphrase when running the tool. This option cannot be used for keystore and truststore PIN files that need to be accessed by the server.

To encrypt a file with the server’s preferred encryption settings definition, use a command such as:

```
$ bin/encrypt-file --input-file password.txt \
  --output-file password.txt.encrypted
```

To use a key from an encryption settings definition that isn’t the default, use the `--encryption-settings-id` argument to specify the ID of the desired encryption settings definition (which can be obtained with `encryption-settings list`), such as:

```
$ bin/encrypt-file --input-file password.txt \
  --output-file password.txt.encrypted \
  --encryption-settings-id 4B6899D6716FC3AFFD71F7B447EB135063A0E724
```

To encrypt the file with a passphrase rather than a key from an encryption settings definition, use either the `--prompt-for-passphrase` argument to interactively prompt for the passphrase, or use the `--passphrase-file` argument to specify the path to a file containing the clear-text passphrase. For example:

```
$ bin/encrypt-file --input-file password.txt \
  --output-file password.txt.encrypted \
  --prompt-for-passphrase
```

Global configuration for backups and LDIF exports

Configuration for backups and LDIF exports can be done with the following global properties:

- automatically-compress-encrypted-ldif-exports. Indicates whether to automatically compress LDIF exports that are also encrypted. If set to true, any LDIF export that is encrypted (either explicitly with --encryptLDIF,or implicitly with the encrypt-ldif-exports-by-default configuration property) will automatically be gzip-compressed. If this is false, encrypted LDIF exports can still be manually compressed using the --compress command-line argument.
- backup-encryption-settings-definition-id. The unique identifier for the encryption settings definition to use to generate the encryption key for encrypted backups by default. If this property is given a value, then a definition with that ID must exist in the server's encryption settings database. If this property is not given a value, but the server is configured with at least one encryption settings definition, then the preferred definition is used. If no encryption settings definitions are available, the server will use an internal key shared among servers in the topology. Regardless of this property's value, it can be overridden with the backup command-line tool. Providing one of the --promptForEncryptionPassphrase or --encryptionPassphraseFile arguments will generate the encryption key from the provided passphrase. Or, the --encryptionSettingsDefinitionID argument can be used to generate the key from the specified encryption settings definition.
- encrypt-backups-by-default. Indicates whether the server should encrypt backups by default. If set to true, a defined backup-encryption-settings-definition-id value will be used to generate the encryption key for the backup. If this property is true, and if a backup-encryption-settings-definition-id value is not specified, the server will try to use the preferred encryption settings definition to generate the encryption key. If the server is not configured with any encryption settings definitions, an internal key that is shared among instances in the topology is used. Regardless of this property's value, it can be overridden with the backup command-line tool's --encrypt argument, even if this property is set to false. The --doNotEncrypt argument will always cause the backup to be unencrypted, even if this property has a value of true.
- encrypt-ldif-exports-by-default. Indicates whether the server should encrypt LDIFexports by default. If set to true, and an ldif-export-encryption-settings-definition-id value is specified, then that encryption settings definition is used to generate the encryption key for the export. If this property is true, and an ldif-export-encryption-settings-definition-id value is not specified, the server will first try to use the preferred encryption settings definition to generate the encryption key. If the server is not configured with any encryption settings definitions, then an internal key that is shared among instances in the topology is used. Regardless of this property's value, the default behavior can be overridden with the export-ldif command-line tool. The tool's --encryptLDIF argument will always encrypt the export, and the --doNotEncryptLDIF argument will always create an unencrypted exports.

Devise backup and restore strategies

Regularly-tested backup and restore strategies ensure that directory data is safe, correct, and usable. Beyond the basic data backup mechanisms that the server provides, make sure that the entire server installation is archived on a regular basis. This ensures that supporting content like encryption keys, certificate databases, and PIN files are properly backed up. Encrypted data cannot be used without them, and signed data cannot be trusted.

A security breach may result in altered data. It is important to have copies of data to identify what has changed. In the event that an attacker might have had access to a system for a long period of time, archived log data may be critical to understanding how the breach occurred and the extent of the damage.

Encrypt backups

Even if data encryption is enabled, someone accessing database files may be able to determine information about the database environment, such as indexes defined and unique values contained in the data. This applies not only to copies of the database on the server filesystem, but also for copies such as server backups.

The server provides the ability to encrypt the backup contents, including index and database structure information, which is not covered by data encryption. The server uses a different encryption mechanism for backups and LDIF exports than it does for data encryption, and automatically uses an encryption key shared across all servers in a replicated environment. Creating or restoring an encrypted backup requires that the server be online. For example, the following command performs an encrypted backup of all server backends:

```
$ bin/backup --task \
  --hostname directory.example.com \
  --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPasswordFile admin.password \
  --backupDirectory bak \
  --backUpAll \
  --encrypt
```

Note

If data encryption is enabled, make sure that the encryption settings definitions are backed up with the encrypted data. An attempt to restore a backend containing encrypted data without the necessary encryption settings definitions, will result in inaccessible data. If all backends (as in the example above) are included in the backup, the server automatically includes the encryption settings database. If any backends contain encrypted data, include the encryption settings backend to make sure that the necessary keys needed to access that data are available.

When a backup is performed, a `backup.info` file is created in the backup directory. This file provides information about the settings for that backup, including whether the backup is encrypted. The process for restoring a backup generated with or without the `--encrypt` argument is the same. For example:

```
$ bin/restore --task \
  --hostname directory.example.com \
  --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPasswordFile admin.password \
  --backupDirectory bak/userRoot
```

If restoring a backend that contains encrypted data, first restore any applicable encryption settings definitions that may be in use before restoring the data itself. The restore process for most backends completely eliminates any existing content. The resulting data set is only that which is contained in the backup. Restoring a backup of the encryption settings database preserves all definitions contained within, and adds any new definitions that are contained in the backup.

Secure LDIF exports

LDIF exports provide an additional backup mechanism that offers protection against latent corruption in the database, can iterate across every entry to discover corruption, and can also apply data encryption to an existing data set. LDIF exports may also be a more efficient way of generating indexes for a large number of attributes.

LDIF exports can contain sensitive data. The server provides the ability to encrypt LDIF exports similar to binary backups with global configuration options, or using the `--encryptLDIF` argument. To encrypt the data, perform the export with the server online. For example:

```
$ bin/export-ldif --task \
  --hostname directory.example.com \
  --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPasswordFile admin.password \
  --backendID userRoot \
  --ldifFile /ds/ldif/userRoot.ldif \
  --encryptLDIF
```

Although backups generate a descriptor file with information about the settings used for that backup, that is not available for LDIF files. When performing an import, it is necessary to explicitly indicate that the data is encrypted, using the `--isEncrypted` argument. For example:

```
$ bin/import-ldif --task \
  --hostname directory.example.com \
  --port 389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPasswordFile admin.password \
  --backendID userRoot \
  --ldifFile /ds/ldif/userRoot.ldif \
  --isEncrypted
```

Chapter 6: Protect the PingData Platform

Protecting the PingData Platform refers to protecting all of the server components that manage or store valuable user data. The PingDirectoryProxy Server can front the PingDirectory Server backends, providing efficient load-balancing or entry-balancing deployments. The PingDataSync Server can be deployed to provide synchronization capabilities between disparate system databases.

Topics include:

Separate user and administrator accounts

Accounts used to perform system or software administration should not be allowed to authenticate directly to the system. Only a set of users that have already authenticated to the system using limited individual accounts should be allowed to act as administrator.

This is best accomplished with roles, or through the use of the `sudo` command. Only users with a legitimate need should be allowed to access those accounts.

In some organizations, the individuals responsible for managing systems are different from those responsible for installing them. If there is overlap between these roles, it may be helpful for those individuals to have separate accounts for each task. This may help in tracking the actions performed by each account.

Installing or managing PingData software as a user other than the server user account, can cause files to be created with incorrect ownership, which will interfere with subsequent attempts to run the software using the server account. PingData server software can detect and prevent attempts to start the server or use certain administrative tools with an unexpected user account, but it may still be possible to cause some problems by attempting to manage the server with an unexpected user account. By maintaining a hard separation between the accounts for system and service administration, problems arising from mistakes like this are easier to avoid.

Use a limited account to run identity server services

PingData software should run under a user account that has a minimal set of capabilities. The account must be able to perform the following:

- Perform network communication.
- Read and write files at least below the server root, and potentially in other locations if components like log and database files are to be spread across multiple filesystems.
- Execute commands in at least the `bin` and `usr/bin` directories, as well as those in the `bin` directories below the server and JVM installations.

The account may need additional capabilities not normally granted to regular users, including the ability to listen on a privileged network port and the ability to use a greater number of file descriptors.

If running multiple instances of PingData server software on the same system, consider running each under a separate account. This provides a degree of isolation that can help minimize exposure if one of the accounts is compromised.

Considerations for root users

A directory root user is an all-powerful account that cannot be limited by access control or password policy restrictions, and in some cases is only allowed to authenticate through relatively insecure means. With PingData servers, there is very little difference between root users and regular users, with the exception that root user entries exist in the server configuration rather than in user data, and root users can be configured to automatically inherit certain privileges.

Many directories support only a single root user, which can cause several problems. First, it requires all administrators to share the same credentials, which makes it difficult to coordinate users to change those credentials. In addition, the need to share credentials among multiple individuals increases the risk that those credentials will be exposed, and also makes it difficult or impossible to audit the activities of individual administrators.

PingData servers can have any number of root users with their own credentials (including non-simple credentials, like a certificate for SASL EXTERNAL), individual privilege sets, and password policy restrictions. Since the special rights that root users have are granted through privileges and operational attributes, it is possible to create a non-root user that is just as powerful as a root user. The only real difference between a root user and a similarly defined non-root user is that the root user exists in the server configuration, and will be available even when other users may not be (if a backend containing user data is taken offline or temporarily unavailable). It is strongly recommended that root user accounts only be created for server administrators. If a non-administrator needs elevated privileges, create a normal user account with only those privileges needed to accomplish the desired tasks.

Root user accounts exist as user entries in the server configuration, below `cn=Root DNs,cn=config`. These entries should have a regular user structural object class, such as `inetOrgPerson`, and should also include the `ds-cfg-root-dn-user` auxiliary class. Other attributes to include in root user entries are:

- ds-cfg-alternate-bind-dn – Specifies an alternate DN that can be used to reference the root user when authenticating. For example, the default configuration has a single root user with a DN of cn=Directory Manager,cn=Root DNs,cn=config, but it is also possible to authenticate as that user with a DN of just cn=Directory Manager.
- ds-cfg-inherit-default-root-privileges – Indicates whether the root user automatically inherits the set of default root privileges as defined in the default-root-privilege-name property of the root DN configuration object. If this attribute is included in a root user's entry with a value of false, then that root user will only have an explicitly-designed set of privileges.
- ds-privilege-name – Explicitly configures individual privileges for the user. It can be used in conjunction with the ds-cfg-inherit-default-root-privileges attribute to add additional privileges on top of the default root privileges, or by prefixing the privilege name with a minus sign to indicate that privilege should not be granted to the user. For example, a value of -unindexed-search indicates that the root user should not have the unindexed-search privilege, even through it would normally be inherited as a default root privilege. It can also be used to specify the entire set of privileges for a user if ds-cfg-inherit-default-root-privileges is false.
- ds-pwp-password-policy-dn – Specifies which password policy should be applied to the root user. If no value is specified, the root user is subject to the server's default password policy. The server provides a special root password policy (in the "cn=Root Password Policy,cn=Password Policies,cn=config" configuration entry) that can be configured independently of the default policy. With the exception of ds-cfg-alternate-bind-dn and ds-cfg-inherit-default-root-privileges, all of these attributes can be included in the entries for any user in the server.

The ds-auth-allowed-address, ds-auth-allowed-authentication-types, ds-auth-require-secure-authentication, ds-auth-require-secure-connection, ds-auth-is-proxyable, and ds-auth-is-proxyable-by operational attributes can be used in both root user entries and normal entries. They can be assigned as either real or virtual attributes.

The ds-rlim-size-limit, ds-rlim-time-limit, ds-rlim-lookthrough-limit and ds-rlim-idle-time-limit operational attributes can be applied to prevent root user accounts from being used to perform denial of service attacks. See Denial-of-service prevention.

Centralized and remote logging

Directory syslog events should be written to a remote system. Logging to a remote server can be a vital aspect of security because it is much more difficult for an attacker to alter that content and compromise multiple systems. Also consider the use of WORM (write once, read many) drives or filesystems, which offer support for an append-only mode of operation in which data cannot be altered once it has been written.

PingData server products offer capabilities for logging to remote systems, such as:

- **UDP-based Protocol** – The PingDirectory Server only supports the UDP-based syslog protocol, which alone lacks communication security. Therefore, it is recommended that the server only communicate with a `syslog` daemon running on the local system over the loopback interface.
- **Loopback Communication** – To have the log messages delivered to a remote system, use loopback communication, to have the local syslog daemon simply act as an encrypted relay to a remote server. Open source and commercial syslog software (including `rsyslog` and `syslog-ng`) provide the ability to act as a syslog relay for the purpose of securely logging to a centralized server.
- **Custom Logging using the Server SDK** – The Server SDK can be used to create custom loggers to send messages to a centralized system using another mechanism, such as publishing them to a message queue.

Secure the configuration using privileges

The following are recommended steps to limit access and restrict changes to the system's configuration settings. This makes it more difficult for attackers to undo protections that are in place, or to grant themselves additional access:

- Access to the configuration requires the `config-read` privilege.
- Modifying the configuration requires the `config-write` privilege.
- Modifying the server schema requires the `update-schema` privilege.
- Modifying the server's access control configuration requires the `modify-acl` privilege.
- All configuration access is subject to access control evaluation. Users must have access control rights to perform the requested operations (or have the `bypass-acl` privilege).
- Access to the configuration can be restricted by client connection policy through the `include-dbackend-base-dn` and `excluded-backend-base-dn` properties. Criteria can also be defined to indicate which clients are allowed access to the configuration from specific IP addresses, only over secure connections, or only with specific authentication methods.
- Configuration changes made with the server online are recorded in the `config.audit` log. Configuration changes also generate administrative alerts. Configuration changes

made with the server offline are detected and an alert is generated when the server starts.

Safe use of dsconfig and the Administrative Console

The following points should be considered when making server configuration changes with the `dsconfig` tool and the Administrative Console:

- **Using dsconfig with No Arguments** – When launching `dsconfig` with no arguments, only LDAP simple authentication (optionally secured with SSL or StartTLS) is supported. SASL authentication is available when using command-line arguments (and if providing arguments needed for SASL in interactive mode).
- **Use SSH on Remote Systems** – When running `dsconfig` on a remote system, make sure that the communication is encrypted so that any credentials or other sensitive information provided to `dsconfig` are protected.
- **Use SSL or StartTLS** – Communication between `dsconfig` or the Administrative Console and the target servers, should use LDAP over SSL or StartTLS.
- **Use HTTPS** – When accessing the Administrative Console, use HTTPS rather than HTTP to make sure that any credentials or other sensitive information is encrypted.

Maintain consistent server configurations

For deployments with multiple servers, make sure that each server is configured identically. This prevents a configuration difference that may open one of the servers up to an attacker. The following global configuration property can be used to specify a server group:

configuration-server-group – Specifies a group (in the administrative data repository) of servers related to the current server. This is used by tools like `dsconfig` and the Administrative Console, so that changes can be applied to all servers in the group. This ensures that the configuration of all servers in the environment remains synchronized.

Data security audits

The PingDirectory Server provides an `audit-data-security` command-line tool that invokes a set of data security auditors to identify the audit type performed, current account, password, and privilege settings. Each data security auditor generates a time-stamped report. Reports from consecutive runs can be compared to determine if changes resolve one or more identified issues.

The audit-data-security tool enables specifying the backends to audit as well as the specific severity and verbosity levels for each event. The tool can execute on the Local DB, LDIF, and Configuration backends. Only one data security audit can run on any Local DB backend at any given time. If multiple backends are audited, the data security auditors scan these simultaneously. If scheduled for a specific time, the tool invokes an PingDirectory Server task that runs one or more auditors on the selected backends.

Only administrators with the audit-data-security privilege can run data security audits.

Data security audit reports

Each data security audit generates a detailed report file named after the type of auditor used. By default, the security audit report files are saved in LDIF format to the <server-root>/reports/audit-data-security/<timestamp> directory. In the top-level reports directory, a summary.ldif file provides an overview of the audit results. Sub directories for each backend store the detailed report files.

Open a report file using a text editor. The following command opens the entries-with-acis.ldif report for the userRoot backend.

```
$ cat /PingDirectory/reports/audit-data-security/20110811201049Z/userRoot/entries-with-
acis.ldif
dn: dc=example,dc=com
```

```
objectClass: ds-audit-report-aci-entry
objectClass: extensibleObject
ds-audit-severity: notice
ds-audit-reason: presence of access control information
aci: (targetattr!="userPassword")
  (version 3.0; acl "Allow anonymous read access for anyone";
    allow (read,search,compare) userdn="ldap:///anyone";)
aci: (targetattr="*")
  (version 3.0; acl "Allow users to update their own entries";
    allow (write) userdn="ldap:///self";)
aci: (targetattr="*")
   (version 3.0; acl "Grant full access for the admin user";
    allow (all) userdn="ldap:///uid=admin,dc=example,dc=com";)
```

Data security auditors

The following table lists the available data security auditors in the PingData. Each auditor is enabled by default, but can be disabled or modified using the dsconfig command.

Data Security Auditors

Auditor	Description
ACCESS-CONTROL	Reports all entries with access control information.
DISABLED-ACCOUNT	Reports all disabled accounts.
EXPIRED-PASSWORD	Reports all accounts with expired passwords, accounts with passwords about to expire, as well as accounts with passwords exceeding a specified age.
LOCKED-ACCOUNT	Reports locked accounts including the reason for locking.
MULTIPLE-PASSWORD	Reports entries with multiple password values. It is possible to configure the auditor to only report those entries that have multiple passwords using different password storage schemes.
PRIVILEGE	Reports entries with privileges. The report distinguishes between directly assigned privileges and privileges assigned by a virtual attribute.
WEAKLY-ENCODED-PASSWORDS	Reports all entries that use one of the specified weak password storage schemes.

Configure the data security auditors

Data security auditors can be configured using the `dsconfig` tool. Each auditor can be independently enabled or disabled. They may also be configured to include one or more attributes from the audited entry in the detailed report.

Each auditor can be configured to report events at a specific severity and verbosity level. The three possible severity values are: `Error`, `Warning`, and `Verbose`. If the `Warning` level is selected, then both `Error` and `Warning` events are included in the reports. Similarly, if the `Verbose` level is selected, the report will include events with any severity. By default, all auditors are configured with the `Warning` audit severity.

Perform the following steps to configure the data security auditors:

1. Run the `dsconfig` command and enter the connection parameters for the server.
2. Change to the Advanced menu.
3. On the main menu, enter the number corresponding to Data Security Auditor.
4. On the Data Security Auditor menu, choose to View and Edit an existing Data Auditor.
5. From the displayed list, choose a Data Auditor to modify.
6. For the specific Data Auditor, add or change any properties to be included in the audit.
7. Enter **f** to apply the changes.

The audit-data-security tool

The `audit-data-security` tool runs in either interactive mode or non-interactive mode. By default, the tool executes security audits on all enabled data security auditors on all supported backends (Local DB, LDIF, and Configuration).

Perform the following steps to run the `audit-data-security` tool:

1. Run the `audit-data-security` tool to perform a full audit of the PingDirectory Server.

   ```
   $ bin/audit-data-security
   ```

2. Open the `summary.ldif` to view the summary audit report, which will be in the time-stamped directory created by the audit.

   ```
   $ cat PingDirectory/reports/audit-data-security/20110811201049Z
   ```

3. To view a specific audit report, open the report in the backend sub directory.

To run a security audit on a subset of entries, run the `audit-data-security` tool on a subset of entries in the `userRoot` backend, but do not report on entries having privileges:

```
$ bin/audit-data-security --backendID userRoot \
  --excludeAuditor PRIVILEGE \
  --reportFilter "(employeeType=contactor)"
```

PingDirectoryProxy Server considerations

Most of the PingDirectory Server security features are also available in the PingDirectoryProxy Server, such as global configuration options, client connection policy restrictions, and connection handler options.

Security recommendations specifically for the PingDirectoryProxy Server are:

- **Use SSL or StartTLS** – Make sure that communication with backend servers is secured with SSL or StartTLS, so that third parties cannot access them.
- **Protect the Proxy User Account** – Make sure that the account used by the PingDirectoryProxy Server to authenticate to the backend PingDirectory Server instances is sufficiently protected. It should have strong credentials, and should not be used by any application other than the PingDirectoryProxy Server.
- **Prevent Direct Access to Backend Servers** – Consider preventing clients from directly accessing the backend PingDirectory Server instances and only allow access through the PingDirectoryProxy Server. It may be necessary to allow some degree of

direct administrative access to the PingDirectory Server instances, but all general client access should pass through the PingDirectoryProxy Server.

- **Use Proxy Transformations** – Consider installing proxy transformations, such as suppress attribute/suppress entry, to restrict what data is accessible through the PingDirectoryProxy Server.
- **Use a Generic User Account for Entry-Balancing Deployments** – If using entry balancing, consider configuring a generic user account with the same rights as various classes of users. If a user needs to perform an operation that requires processing in a backend set that does not contain the user's entry, use an authorization identity of the closest acceptable generic account.

PingDataSync Server considerations

The PingDataSync Server shares many of the same security features as the PingDirectory Server and PingDirectoryProxy Server. However, because the PingDataSync Server does not store any data, most of the security considerations involve securing data transmission from the Sync Source to the Sync Destination.

Security recommendations specifically for the PingDataSync Server are:

- **Use SSL or StartTLS with Endpoint Servers** – Make sure that communication with the endpoint servers is secured.
- **Use SSL or StartTLS with Client Communication** – Make sure that client communication with the PingDataSync Server is secured.
- **Make the Encryption Key Sufficiently Complex** – When using the Changelog Password Encryption Plugin in the PingDirectory Server to synchronize passwords to a non-PingData endpoint, make sure that the encryption key is complex and is handled securely. The actual decryption key is derived from the user-configured key using a proprietary method, so it is unlikely that hackers could decrypt passwords stored in the changelog, even with access to the key.
- **Consider Access Control Filters for Notification Mode** – For deployments using notification mode, administrators can configure a Sync Pipe that performs access control filtering on the changelog data as it comes back from the source PingDirectory Server. The access controls filter out attributes that the user does not have the privileges to see before they are returned. This is configurable using the `filter-changes-by-user` property on the Sync Pipe configuration.
- **Consider Obfuscating Attributes** – To secure sensitive user data, the PingDataSync Server is capable of fully synchronizing test or stage servers with production servers

while also obfuscating sensitive customer information, such as social security numbers and passwords. This is configurable using the `scramble-value` property on the Direct Attribute Mapping configuration.

- **Set the Appropriate Log Detail Levels** – The Sync Log Publisher provides information about synchronization operations that are processed. The level of detail can be specified by setting the `logged-message-type` property. There are three values that are useful for debugging, but can potentially expose sensitive information in the sync log: `change-detected-detailed`, `entry-mapping-details`, `change-applied-detailed`. The PingDataSync Server is not aware which attributes are sensitive while they are in transit, so using one of these detailed log levels may be a security risk. However, passwords always appear in the Sync log in hashed form, regardless of the log detail level.

Chapter 7: Data Integrity

The server can be configured to require secure communication with all clients, but that does not provide protection for individuals that can access the server filesystem and may be able to interact with the database files. The database files should be protected, as well as alternate representations of that data including the live database files, database backups, and LDIF exports of the data).

Topics include:

Stored Entry Checksums

The PingDirectory Server provides two checksum features that can be used to help ensure data integrity: cryptographic digests and entry checksum operational attributes.

Cryptographic Digests

Cryptographic digests can be included in the encoded representation of an entry stored in the database. This can help detect database corruption, for example if a bit gets flipped between the time the server tries to store an entry in the database and the time the data is actually written to disk. The server provides an option to write these checksums when storing entries. Another available option enables the server to look for and validate that the entry content still matches the digest from which it was retrieved. If validation fails, the server generates an administrative alert.

The following two options can be used to control cryptographic digests in entry contents:

- **hash-entries** – In JE backends, the hash-entries property is used to indicate whether the server should include cryptographic digests in entries when they are written. If set to `true`, any entry created or updated after that time will be stored with an MD5 digest of the contents of that entry.
- **verify-entry-digests** – In the global configuration, indicates whether the server should automatically verify any cryptographic digests that exist in the encoded representation of entries when decoding them. Whether entry digests should be generated is controlled by the hash-entries configuration property in backends that support this capability (including JE backends). The process of generating these digests can be controlled independently of their verification, so that verification can be enabled only if there may be a database corruption.

Because there are separate properties that control generating digests and verifying them, entry digests can be generated when entries are written to the database, reducing the added cost of generating the digest. Periodically enable digest validation before performing an operation that requires retrieving each entry from the database (the `export-ldif` , `verify-index`, or `audit-data-security` tools). This identifies any entries whose encoded representation does not match the stored digest.

If data encryption is enabled, the encryption performed when storing entries also serves as a method for verifying the integrity of encoded entries. If an encrypted entry becomes corrupted, the server cannot decrypt it, and an administrative alert is generated. If data encryption is enabled, storing cryptographic digests is redundant.

Entry Checksum Operational Attribute

The PingDirectory Server provides the ability to include a `ds-entry-checksum` operational attribute in entries returned to clients, whose value will be a checksum of the attributes contained in that entry.

This is useful in conjunction with the LDAP assertion control to ensure that an entry has not been altered since it was last retrieved. The process to use both would be to retrieve an entry, including the value of the `ds-entry-checksum` attribute, and then issue a modify request that includes an LDAP assertion control with a filter that ensures that the entry's current `ds-entry-checksum` value matches the value that was retrieved. If the entry had been altered by another client between the time the entry was retrieved and the time the modify request was sent, the assertion filter will not match and the modify operation is not performed.

The `ds-entry-checksum` attribute includes a checksum of all attributes in the entry except for those listed in the `excluded-attribute` property of the virtual attribute configuration. However, a similar process without the `ds-entry-checksum` attribute, would be creating an assertion filter with the values of a specified set of attributes from that entry. This ensures that those attributes have not been altered since the entry was last retrieved, but enables updates to other attributes without causing the assertion to fail. If only concerned about a specified set of attributes rather than the entire entry, this approach may be used to reduce the likelihood of an operation failure due to a conflicting update. It can also be used in environments containing non-PingData servers.

Schema Integrity

The PingDirectory Server supports schema validation, including features such as DIT content rules, DIT structure rules, name forms, and matching rule uses that many servers do not support. It also ensures that attribute values conform to the constraints of the associated attribute syntax.

Some servers perform better for operations (like LDIF import) with schema checking disabled, and administrators opt for performance over integrity. This is not a trade-off that needs to be made in the PingDirectory Server, because it actually performs better with schema validation enabled.

There are several security implications when schema validation is disabled. For example, a client can inadvertently store data in an incorrect attribute. If an ACI or sensitive attribute definition is configured to deny access, information intended for the specified attribute, but stored with the wrong name, may not be protected. If that attribute is used for some operational configuration (for defining access control rules or privileges), the server would not apply the intended restrictions. Similarly, clients attempting to find that information would not be able to find it and may behave incorrectly.

Even if the client uses the correct attribute name, supplying a value that violates the associated syntax may cause unintended behavior. For example, if schema checking or syntax validation is disabled during an LDIF import, the server will not detect or reject malformed access control rules. The server can detect malformed access control rules at start up or after an online import, but it is better to have these problems detected during import rather than at startup.

The Global Configuration Properties contain these two controls for enforcing schema validation:

- `check-schema` – Specifies whether the server should enforce compliance with the defined schema. Schema checking is highly recommended.
- `invalid-attribute-syntax-behavior` – Specifies the behavior that the server should exhibit when encountering attribute values that violate the associated attribute syntax. The value should be one of the following:
    - `reject` – Specifies that the server reject any values that violate the syntax.
    - `accept` – Specifies that the server silently accept such values.
    - `warn` – Specifies that the server accept malformed values, but log a warning message when these values are encountered.

Note

It is strongly recommended that invalid values be rejected.

Limiting Exposure of Stale Data

Another security concern is the possibility of serving stale data to clients. If one or more servers have fallen behind in replication, changes to data that impact the security of the environment may not propagate as quickly, which can leave a window of vulnerability.

The PingDirectory Server detects if there are missed changes and enters lock-down mode if the missed changes are no longer stored on any other PingDirectory Server. This typically occurs after the server has been offline or isolated on the network for a period of time longer than the `replication-purge-delay`. While in lockdown mode, only Root DN access is allowed.

For example, if an employee leaves the company, his or her account should be removed or disabled as quickly as possible to remove the ability to authenticate to applications that use the directory. If one PingDirectory Server instance has fallen behind in replication, that former employee account still has access until the update to the account is propagated. The same could be the case for changes that grant or revoke privileges, change group membership, or otherwise impact the level of access that a given individual might have.

The PingDirectory Server offers the following global configuration properties to avoid exposing stale data:

- `startup-min-replication-backlog-count` – Specifies the minimum number of outstanding replication changes that causes server startup to be delayed until replication can complete. A server that has been offline for a period of time may have stale data. Delaying startup until replication has finished prevents stale data from being served to clients.
- `replication-backlog-count-alert-threshold` – Specifies the minimum number of outstanding replication changes that will cause the server send an alert that it has a significant replication backlog.
- `replication-backlog-duration-alert-threshold` – Specifies the minimum age of any outstanding replication changes that will cause the server to generate an alert that it has a significant replication backlog and may be serving stale data to clients.

The previous properties are applicable only to the PingDirectory Server. Stale data can also be an issue for the PingDirectoryProxy Server and thePingDataSync Server. Each have features for addressing this problem:

- In the Proxy, the replication backlog LDAP health check can be used to monitor the replication state of each PingDirectory Server, and can de-prioritize or stop using a given server if it falls too far behind, based on either the absolute number of outstanding changes, or the age of those changes.

- In the PingDataSync Server, the `sync-backlog-alert-threshold` property for Sync Source objects (for PingData and Sun/Oracle PingDirectory Server instances) can be used to generate an alert if the PingDataSync Server detects a significant number of unprocessed changes.

In addition, the systems expose replication metrics in two ways:

- **Replication MIB** – Within the SNMP monitoring feature, each server exposes a standards compliant Replication MIB containing metrics related to the current state of replication, which can help diagnose how much outstanding work replication may have to do.
- **Replication Metrics available in the PingDataMetrics Server** – Within the PingDataMetrics Server, there are 25-30 metrics that deal specifically with replication, such as send/receive windows, current backlog, and conflict counts.

Time Synchronization

All systems that participate in an PingData server environment should have time synchronization enabled, preferably with Network Time Protocol (NTP). Servers should be synchronized to an atomic clock so that they are accurate as well. Time synchronization is important for replication conflict resolution, proper handling of password expiration, proper handling of account expiration and account lockout, proper handling of certificate expiration, and proper handling of GSSAPI authentication. It is also important to have accurate timestamps in log messages to correlate events across multiple systems, and potentially with logs maintained by clients.

If the clocks of one or more systems are out of synchronization by a relatively small amount (within hours of each other), it is recommended that NTP be used to synchronize them. NTP synchronization works by adjusting the rate at which the system clock advances gradually to synchronize it, rather than using a massive jump that can disrupt replication and other server components like JVM pause. This is important for cases in which a system clock is too fast. Time changes that cause the clock to move backward could result in unexpected behavior from replication conflict resolution.

If the system clocks are significantly out of synchronization (more than a couple of days), work with an authorized support provider to determine the best course of action for correcting the problem without introducing risk of replication problems, accounts being unexpectedly locked, passwords prematurely expired, or other issues pertaining to state information.

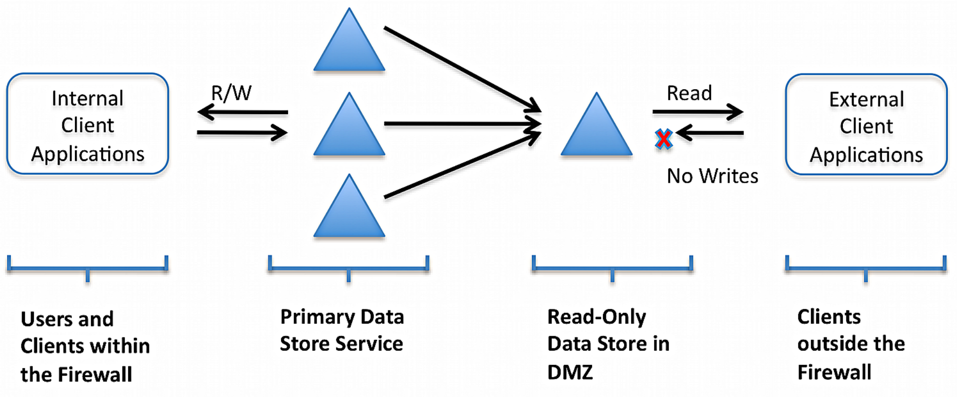
Creating a Read-Only Instance of the PingDirectory Server

The PingData product family provides configuration properties that disable all write access to the data. This can be useful when exposing the directory service publicly, such as an authentication service, while blocking malicious attempts to alter the data.

The `writability-mode` property indicates whether the server will allow write operations. The value may be one of the following:

- `enabled` – Write operations are allowed for properly authorized clients.
- `disabled` – No write operations are allowed.
- `internal-only` – Allows writes invoked by internal operations or received from replication, but rejects any write request received from an external client.

Some environments may want to make a server (one located in a DMZ) read-only, but still allow replicated operations from internal servers, as in the following illustration:


Internal Client Applications
R/W
Read
No Writes
External Client Applications
Users and Clients within the Firewall
Primary Data Store Service
Read-Only Data Store in DMZ
Clients outside the Firewall

Note

It is also possible to configure the writability mode for individual backends.

Server Lock-Down Mode

A PingData server will place itself into lockdown mode to protect data in the following circumstances:

- **Lockdown for ACI Integrity** – The PingData product family examines and validates all ACIs stored in the data whenever a backend is brought online. If any malformed ACIs are found in the backend, the server generates an alert to notify administrators of the problem and places itself in lockdown mode. While in lockdown mode, the server only allows requests from users who have the `lockdown-mode` privilege. This enables administrators to correct the malformed ACI while ensuring that no sensitive data is inadvertently exposed. When the problem has been corrected, the administrator can use the `leave-lockdown-mode` tool or restart the server to enable it to resume normal operation.
- **Lockdown for Data Integrity** – The PingDirectory Server detects if there are missed changes and enters lock-down mode if the missed changes are no longer stored on any other PingDirectory Server. This typically occurs after the server has been offline or isolated on the network for a period of time longer than the configured `replication-purge-delay`. While in lockdown mode, only Root DN access is allowed..

Storing Reversible Changes in the Log

This configuration option is available for the Changelog backend:

`use-reversible-form` – Indicates whether changelog entries for modify operations should record information about the change in a way that will allow it to be reverted, restoring the entry to the way it appeared before the change was applied. If reversible form is enabled, then delete changelog records will automatically include all deleted entry attributes.

Chapter 8: Client connection and password policies

Client connection policies provide a way to segregate client connections based on similar characteristics, and configure the ways they can interact with the server.

The password policy system can assign, manage, or remove password policies for root and non-root users. The password policy contains configurable properties for password expiration, failed login attempts, account lockout and other aspects of password and account maintenance on the PingDirectory Server.

Topics include:

Password policies and the PingDirectoryProxy Server

Associating a Client Connection Policy with a client connection

When a client establishes a connection to the server, the server assigns a Client Connection Policy for that connection. If the client performs a bind (which can change the identity of that connection) or uses the StartTLS extended operation (which can change an insecure connection to a secure one), the server will re-evaluate the connection and may assign it a different policy.

The policy properties that the server uses to select a Client Connection Policy for a client connection are:

- **enabled** – If a policy is enabled, it is eligible to be selected.
- **evaluation-order-index** – The evaluation order index controls the order in which policies are examined to determine whether they are appropriate for a connection. Each Client Connection Policy must have a unique evaluation order index value. Policies are evaluated in ascending order based on this index. The evaluation order index values do not need to be in sequential order.
- **connection-criteria** – If a policy is associated with connection criteria, then a connection must match that criteria for it to be associated with the Client Connection Policy. If a policy does not have any connection criteria, it will match any connection.

When evaluating Client Connection Policies, the server selects the enabled policy with the lowest `evaluation-order-index` that either has or does not have criteria that matches that connection. If none of the enabled policies match the client connection (either at the time the connection is established, or after performing a bind or StartTLS operation), that connection is terminated. Similarly, if the Client Connection Policy that is selected has a `terminate-connection` value of `true`, the connection is terminated.

Recommendations for creating Client Connection Policies

If using Client Connection Policies to enforce restrictions for different classes of clients, consider the following:

- **Client Connection Policies for Unauthenticated Clients** – Make sure that a policy exists that will allow unauthenticated clients. When a new connection is established to the server, it will be unauthenticated, and remains that way until the client has successfully completed a bind operation. A Client Connection Policy is selected for the connection when it is established, so at least one policy must allow unauthenticated clients by not having a `connection-criteria` value, or having a `connection-criteria` value that references a criteria with a `user-auth-type` that includes `none`.
- **Client Connection Policies for Authenticated Connections** – To have a policy that only applies to authenticated connections, the policy must have a `connection-criteria` object, and the referenced criteria must have a `user-auth-type` value that does not include `none`.
- **Client Connection Policies for StartTLS** – To allow the use of the StartTLS extended operation, have a client connection policy that allows insecure connections. StartTLS converts an existing insecure operation into a secure connection, so it is necessary to have a client connection policy that allows the initially insecure connection and allows it to issue the StartTLS extended request.
- **Multiple Client Connection Policies** – If configuring multiple client connection policies, it is possible for a connection to match the criteria for more than one policy. Make sure that the evaluation order indexes of those policies are configured so that the most appropriate policy for a given connection will have a lower evaluation order index than any other policy that could be selected for that connection.

Password Policies

Password Policies are used to make sure that a password provided during authentication is correct, that password changes use strong formats, and that a user can't continue using the same password for too long. Password Policies also provide features that aren't strictly password-related, including locking accounts if there are too many failed authentication attempts, keeping track of the last time that a client authenticated, and enforcing constraints around the kinds of authentication that are allowed. The server also provides support for account status notification handlers, which can be used to notify end users or administrators when significant password policy events occur.

There may be some users that have different password policy requirements than others. For example:

- Administrative accounts provide a greater level of access than normal user accounts and may warrant additional forms of protection that are not considered necessary for normal users.

- Account lockout can be enabled after a certain number of failed attempts for most users. However, administrative accounts should not be locked out because that might enable an attacker to orchestrate a denial-of-service attack.
- A small set of users may need to use a particular application, which requires the use of a weaker password storage scheme than required for most accounts.

The PingDirectory Server provides the ability to define multiple password storage schemes, which can be configured to reflect the needs of different groups of users. One of those password policies is configured as the default policy for the server through the `default-password-policy` property in the global configuration. It is applied to any user for which no alternate policy is configured. To apply an alternate password policy for a user, add the `ds-pwp-password-policy-dn` attribute to that user's entry with a value equal to the password policy that should be enforced for that user. If a user's entry contains a `ds-pwp-password-policy-dn` attribute that references a password policy that does not exist, that user will not be allowed to authenticate to the server.

An alternate password policy can also be assigned by creating a virtual attribute that generates a `ds-pwp-password-policy-dn` value in entries for users that match certain criteria. The `user-defined` virtual attribute type is ideal for this. For example, the following command can be used to assign the password policy defined in configuration entry `cn=Secure Password Policy,cn=Password Policies,cn=config` to any user that is a member of the `cn=Secure Users,ou=Groups,dc=example,dc=com` group:

```
$ bin/dsconfig create-virtual-attribute \
  --name "Assign Secure Password Policy" \
  --type user-defined \
  --set enabled:true \
  --set attribute-type:ds-pwp-password-policy-dn \
  --set "value:cn=Secure Password Policy,cn=Password Policies,cn=config" \
  --set "group-dn:cn=Secure Users,ou=Groups,dc=example,dc=com" \
  --set "conflict-behavior:real-overrides-virtual"
```

The `real-overrides-virtual` conflict behavior means that if any member of that group already has an explicitly-assigned alternate password policy, that assignment takes precedence over the virtual attribute. If multiple virtual attributes attempt to assign different values for the same attribute in the same entry, only one of those values is selected. The process for selecting which virtual attribute is used is undefined. If virtual attributes are used to assign a password policy, do not configure virtual attributes that may overlap.

Password validators

Password storage schemes can encode passwords to protect the clear-text password used to generate that encoding using an algorithmic approach. To protect against attacks that try to guess passwords, make sure that users have passwords that are complex, and use password validators to ensure that new passwords are strong.

When a new password is assigned to a user through an add operation, modify operation, or password modify extended operation, that password must be considered acceptable by all password validators configured in the password policy for that user. If any password validator considers the password to be unacceptable, then the attempt to assign that password is rejected.

The `password-validator` configuration property controls the set of password validators that should be used for a given policy. Create a password validator configuration entry in the server and update one or more password policies to make use of that validator.

The PingDirectory Server supports several password validators, including:

- **Attribute value password validator** – Ensures that the proposed password does not match the value of any attribute in the user's entry, such as the user's name, telephone number, or address. It can be configured to match all attributes in the user's entry or a subset. It can check both forward and reversed versions of the password. It can also check for cases in which the password is a substring of an attribute value or an attribute value is a substring of the password.
- **Character set password validator** – Ensures that the proposed password includes characters from a number of character sets such as one lowercase letter, one uppercase letter, one numeric digit, and one symbol. The character sets can be defined, including the minimum number of characters from each set, and whether passwords can include characters that are not in any of the defined sets.
- **Commonly-Used passwords dictionary validator** – Ensures that the proposed password is not one of 10,000 commonly used passwords. These are words that are common for attackers to use when trying to access user accounts. The Commonly-Used Passwords validator is invoked by the Secure Password Policy by default. The word list is located in `<server-root>/config/commonly-used-passwords.txt`, and can be used to create a custom validator, but should not be modified.
- **Dictionary password validator** – Ensures that the proposed password is not contained in a specified dictionary file, to prevent common words from being used as passwords. The server comes with a dictionary file, but an alternate dictionary can be

selected. The validator can also perform case-sensitive validation and look to see if the reversed password is present in the dictionary.

- **Haystack Password validator** – Ensures that the proposed password is secure based on a combination of its length and the types of characters that it contains. For example, a longer password containing only lowercase letters may be stronger than a shorter password containing a mix of uppercase and lowercase letters, numbers, and symbols. This is based on the Gibson Research Corporation Password Haystacks concept.
- **Length-based password validator** – Ensures that the proposed password meets certain length constraints.
- **Regular expression password validator** – Ensures that the proposed password either matches or does not match a given regular expression.
- **Repeated characters password validator** – Ensures that the proposed password does not contain any character repeated more than a specified number of times in a row. The maximum number of times a character can appear consecutively and case-sensitive validation can be configured.
- **Similarity-based password validator** – Ensures that the proposed password is not too similar to the user's current password. It uses the Levenshtein Distance algorithm to compute the number of changes (where a change may include inserting a character, removing a character, or replacing a character). To use this validator, make sure that the `password-change-requires-current-password` option is enabled in the password policy, which requires users to supply the current password when setting a new one.
- **Unique characters password validator** – Ensures that the proposed password contains at least a specified number of different characters. The minimum number of unique characters and whether to use case-sensitive validation can be defined.
- **Custom Password Validator** – The Server SDK can also be used to create custom password validators in Java classes or Groovy scripts. Any number of password validators can be configured. For example, one dictionary validator can be enabled with the default `wordlist.txt` dictionary, and another with an additional set of words that should be forbidden.

By default, password validation is applied for users changing their own passwords and for administrators resetting the passwords for other users. If administrators are able to set temporary passwords that are weak or easily guessable, that may allow an attacker to request a password reset for another user, creating an avenue to gain access to that user's account. However, if for some reason a user's password must be reset without password validator restrictions, the `skip-validation-for-administrators` property can be set.

Password expiration

Given enough time, a dedicated attacker may be able to guess a user's password. If someone has access to the encoded representation of a password, with enough time and computing power they will be able to break the password by trying every possible combination of characters. There are a number of factors that may impact the time required to accomplish this, including the length of the password, the set of characters that may be included in it, and the storage scheme used to encode it.

If using passwords to authenticate, the best ways to mitigate risk is to increase the cost of a brute force attack by choosing an expensive password storage scheme (like the 512-bit SHA-2 variant of the crypt algorithm and increase the number of digest rounds), and/or to reduce the length of time that a password can remain valid. The latter can be enforced with password expiration, which may require that users change their passwords on a regular basis.

Several configuration properties can be used to configure password expiration, including:

- `max-password-age` – Specifies the maximum length of time that a user can continue to use a password before it expires. Password expiration is not enabled in the default password policy.
- `password-expiration-warning-interval` – Specifies the length of time before a user receives warnings about a password expiration. Warnings are delivered with LDAP response controls, but account status notification handlers can also be used to deliver warnings in other forms, such as e-mail.
- `expire-passwords-without-warning` – Indicates whether a user's password should be allowed to expire even if that user has not received a warning about an upcoming expiration. For example, if there is a relatively short warning interval (such as 5 days) and a user has been on vacation and has not received the warning. If `expire-passwords-without-warning` is set to `false` (which is the default), the server sends at least one warning before the password is considered expired.
- `grace-login-count` – Indicates that a user is given a number of grace logins. If a user's password has expired, a grace login can enable a user to authenticate, but that user cannot do anything until the password is changed. By default, no grace logins are granted.
- `allow-expired-password-changes` – Indicates whether a user should be allowed to change his or her password after it has expired, using the password modify extended operation. In the default password policy, this is not allowed. The user can use the account again is to have an administrator reset the password.

If password expiration is enabled, make sure that LDAP clients can look for and consume the bind response controls that indicate that a password is about to expire, so that clients can display those warnings to users. A regular auditing process can also be used to periodically identify and notify clients whose passwords are about to expire.

Password changes and administrative reset

The PingDirectory Server differentiates between user password changes and administrative password reset based on whether the user issuing the request is the same as the user whose password is being changed. If a user is changing his or her own password, that is considered a self password change. If a user changes someone else's password, that is considered an administrative password reset. For either password change, the requester must have the necessary access control permissions to make the change. For administrative password reset, the requester must also have the `password-reset` privilege.

The server may be configured to enforce a different set of restrictions for administrative password reset operations than it does for self password changes. For self password changes, the following configuration properties may be in effect:

- `allow-user-password-changes` – Specifies whether users can change their own passwords. Even if this is set to `true`, the user must have permission by the access control subsystem.
- `password-change-requires-current-password` – Specifies whether users are required to supply their current password when choosing a new password. If this is set to true, the current password may be provided in a modify operation by deleting the old password value and adding the new password. The password modify operation has a dedicated field for providing the current password.
- `min-password-age` – Specifies the minimum length of time before a user is allowed to change his or her password.

Configuration properties that apply to administrative password reset operations include:

- `force-change-on-add` – Specifies whether users are required to change their password the first time they authenticate. If so, no actions can be performed by the user until the password is changed.
- `force-change-on-reset` – Specifies whether users are required to change their password after it has been reset by an administrator. If so, no actions can be performed until the password is changed.

- `max-password-reset-age` – Specifies the maximum length of time that a user has to change his or her password after their account has been created (if `force-change-on-add` is `true`) or their password has been reset (if `force-change-on-reset` is `true`). If specified, this can limit the length of time that the administrator-supplied password can be used to authenticate, which limits the time that an attacker could use that password to gain control of the user's account.

An additional set of configuration properties apply to both user password changes and administrative reset, including:

- `allow-pre-encoded-passwords` – Specifies whether clients are allowed to provide passwords in a pre-encoded form in add operations, modify operations, or password modify extended operations. By default, pre-encoded passwords are not allowed because they cannot be interpreted by the server to invoke password validators, check them against password history, and perform other necessary checks. Pre-encoded passwords should only be allowed for clients that can only provide new passwords in that format. Clients will never be allowed to use pre-encoded passwords for authentication.
- `password-history-count` – Specifies the maximum number of previous passwords to retain in the password history.
- `password-history-duration` – Specifies the maximum length of time to retain a history of previous passwords. This can be useful if password history should be based on a period of time rather than a fixed number of passwords.

Note

If password history is enabled (by setting a `password-history-count`, a `password-history-duration`, or both), users cannot choose a new password that is the same as any password in the history.

Account lockout, expiration, and disablement

To prevent an attacker from sending repeated LDAP bind requests in an attempt to guess user passwords, the server provides the ability to lock accounts after too many failed attempts. The lockout can persist until an administrator resets the user's password, or the account can be automatically unlocked after a period of time without any administrative action required.

The configuration properties related to account lockout include:

- `lockout-failure-count` – Specifies the maximum number of failed authentication attempts allowed before an account is locked. A value of zero disables account lockout.

- `lockout-duration` – Specifies the length of time that an account should remain locked before it is automatically unlocked. A duration of zero seconds indicates that locked accounts are not unlocked until an administrator resets the user's password.
- `lockout-failure-expiration-interval` – Specifies the length of time that information about a failed authentication attempt is retained by the server. The record of previous authentication failures for a user is automatically cleared when that user successfully authenticates, but information about failed authentication attempts can be configured to expire after a period of time even without a successful authentication.
- `ignore-duplicate-password-failures` – Specifies whether the server should consider repeated authentication failures with the same incorrect password as a single failure. Repeatedly trying the same wrong password is obviously not an attack designed to guess a user's password, and is more likely the case that the user's password has recently changed and a client is still trying to use the old password. This feature can prevent the user's account from being inadvertently locked if the failure looks like an honest mistake.

In addition to locking an account, the PingDirectory Server provides other features that can be used to disable accounts. These are not configured in the password policy, but are used by setting operational attributes in the user's entry.

Those attributes include:

- `ds-pwp-account-disabled` – When present in a user's entry with a value of `true`, that user account is disabled and cannot be used to authenticate. The user's account can be re-enabled by either changing the value to `false` or removing the entire attribute.
- `ds-pwp-account-expiration-time` – When present in a user's entry, the value must be a timestamp (in generalized time format). The account is not allowed to authenticate after that time. This can be useful when creating temporary accounts. Once an account expires, it can be made usable again by changing or removing the account expiration time.

Alternative to account lockout

As an alternative to account lockout, a `failed-bind-response-delay` configuration property can be set on the LDAP Connection handler to instruct the server to introduce a delay (such as one second) into the process of returning a response to an unsuccessful bind operation. Delaying the response to a failed bind only affects the connection on which the bind was attempted, and still limits the rate at which a malicious client can try to guess a user's password. However, it will not affect other attempts to authenticate as that user on other connections, so the legitimate user can still authenticate with the correct password.

Last login time and last login IP address tracking

The PingDirectory Server can track information about a user's authentication behavior in that user's entry, including the time that the user last authenticated and the IP address of the client system. These features can be configured using the following properties:

- `last-login-time-attribute` – Specifies the name of the attribute in which the time of the user's last login may be recorded. The server schema includes the `ds-pwp-last-logintime` operational attribute which can be used for this purpose.
- `last-login-time-format` – Specifies the format in which the last login time value should be recorded. This should be given in the format supported by the `java.text.SimpleDateFormat` class, like `yyyyMMddHHmmss.SSS'Z'` (which will report values in the generalized time format, including millisecond accuracy). An alternate format with less accuracy can be used, because the server will only update the value in the user's entry if it is different from the existing value. For example, a value of `yyyyMMdd` has only day-level accuracy, which means that the value will be updated once per day.
- `previous-last-login-time-format` – Used to hold previous values used by the `last-logintime-format`. This allows the server to parse old last login time values for the purposes of evaluating them for the idle lockout interval.
- `idle-lockout-interval` – Specifies the maximum length of time allowed to pass without a user authentication (as determined by the `last-login-time` attribute) or password change before that account is locked due to inactivity. If an account is locked because the user has not authenticated in a period of time greater than the idle lockout interval, it can be unlocked by an administrative password reset.
- `last-login-ip-address-attribute` – Specifies the name of the attribute in which the server will record the IP address of the system from which the client last authenticated. The server schema includes the `ds-pwp-last-login-ip-address` attribute, which may be used for this purpose.

Password generators

When using the password modify extended operation, the client can either supply a new password or have the server generate a new password for the user included in the extended response. If the client does not provide the new password, the server uses a password generator to create one.

The password generators available for use in the server include:

- **Random password generator** – Can be used to construct passwords from characters selected at random from one or more character sets. The character sets and patterns can be defined, specifying which character sets to use and the number of characters from each. For example, the default instance of the random password generator is configured to generate eight-character passwords comprised of three alphabetic characters, two numeric digits, and three more alphabetic characters.
- **Custom password generators** – The Server SDK also provides the ability to define custom password generators, using either Java classes or Groovy scripts.

Account status notification handlers

The PingDirectory Server includes the following account status notification handlers:

- **Error log account status notification handler** – Cause messages to appear in the server error log for selected account status notification events.
- **SMTP account status notification handler** – Cause e-mail messages to be sent to end users (and optionally administrators) for selected account status notification events.

Account status notification handlers can be used to make information available to administrators and/or end users about significant events related to password policy processing. These events include:

- `account-temporarily-locked` – Indicates that a user's account has been locked due to too many failed authentication attempts, but will automatically be unlocked after a period of time.
- `account-permanently-locked` – Indicates that a user's account has been locked due to too many failed authentication attempts, and will require a password reset to be unlocked.
- `account-unlocked` – Indicates that a user's account has been unlocked by an administrator.
- `account-idle-locked` – Indicates that a user authentication attempt failed because the account remained idle for too long.
- `account-reset-locked` – Indicates that a user authentication attempt failed because the user failed to change the password in a timely manner, after an administrative password reset.

- `account-disabled` – Indicates that an administrator disabled a user account.
- `account-enabled` – Indicates that an administrator enabled an account that was disabled.
- `account-expired` – Indicates that a user authentication attempt failed because the user's account expired.
- `password-expired` – Indicates that a user authentication attempt failed because the user's password expired.
- `password-expiring` – Indicates that a user's password is about to expire. This notification will only be used the first time that a warning is sent before an upcoming expiration. Subsequent authentication attempts during the warning interval will not result in this notification.
- `password-reset` – Indicates that an administrator reset the password for another user.
- `password-changed` – Indicates that a user changed his or her own password.

Per-user Password Policies

To apply an alternate password policy for a particular user, add the `ds-pwp-password-policy-dn` attribute to that user's entry with a value equal to the password policy that should be enforced.

Note

> If a user's entry contains a `ds-pwp-password-policy-dn` attribute that references a password policy that does not exist, that user will not be allowed to authenticate to the server.

An alternate password policy can be assigned by creating a virtual attribute that generates a `ds-pwp-password-policy-dn` value in entries for users that match certain criteria. The user-defined virtual attribute type is ideal for this. For example, the following command can be used to assign the password policy defined in configuration entry `cn=Secure Password Policy,cn=Password Policies,cn=config` to any user that is a member of the `cn=Secure Users,ou=Groups,dc=example,dc=com` group:

```
$ bin/dsconfig create-virtual-attribute \
  --name "Assign Secure Password Policy" \
  --type user-defined \
  --set enabled:true \
  --set attribute-type:ds-pwp-password-policy-dn \
  --set "value:cn=Secure Password Policy,cn=Password Policies,cn=config" \
  --set "group-dn:cn=Secure Users,ou=Groups,dc=example,dc=com" \
  --set "conflict-behavior:real-overrides-virtual"
```

The `real-overrides-virtual` conflict behavior means that if any member of that group already has an explicitly-assigned alternate password policy, that assignment will take precedence over the virtual attribute. If multiple virtual attributes attempt to assign different values for the same attribute in the same entry, only one value is selected, and the process for selecting which virtual attribute is currently undefined. If virtual attributes are used to assign a password policy, make sure that virtual attributes are not configured to overlap for users.

Additional password policy properties

Password policies also provide support for a number of additional configuration properties that do not fit into the previous categories. They include:

- `password-attribute` – Specifies the name or object ID of the attribute that should hold password values in user entries. The attribute type must be defined in the schema and must have a syntax of either `1.3.6.1.4.1.30221.1.3.1` (for values in the `userPassword` syntax) or `1.3.6.1.4.1.4203.1.1.2` (for values in the `authPassword` syntax).
- `require-secure-authentication` – Specifies whether users associated with this policy are required to authenticate in a secure manner (either over a secure connection, or using a secure authentication mechanism that does not expose user credentials).
- `require-secure-password-changes` – Specifies whether password changes (including administrative password reset) for users associated with this policy must be processed over a secure connection.
- `allow-multiple-password-values` – Specifies whether users are allowed to have multiple passwords. Although this is technically permitted, it is difficult to maintain multiple passwords and not recommended.
- `require-change-by-time` – Requires all users with this password policy to change their passwords at least once before the specified date and time. It is similar to password expiration, but it is a one-time event and does not require password expiration to be enabled.
- `state-update-failure-policy` – Specifies how the server should behave if a failure occurred while attempting to update password policy state information in a user's entry during an authentication attempt. The value can be one of the following:
  - `proactive` – Specifies that a bind request is rejected if it is known that password policy state information cannot be updated, for example if the associated backend is operating in read-only mode.

- `reactive` – Specifies that an otherwise successful bind should be rejected if an error occurs while attempting to update password policy state information.
- `ignore` – Specifies that a successful bind should still be successful even if an error is encountered while attempting to update password policy state information.

Password encoding during LDIF Import

When performing an LDIF import, if the data being imported contains clear-text passwords, the server needs to make sure that they are properly encoded before adding them into the database. If an entry to import includes an explicit value for the `ds-pwp-passwordpolicy-dn` attribute, that password policy is retrieved and its default storage schemes are used to encode the password. For entries that do not have an explicit `ds-pwp-password-policy-dn` value, they may either be governed by the default password policy or they are governed by a policy assigned by a virtual attribute (which is not computed during import processing).

To make sure that passwords are encoded for users without an explicitly assigned policy, the server provides a password policy import plugin. For users with an explicitly-defined policy, encoding uses the default schemes for their policy. For other users, it will use the schemes configured in the plugin itself. It will automatically encode any clear-text values found in attributes with either the `userPassword` or `authPassword` syntax. Different schemes can be configured for use with each syntax.

Password policies and the PingDirectoryProxy Server

Both the PingDirectory Server and PingDirectoryProxy Server share the same password policy feature set. However, bind requests sent to an PingDirectoryProxy Server are handled differently based on whether the request uses simple or SASL authentication.

For bind requests using simple authentication, the PingDirectoryProxy Server can easily determine whether the target user is local or remote, and will always forward simple bind requests for remote users to an appropriate backend server. For SASL bind requests, it is not possible to determine the identity of the target user until much later in the bind processing, and therefore SASL binds will always be processed by the PingDirectoryProxy Server itself (although it may use information from backend servers in the process). This means that password policy processing for simple bind requests is delegated to backend servers, but for

SASL bind requests it will be performed by the PingDirectoryProxy Server itself. Make sure that PingDirectory Server and PingDirectoryProxy Server password policies are configured identically.

Chapter 9: Access control

The PingDirectory Server provides a fine-grained access control model to ensure that users are able to access the information they need, but are prevented from accessing information that they should not be allowed to see. It also includes a privilege subsystem that provides even greater flexibility and protection in many key areas.

This chapter presents the access control model and provides examples of key access control functionality.

Topics include:

Overview of access control

The access control model uses access control instructions (ACIs), which are stored in the `aci` operational attribute, to determine what a user or a group of users can do with a set of entries, down to the attribute level. The operational attribute can appear on any entry and affects the entry or any sub-entries within that branch of the directory information tree (DIT).

Access control instructions specifies four items:

- **Resources** – Resources are the targeted items or objects that specify the set of entries and/ or operations to which the access control instruction applies. For example, access can be given to certain attributes, such as the `cn` or `userPassword` password.
- **Name** – Name is the descriptive label for each access control instruction. Typically, there are multiple access control instructions for a given branch of your DIT. The access control name helps describe its purpose. For example, configure an access control instruction labeled "ACI to grant full access to administrators."
- **Clients** – Clients are the users or entities to which access is granted or denied. Specify individual users or groups of users using an LDAP URL, such as: `groupdn="ldap:///cn=admins,ou=groups,dc=example,dc=com`.
- **Rights** – Rights are permissions granted to users or client applications. Access to certain branches or operations can be denied or granted. For example, read or write permission can be granted to a `telephoneNumber` attribute.

Validation and security

The PingDirectory Server provides an access control model with strong validation to make sure that invalid ACIs are not allowed. For example, the PingDirectory Server ensures that all access control rules added over LDAP are valid and can be fully parsed. Any operation that attempts to store one or more invalid ACIs are rejected. The same validation is applied to ACIs contained in data imported from an LDIF file. Any entry containing a malformed ACI value is rejected.

As an additional level of security, the PingDirectory Server examines and validates all ACIs stored in the data whenever a backend is brought online. If any malformed ACIs are found in the backend, the server generates an alert to notify administrators of the problem and places itself in lockdown mode. While in lockdown mode, the server only allows requests from users who have the `lockdown-mode` privilege. This action enables administrators to correct the malformed ACI while ensuring that no sensitive data is inadvertently exposed. When the problem is corrected, the administrator can use the `leave-lock down-mode` tool or restart the server to resume normal operation.

Global ACIs

Global ACIs are a set of ACIs that can apply to entries anywhere in the server (although they can also be scoped so that they only apply to a specific set of entries). They work in conjunction with access control rules stored in user data and provide a convenient way to define ACIs that span disparate portions of the DIT.

In the PingDirectory Server, global ACIs are defined within the server configuration, in the `global-aci` property of configuration object for the access control handler. They can be viewed and managed using configuration tools like `dsconfig` and the Administrative Console.

The global ACIs available by default in the PingDirectory Server include:

- Allow anyone (including unauthenticated users) to access key attributes of the root DSE, including: `namingContexts`, `subschemaSubentry`, `supportedAuthPasswordSchemes`, `supportedControl`, `supportedExtension`, `supportedFeatures`, `supportedLDAPVersion`, `supportedSASLMechanisms`, `vendorName`, and `vendorVersion`.
- Allow anyone (including unauthenticated users) to access key attributes of the subschema subentry, including: `attributeTypes`, `dITContentRules`, `dITStructureRules`, `ldapSyntaxes`, `matchingRules`, `matchingRuleUse`, `nameForms`, and `objectClasses`.
- Allow anyone (including unauthenticated users) to include the following controls in requests made to the server: authorization identity request, manage DSA IT, password policy, real attributes only, and virtual attributes only.
- Allow anyone (including unauthenticated users) to request the following extended operations: get symmetric key, password modify request, password policy state, StartTLS, and Who Am I?

Access controls for public or private backends

The PingDirectory Server classifies backends as either public or private. A private backend is one whose content is generated by the PingDirectory Server itself, is used in the operation of the server (for example, the configuration, schema, task, and trust store backends), or whose content is maintained by the server (for example, the LDAP changelog backend). A public backend is intended to hold user-defined content, such as user accounts, groups, application data, and device data.

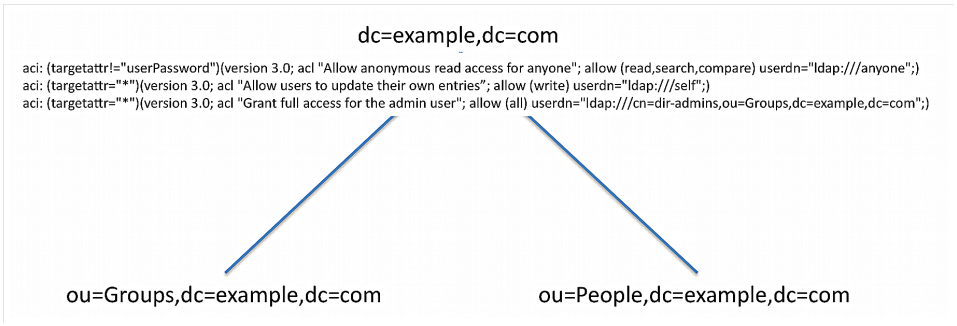
The PingDirectory Server access control model also supports the distinction between public backends and private backends. Many private backends do not allow writes of any kind from clients, and some of the private backends that do allow writes only allow changes to a specific set of attributes. As a result, any access control instruction intended to permit or restrict access to information in private backends should be defined as global ACIs, rather than attempting to add those instructions to the data for that private backend.

General format of the access control rules

Access control instructions (ACIs) are represented as strings that are applied to one or more entries within the Directory Information Tree (DIT). Typically, an ACI is placed on a subtree, and applies to that base entry and all entries below it in the tree.

The PingDirectory Server iterates through the DIT to compile the access control rules into an internally-used list of denied and allowed targets and their permissable operations. When a client application, such as `ldapsearch`, enters a request, the PingDirectory Server checks that the user who binds with the server has the necessary access rights to the requested search targets. ACIs are cumulatively applied, so that a user who has an ACI at an entry, may also have other access rights available if ACIs are defined higher in the DIT.

In most environments, ACIs are defined at the root of a main branch or a subtree, and not on individual entries unless absolutely required.



```
aci: (targetattr!="userPassword")(version 3.0; acl "Allow anonymous read access for anyone"; allow (read,search,compare) userdn="ldap:///anyone";)
aci: (targetattr="*")(version 3.0; acl "Allow users to update their own entries"; allow (write) userdn="ldap:///self";)
aci: (targetattr="*")(version 3.0; acl "Grant full access for the admin user"; allow (all) userdn="ldap:///cn=dir-admins,ou=Groups,dc=example,dc=com";)
```

An access control rule has the following syntax:

```
aci : (targets) (version 3.0; acl "name"; permissions bind rules;)
```

Access Control Components

Access Control Component	Description
targets	Specifies the set of entries and/or attributes to which an access control rule applies. Syntax: `(target keyword = || != expression)`
name	Specifies the name of the ACI.
permissions	Specifies the type of operations to which an access control rule might apply. Syntax: `allow||deny (permission)`
bind rules	Specifies the criteria that indicate whether an access control rule should apply to a given requestor. Syntax: `bind rule keyword = ||!= expression;`. The bind rule syntax requires that it be terminated with a "`;`".

Examples of common access control rules

The following examples demonstrate access controls that are commonly used. To be able to alter access control definitions in the server, a user must have the `modify-acl` privilege.

Administrator access

The following ACI can be used to enable any member of the "`cn=admins,ou=groups,dc=example,dc=com`" group to add, modify and delete entries, reset passwords and read operational attributes such as `isMemberOf` and password policy state:

```
aci: (targetattr="+")(version 3.0; acl "Administrators can read, search
  or compare operational attributes";
allow (read,search,compare) groupdn="ldap:///cn=admins,ou=groups,dc=example,dc=com";)
aci: (targetattr="*")(version 3.0; acl "Administrators can add,
  modify and delete entries";
allow (all) groupdn="ldap:///cn=admins,ou=groups,dc=example,dc=com";)
```

Anonymous and authenticated access

The following ACI allows anonymous read, search, and compare on select attributes of `inetOrgPerson` entries while authenticated users can access several more. The authenticated user will inherit the privileges of the anonymous ACI. In addition, the authenticated user can change `userPassword`:

```
aci: (targetattr="objectclass || uid || cn || mail || sn || givenName")
  (targetfilter="(objectClass=inetorgperson)")
  (version 3.0; acl "Anyone can access names and email addresses of
  entries representing people";
allow (read,search,compare) userdn="ldap:///anyone";)
aci: (targetattr="departmentNumber || manager || isMemberOf")
  (targetfilter="(objectClass=inetorgperson)")
  (version 3.0; acl "Authenticated users can access these fields for entries
  representing people";
allow (read,search,compare) userdn="ldap:///all";)
aci: (targetattr="userPassword")(version 3.0; acl "Authenticated users
  can change password";
allow (write) userdn="ldap:///all";)
```

If no unauthenticated access should be allowed to the PingDirectory Server, the preferred method for preventing unauthenticated, or anonymous access is to set the Global Configuration property `reject-unauthenticated-requests` to `true`.

Delegated access to a manager

The following ACI can be used to allow an employee's manager to edit the value of the employee's `telephoneNumber` attribute. This ACI uses the `userattr` keyword with a bind type of `USERDN`, which indicates that the target entry’s manager attribute must have a value equal to the DN of the authenticated user:

```
aci: (targetattr="telephoneNumber")
(version 3.0; acl "A manager can update telephone numbers of her direct reports";
allow (read,search,compare,write) userattr="manager#USERDN";)
```

Proxy authorization

The following ACIs can be used to allow the application "`cn=OnBehalf,ou=applications,dc=example,dc=com`" to use the proxied authorization v2 control to request that operations be performed using an alternate identity. The application user is also required to have the `proxied-auth` privilege:

```
aci: (version 3.0;acl "Application OnBehalf can proxy as another entry";
allow (proxy) userdn="ldap:///cn=OnBehalf,ou=applications,dc=example,dc=com";)
```

Validating ACIs before migrating data

Rather than unexpectedly exposing sensitive data, the PingDirectory Server rejects any ACIs that it cannot interpret, which ensures data access is properly limited. However, problems can arise when migrating data with existing access control rules to the PingDirectory Server.

To validate an access control instruction, the PingDirectory Server provides a `validate-acis` tool in the `bin` directory (UNIX or Linux systems) or `bat` directory (Windows systems) that identifies any ACI syntax problems before migrating data. The tool can examine access control rules contained in either an LDIF file or an LDAP directory, and write its result in LDIF with comments about problems that were identified. Each entry in the output contains a single ACI. Therefore, if an entry in the input contains multiple ACIs, it may be present multiple times in the output, each time with a different ACI value. The entries contained in the output contains only ACI values, and all other attributes are ignored.

Working with privileges

In addition to the access control implementation, the PingDirectory Server includes a privilege subsystem that can also be used to control what users are allowed to do. Privileged operations are only allowed if they are allowed by the access control configuration and the user has all of the necessary privileges.

Privileges can be used to grant normal users the ability to perform certain tasks that, in most other directories, would only be allowed for the root user. In fact, the capabilities extended to root users in the PingDirectory Server are all granted through privileges. A normal user account can be created with the ability to perform some or all of the same actions as root users. Multiple root users can be defined in the server with different sets of privileges so that the capabilities that they have are restricted to only the tasks that they need to be able to perform.

Available privileges

The following privileges are defined in the PingDirectory Server.

Summary of Privileges

Privilege	Description
`audit-data-security`	Required to initiate a data security audit on the server, which is invoked by the `audit-data-security` tool.
`backend-backup`	Required to initiate an online backup through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the tasks backend.
`backend-restore`	Required to initiate an online restore through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the tasks backend.
`bypass-acl`	Allows a user to bypass access control evaluation. For a user with this privilege, any access control determination is allowed. This does not bypass privilege evaluation. The user must have the appropriate set of privileges to perform any privileged operation.
`bypass-pw-policy`	Allows a user entry to bypass password policy evaluation. This privilege is intended for cases where external synchronization might require passwords that violate the password validation rules. The privilege is not evaluated for bind operations so that password policy

Summary of Privileges

<table>
<thead>
<tr><th>Privilege</th><th>Description</th></tr>
</thead>
<tbody>
<tr><td></td><td>evaluation will still occur.</td></tr>
<tr><td><code>bypass-read-acl</code></td><td>Allows a user to bypass access control checks performed by the server for bind, search, and compare operations. Access control evaluation may still be enforced for other types of operations.</td></tr>
<tr><td><code>config-read</code></td><td>Required for a user to access the server configuration. Access control evaluation is still performed and can be used to restrict the set of configuration objects that the user is allowed to see.</td></tr>
<tr><td><code>config-write</code></td><td>Required for a user to alter the server configuration. The user is also required to have the <code>config-read</code> privilege. Access control evaluation is still performed and can be used to restrict the set of configuration objects that the user is allowed to alter.</td></tr>
<tr><td><code>disconnect-client</code></td><td>Required for a user to request that an existing client connection be terminated through the disconnect client task. The server's access control configuration must also allow the user to add the corresponding entry to the tasks backend.</td></tr>
<tr><td><code>jmx-notify</code></td><td>Required for a user to subscribe to JMX notifications generated by the PingDirectory Server. The user is also required to have the <code>jmx-read</code> privilege.</td></tr>
<tr><td><code>jmx-read</code></td><td>Required for a user to access any information provided by the PingDirectory Server through the Java Management Extensions (JMX).</td></tr>
<tr><td><code>jmx-write</code></td><td>Required for a user to update any information exposed by the PingDirectory Server through the Java Management Extensions (JMX). The user is also required to have the <code>jmx-read</code> privilege. Currently, all of the information exposed by the server over JMX is read-only.</td></tr>
<tr><td><code>ldif-export</code></td><td>Required to initiate an online LDIF export through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the Tasks backend. To allow access to the Tasks backend, set up a global ACI that allows access to members of an Administrators group as follows:
<pre>$ dsconfig set-access-control-handler-prop \
  --add 'global-aci:(target="ldap:///cn=tasks")(targetattr="*||+")
   (version 3.0; acl "Access to the tasks backend for administrators";
  allow (all) groupdn="ldap:///
   cn=admins,ou=groups,dc=example,dc=com";)'</pre></td></tr>
<tr><td><code>ldif-import</code></td><td>Required to initiate an online LDIF import through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the Tasks backend. To allow access to the Tasks backend, configure the global ACI as shown in the previous description of the <code>ldif-export</code> privilege.</td></tr>
<tr><td><code>lockdown-mode</code></td><td>Allows the associated user to request that the server enter or leave lockdown mode, or to perform operations while the server is in lockdown mode.</td></tr>
<tr><td><code>modify-acl</code></td><td>Required for a user to add, modify, or remove access control rules defined in the server. The server's access control configuration must also allow the user to make the corresponding change to the <code>aci</code> operational attribute.</td></tr>
<tr><td><code>password-reset</code></td><td>Required for one user to change another user’s password. This privilege is not required for a user to change his or her own password. The user must also have the access control instruction privilege to write the <code>userPassword</code> attribute to the target entry.</td></tr>
<tr><td><code>privilege-change</code></td><td>Required for a user to change the set of privileges assigned to a user, including the set of privileges, which are automatically granted to root users. The server's access control</td></tr>
</tbody>
</table>

Summary of Privileges

Privilege	Description
	configuration must also allow the user to make the corresponding change to the `ds-privilege-name` operational attribute.
`proxied-auth`	Required for a user to request that an operation be performed with an alternate authorization identity. This privilege applies to operations that include the proxied authorization v1 or v2 control operations that include the intermediate client request control with a value set for the client identity field, or for SASL bind requests that can include an authorization identity different from the authentication identity.
`server-restart`	Required to initiate a server restart through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the Tasks backend.
`server-shutdown`	This privilege is required to initiate a server shutdown through the tasks interface. The server's access control configuration must also allow the user to add the corresponding entry in the Tasks backend.
`soft-delete-read`	Required for a user to access a soft-deleted-entry.
`stream-values`	Required for a user to perform a stream values extended operation, which obtains all entry DNs and/or all values for one or more attributes for a specified portion of the DIT.
`unindexed-search`	Required for a user to be able to perform a search operation in which a reasonable set of candidate entries cannot be determined using the defined index and instead, a significant portion of the database needs to be traversed to identify matching entries. The server's access control configuration must also allow the user to request the search.
`update-schema`	Required for a user to modify the server schema. The server's access control configuration must allow the user to update the operational attributes that contain the schema elements.

Chapter 10: Authentication Mechanisms

One of the most common uses of LDAP directory environments is as an authentication repository. This chapter highlights some properties that can be used for client authentication and authorization.

Topics include:

Configuring Authentication Types

Configuring a SASL Mechanism Handler

Configuring Certificate Mappers

Configuring Pass-Through Authentication

Preventing Bind Information Leaks

Configuring authentication types

The PingDirectory Server supports two kinds of authentication: simple and SASL.

Simple authentication allows a client to identify itself to the PingDirectory Server using the DN and password of the target user. Because the password is provided in the clear, simple authentication is inherently insecure, unless the client communication is encrypted using a mechanism like SSL or StartTLS.

Using SASL authentication mechanisms

SASL (the Simple Authentication and Security Layer, as defined in RFC 4422) is an extensible framework that includes a number of mechanisms that can use very different kinds of credentials and ways of authenticating clients. Each supported SASL mechanism is associated with a SASL mechanism handler configuration object.

- To disable certain SASL mechanisms on a server-wide basis, modify the configuration to disable the associated SASL mechanism handler.
- To disable one or more SASL mechanisms for only certain clients or to disable simple authentication for some or all clients, use the `allowed-auth-type`, `allowed-sasl-mechanism`, and/or `denied-sasl-mechanism` properties provided by Client Connection Policy configuration objects.

Controll authentication with Client Connection Policies

The following Client Connection Policy properties can be used to enforce secure authentication:

- `allowed-auth-type` – Specifies the authentication that clients are allowed to use in bind requests. Allowed values are `simple` and `sasl`. A bind request with any other type is rejected.
- `allowed-sasl-mechanism` – Specifies the names of SASL mechanisms that clients are allowed to use when authenticating. If one or more `allowed-sasl-mechanism` values are specified, then any SASL bind request that attempts to use a mechanism not included in this list is rejected. If no `allowed-sasl-mechanism` and no `denied-sasl-mechanism` values are specified, clients are allowed to use any mechanism.
- `denied-sasl-mechanism` – Specifies the names of SASL mechanisms that clients will not be allowed to use when authenticating. If a client sends a SASL bind request with a mechanism that matches one of the denied values, it is rejected. If no `allowed-sasl-`

`mechanism` and no `denied-sasl-mechanism` values are specified, clients can use any mechanism.

Controll authentication with password policies

The following password policy properties can be used to enforce secure authentication:

- `require-secure-authentication` – Indicates whether users associated with this policy are required to authenticate either over a secure connection, or using a secure authentication mechanism that does not expose user credentials.
- `require-secure-password-changes` – Indicates whether password changes including administrative password reset for users associated with this policy are required to be processed over a secure connection.

Reject or Limit unauthenticated requests

The more information that the server provides to an unauthenticated user, the greater the risk that information will be compromised. If the needs of supported client applications makes it possible, configure the server to reject all requests from unauthenticated clients.

- **Limiting Access by Global Configuration Properties** – Two global configuration properties are available to limit access to unauthenticated requests.
  - `reject-unauthenticated-requests` – Indicates whether the server should reject any requests received from a client that has not yet authenticated to the server. If enabled, the only requests that are allowed from unauthenticated clients are bind requests (to allow clients to authenticate), the StartTLS extended request, which can be used to enable secure communication before authenticating, and any requests defined in `allowed-unauthenticated-request-criteria`. If the server does not need to accept any requests from unauthenticated clients, this should be enabled.
  - `allowed-unauthenticated-request-criteria` – A set of criteria that may be used to match LDAP requests that may be permitted over an unauthenticated connection even if `reject-unauthenticated-requests` is true. Some types of requests will always be permitted, including bind, StartTLS, and start administrative session requests.
- **Custom Client Connection Policies for Unauthenticated Users** – If unauthenticated client applications do need to perform a limited set of operations prior to authenticating, create a custom Client Connection Policy for unauthenticated users that allows requests for those operations. After they authenticate and are assigned a different Client Connection Policy, they can be granted a greater number of operations.

- **Limiting Access by Unauthenticated Users using Access Controls** – Limiting access by unauthenticated users can also be accomplished through access control configuration. The server's default access control policy provides limited access (including the ability to retrieve selected attributes from the root DSE or server schema, and the ability to issue certain extended requests like StartTLS and Who Am I?), and does not allow any access to user data unless that data itself contains ACIs that allow it.

Restrict authentication with operational attributes

Authentication type restrictions can be enforced globally or through Client Connection Policies. Operational attributes associated with individual user account entries can also be used to enforce a number of constraints. The following operational attributes can restrict authentication. They can either be implemented as explicitly-provided values or as virtual attributes.

- `ds-auth-allowed-address` – Specifies the set of addresses from which that user is allowed to authenticate. Values can be specified address masks including individual IP addresses or resolvable names, addresses with wildcards, CIDR address ranges, or IP addresses with subnet masks.
- `ds-auth-allowed-authentication-type` – Specifies the kinds of authentication that is allowed for that user, which can be `simple` and `sasl {mechanism}`.
- `ds-auth-require-secure-authentication` – Specifies whether the user is required to authenticate in a secure manner that ensures credentials are not exposed to anyone with the ability to observe network communication. If this attribute exists in a users' entry with a value of `true`, that user can only authenticate over a secure connection or using an authentication mechanism (like CRAM-MD5, DIGEST-MD5, or GSSAPI) that does not expose the client's credentials.
- `ds-auth-require-secure-connection` – Specifies whether the user is required to access the server in a secure manner. If this attribute exists in a user's entry with a value of `true`, that user can only issue requests over a connection secured with SSL or StartTLS.
- `ds-auth-is-proxyable` – Specifies whether the user can be specified as an alternate authorization identity through the use of the proxied authorization control, intermediate client control, or a SASL mechanism that supports specifying an alternate authorization identity. The value can be one of the following:
    - `allowed` – Indicates that the account can be accessed either by direct authentication or as an alternate authorization identity.

  - `required` – Indicates that the account can only be accessed as an alternate authorization identity, and is not allowed to directly authenticate.
  - `prohibited` – Indicates that the account can only be accessed through direct authentication but not as an alternate authorization identity.
- `ds-auth-is-proxyable-by`– Indicates which users are allowed to access a user in the form of an alternate authorization identity. If this attribute is present in a user's entry, users whose DNs are included in the value of that attribute are allowed to specify that user as an alternate authorization identity.

Use certificate-based authentication

PingData servers provide the ability to use a client certificate presented to the server during SSL or StartTLS negotiation as the set of credentials for LDAP authentication. There are two ways that this can be accomplished:

- The client application can send a SASL EXTERNAL bind request to the server.
- The client application can present a certificate to the server (the LDAP connection handler that accepts the connection must have the `auto-authenticate-using-client-certificate` property set to `true`.)

In either case, the authentication is processed as if the client had requested SASL EXTERNAL authentication, and the SASL EXTERNAL mechanism handler is used to process the authentication. Refer to the *PingData PingDirectory Server Administration Guide* for information about the certificate mapper configuration properties. The SASL mechanism handler includes the following configuration properties:

- `certificate-mapper` – Specifies which certificate mapper should be used to identify the user to be authenticated.
- `certificate-validation-policy` – Specifies whether the server should attempt to find the certificate presented by the client in the user's entry. Values are:
  - `always` – The server must find the presented certificate in the user's entry for authentication to succeed.
  - `never` – The server will not look in the user entry for the certificate.
  - `ifpresent` – If the user's entry contains one or more certificates, one of them must match the certificate presented by the client, but authentication will be allowed if the user's entry does not have any certificates.
- `certificate-attribute` – Specifies the name of the attribute that will be checked for certificates if the `certificate-validation-policy` property has a value of `always` or

`ifpresent`.

Certificate mappers

Certificate mappers are used to identify the user attempting to authenticate based on information contained in the certificate presented to the server. The certificate mapper can make use of any information contained in the certificate, including the subject, extensions, or certificate fingerprint. The server provides a number of certificate mapper implementations:

- **Subject equals DN** – Matches a user whose DN is the same as the certificate subject.
- **Subject DN to user attribute** – Matches a user whose entry contains a specified attribute with a value equal to the subject of the presented certificate. If this mapper is used, the attribute holding the certificate subjects must be indexed for equality.
- **Subject attribute to user attribute** – Takes attributes from the presented certificate and uses them to generate a filter for an internal search to identify the target user. It is possible to customize the mapping between subject attributes and user entry attributes (for example, "E" in the certificate subject may map to "mail" in the user's entry).
- **Fingerprint** – Performs an internal search to find a user entry in which the value of a specified attribute matches the SHA-1, 256-bit SHA-2, or MD5 fingerprint of the presented certificate. If this mapper is to be used, the attribute holding the fingerprint in user entries must be indexed for equality.
- **Custom** – The Server SDK can also be used to develop custom certificate mappers.

Configure a SASL mechanism handler

The `dsconfig` utility enables configuration of the following SASL mechanism handlers:

ANONYMOUS – Does not perform any authentication, but can be enabled for clients to include a trace string to identify the purpose of a connection.

CRAM-MD5 – Performs password-based authentication through an MD5 digest. The client sends a bind request to the server. The server responds with a randomly-generated challenge to protect against replay attacks. The client responds with an answer to the challenge, a clear-text password, and an authentication ID. The server encodes the password and requires that any clients have a password policy that supports two-way, reversible encryption.

By default, SASL DIGEST-MD5 uses the Exact Match identity mapper, which returns a success result if the authorization ID is an exact match for the value of the `uid` attribute. Other identity mappers, such as the Regular Expression identity mapper or a custom mapper, can also be used.

DIGEST-MD5 – Provides authentication through a stronger MD5 digest that does not expose a clear-text password. The client sends a bind request with credentials to the server. The server sends the client a response with a set of authentication options and a special token. The client sends an encrypted response with the chosen authentication method. The server then validates the client's response. This is the required authentication mechanism for LDAP v3 servers.

EXTERNAL – Allows a client to authenticate using information about the client, which is available to the server, but is not directly provided over LDAP. On the server, SASL EXTERNAL requires the use of a client certificate provided during SSL or StartTLS negotiation. This does not require the use of passwords, although its use on a broad scale is generally only feasible in environments with a PKI deployment.

GSSAPI – Provides authentication for LDAP clients using Kerberos V. User credentials are stored in the Kerberos key distribution center (KDC) rather than the PingData server. When an LDAP client attempts to authenticate with the server, a three-way exchange occurs that allows the client to verify its identity to the server through the KDC.

PingData's support for GSSAPI is based on the Java Authentication and Authorization Service (JAAS). By default, the server automatically generates a JAAS configuration that should be appropriate for most use cases. For more complex deployments, a custom JAAS configuration can be supplied.

PingData servers support GSSAPI only for authenticating clients, not for securing their communication with the server.

PLAIN – Performs password-based authentication with an authentication ID, clear-text password, and optional authorization ID.

UNBOUNDID-CERTIFICATE-PLUS-PASSWORD – Similar to the standard SASL EXTERNAL mechanism in that the user is identified by the certificate that the client presents during SSL or StartTLS negotiation (and the configured certificate mapper is used to identify the user to whom the certificate belongs). But whereas the certificate is the only thing needed to authenticate with the EXTERNAL mechanism, the UNBOUNDID-CERTIFICATE-PLUS-PASSWORD also requires that the user provide their static password as a form of two-factor authentication.

UNBOUNDID-TOTP – Provides a proprietary multifactor authentication mechanism that allows the server to use the Time-based One-Time Password (TOTP) algorithm, specified in RFC 6238. The TOTP algorithm is an extension of the Hash-based Message Authentication Code One-Time Password (HTOP) algorithm, specified in RFC 4226. The TOTP algorithm computes a temporary code using the current time and a secret key that is shared between the client application and the server.

UNBOUNDID-TOTP issues a bind request that includes at least an authentication ID and a TOTP code, but may also include an authorization ID and/or a static password. The server first uses the authentication ID to identify the user that is authenticating and then retrieves the shared secret from the user's entry (stored as a base32-encoded value in the `ds-auth-totp-shared-secret` operational attribute) and uses it with the current time to generate a TOTP code. If that matches the code that the user entered, then that confirms that the client knows the shared secret. If a static password was also provided, then the server will confirm that it matches what is stored in the `userPassword` attribute (or that specified by the password policy). By default, the server will require the client to provide a static password.

The Commercial Edition of the LDAP SDK for Java provides the necessary client-side support for the UNBOUNDID-TOTP mechanism, and provides a `com.unboundid.ldap.sdk.unboundidds.OneTimePassword` class to generate TOTP codes for testing purposes.

UNBOUNDID-DELIVERED-OTP – Provides two-factor authentication, which uses one-time passwords (OTPs) that are delivered to the end user through an out-of-band mechanism. The server provides support for e-mail (through an external SMTP external server), SMS (through the Twilio web service), and custom delivery mechanisms with the Server SDK.

The process for authenticating using this new mechanism involves two steps:

- The client sends a "deliver one-time password" extended request to the server. This request includes an authentication ID, the user's static password, and an optional set of allowed delivery mechanisms. If successful, the server generates a one-time password, stores it in the user's entry, and sends it to the user through one of the allowed mechanisms.
- Once the user has received the one-time password, the client should perform an UNBOUNDID-DELIVERED-OTP bind (which may be on the same connection or a different connection used to send the "deliver one-time password" extended operation). The

credentials for this SASL mechanism include an authentication ID to identify the user, an optional authorization ID (if operations performed by the client should be authorized as a different user), and the one-time password.

Unlike UNBOUNDID-TOTP, there is no need to have a shared secret between the client and the server, or any special client-side software to generate the one-time password, or a need to worry about whether the client and server clocks are synchronized.

UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION – Provides authentication involving credentials that do not reside in, or cannot be forwarded to or validated by the PingDirectory Server (such as social login through Facebook, Google, or Twitter). The bind request will not include any credentials, and authentication with this mechanism will not actually change the state of the underlying client connection. The server will behave as if the bind request included the retain identity request control, regardless of whether or not that control was included.

Bind requests using this mechanism can include any request controls that are permitted for use with other bind requests. If the externally-processed authentication is successful, the client can include the get password policy state issues request control in the bind request to obtain information about any password policy state issues that can cause the PingDirectory Server authentication attempt to fail. The password policy request control can also be included to obtain certain password policy state warnings and errors, or to look for the password expired/password expiring controls in the bind response.

By default, the UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION mechanism uses the Exact Match Identity Mapper, which returns a success result if the authorization ID is an exact match for the value of the `uid` attribute. Other identity mappers, such as the Regular Expression Identity Mapper or a custom mapper, can also be used.

When an UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION bind request is received, the PingDirectory Server validates the request by:

- Decoding the bind request to make sure that it is properly-formed. If not, the authentication attempt is rejected with a `PROTOCOL_ERROR` result. If the bind request is successfully decoded, the operation is updated so that the bind result access log message includes all relevant details from the request.
- Making sure that the bind request was received on a connection that was already authenticated as a user with either the `permit-externally-processed-`

authentication privilege. If not, then the authentication attempt will be rejected with an AUTH_METHOD_NOT_SUPPORTED result.

- Resolving and retrieving the entry indicated by the authenticationID element of the bind request. If this entry does not exist, the authentication attempt is rejected with an INVALID_CREDENTIALS result.

The remainder of the processing depends on whether the externally-processed authentication is considered successful, as determined from the externalAuthenticationWasSuccessful result. If successful, the server will first determine the state of the target user account. If the account is not in a useable state, the bind operation will fail with an INVALID_CREDENTIALS result. The server will also determine whether the authentication should be allowed. If the bind should not be permitted, the operation will fail with an INVALID_CREDENTIALS result.

If the authentication is still considered successful after all PingDirectory Server validation, a number of password policy state updates may be applied, including clearing any previous authentication failures, updating the last login time, or updating the last login IP address. The server will then return a SUCCESS result to the client.

For a failed attempt, the server may increment the number of failed authentication attempts and, if appropriate, lock the account.

UNBOUNDID-YUBIKEY-OTP – Provides authentication with YubiKey devices (available from Yubico), which generate secure one-time passwords. The device generates a different password for every authentication attempt, and that one-time password is sent to a validation service to ensure that it is genuine and has not been used in an earlier authentication attempt. Although it is possible to use this one-time password as the only proof of identity, it is typically combined with a static password as a form of two-factor authentication.

Supported YubiKey devices include:

- YubiKey 4
- YubiKey 4 Nano
- YubiKey NEO
- YubiKey NEO-n
- YubiKey Edge
- YubiKey Edge-n

- YubiKey Standard
- YubiKey Nano

This mechanism is not enabled by default, and requires configuration of a client ID and API key to use when communicating with the validation service. The API key is a shared secret between the YubiKey validation service and the client that is interacting with it (the PingData server), and is used when generating digital signatures so that both the PingData server and the YubiKey validation service can ensure that the peer server is genuine. This signature provides protection against man-in-the-middle attacks in which the PingDirectory Server server might be tricked into communicating with a fraudulent validation server.

The API key must be kept secret. A malicious user who is able to obtain the API key may be able to generate fake responses that appear to be valid. Because of its sensitive nature, the API key itself is never included in the communication between the PingData server and the validation service. However, the client ID is linked to the API key, so that the validation service knows which API key to use in conjunction with a given client ID. All communication between PingData servers and the validation service includes the client ID, and also includes signatures generated with the corresponding API key. If either party is unable to validate a signature, the authentication attempt will fail. To obtain a client ID and API key for free, navigate to https://upgrade.yubico.com. Only one client ID/API key pair is needed for all users across all servers in the topology.

When authenticating a user with a YubiKey one-time password, the server will communicate with the validation service to ensure that the one-time password is valid, and it will also ensure that the public ID from that one-time password is contained in the `ds-auth-yubikey-public-id attribute` in the target user's entry. If the user's entry does not include the public ID for that YubiKey device, authentication will fail.

Configure SASL ANONYMOUS mechanism

The LDAP client tools provided with PingData servers support the use of SASL ANONYMOUS. The optional "trace" SASL option can be used to specify the trace string to include in the bind request.

Perform the following steps to configure SASL ANONYMOUS:

1. Use `dsconfig` to enable the SASL ANONYMOUS mechanism.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name ANONYMOUS \
  --set enabled:true
```

2. Use `ldapsearch` to view the root DSE and enter a trace string in the access log.

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=ANONYMOUS \
  --saslOption "trace=debug trace string" --baseDN "" \
  --searchScope base "(objectclass=*)"
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 59bab79d-4429-49c8-8a88-c74a86792f26
```

3. View the access log using a text editor in the `/ds/PingData<server>/logs` folder.

```
[26/Oct/2011:16:06:33 -0500] BIND RESULT conn=2 op=0 msgID=1
resultCode=0
additionalInfo="trace='debug trace string'" etime=345.663
clientConnectionPolicy="default"
```

Configure SASL CRAM-MD5 mechanism

CRAM-MD5 requires an authentication ID (`authid`) from the client to identify the authenticating user. The format of that authentication ID can be either:

- `dn:` followed by the distinguished name of the target user (or just `dn:` to perform an anonymous bind).
- `u:` followed by a username.

If using `u:`, an identity mapper is used to identify the target user based on that username.

Perform the following steps to configure CRAM-MD5:

1. Use `dsconfig` to enable the SASL CRAM-MD5 mechanism if it is disabled. By default, the CRAM-MD5 mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name CRAM-MD5 \
  --set enabled:true
```

2. For this example, create a password policy for CRAM-MD5 using a reversible password storage scheme, like 3DES.

```
$ bin/dsconfig create-password-policy \
  --policy-name "Test UserPassword Policy" \
  --set password-attribute:userpassword \
  --set default-password-storage-scheme:3DES
```

3. Use `ldapmodify` to add the `ds-pwp-password-policy-dn` attribute to an entry to indicate the Test UserPassword Policy should be used for that entry. When finished, press CTRL-D to process the modify operation.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=Test UserPassword Policy,cn=Password
Policies,cn=config
```

```
Processing MODIFY request for uid=jdoe,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=jdoe,ou=People,dc=example,dc=com
```

4. Use `ldapmodify` to change the `userPassword` to a reversible password storage scheme. The password storage scheme is specified in the user’s password policy.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: secret
```

5. Use `ldapsearch` to view the root DSE using the authentication ID (`authid`) option with the username jdoe. Enter a password for the user.

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=CRAM-MD5 \
  --saslOption "authid=u:jdoe" --baseDN "" \
  --searchScope base "(objectclass=*)"
  Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 50567aa3-acd2-4106-a077-37a092275363
```

Configure SASL DIGEST-MD5 mechanism

DIGEST-MD5 requires an authentication ID (`authid`) from the client to identify the authenticating user. The format of that authentication ID can be either:

- `dn:` followed by the distinguished name of the target user (or just `dn:` to perform an anonymous bind).
- `u:` followed by a username. If using `u:`, an identity mapper is used to identify the target user based on that username.

The client may also include the following properties:

- `authzID` – Specifies an optional authorization ID that should be used for operations processed on the connection.
- `realm` – The realm in which the authentication should be processed. This may or may not be required, based on the server configuration.
- `digest-uri` – The digest URI that should be used for the bind. It should generally be "ldap://" followed by the fully-qualified address for the PingDataMetrics Server. If this is not provided, then a value will be generated.
- `qop` – The quality of protection to use for the bind request. Only `auth` is supported (indicating that the DIGEST-MD5 bind should only be used for authentication and should not provide any subsequent integrity or confidentiality protection for the connection), and if no value is provided then auth will be assumed.

Perform the following steps to configure CRAM-MD5:

1. Use `dsconfig` to enable the SASL DIGEST-MD5 mechanism if it is disabled. By default, the DIGEST-MD5 mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name DIGEST-MD5 \
  --set enabled:true
```

2. For this example, create a password policy using a reversible password storage scheme, like 3DES.

```
$ bin/dsconfig create-password-policy \
  --policy-name "Test UserPassword Policy" \
  --set password-attribute:userpassword \
  --set default-password-storage-scheme:3DES
```

3. Use `ldapmodify` to add the `ds-pwp-password-policy-dn` attribute to an entry to indicate the Test UserPassword Policy should be used for that entry. When finished, press CTRL-D to process the modify operation.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=Test UserPassword Policy,cn=Password
Policies,cn=config
```

```
Processing MODIFY request for uid=jdoe,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=jdoe,ou=People,dc=example,dc=com
```

4. Use `ldapmodify` to change the `userPassword` to a reversible password storage scheme. The password storage scheme is specified in the user’s password policy.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
```

```
replace: userPassword
userPassword: secret
```

5. Use `ldapsearch` to view the root DSE using the authentication ID with the username `jdoe`. Enter a password for the authentication ID.

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=DIGEST-MD5 \
  --saslOption "authid=u:jdoe" --baseDN "" \
  --searchScope base "(objectclass=*)"
  Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

6. The user identified by the authentication ID requires the `proxied-auth` privilege to allow it
   to perform operations as another user.

```
$ bin/ldapmodify

dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modifyadd: ds-privilege-name
ds-privilege-name: proxied-auth
```

7. Use `ldapsearch` with the `authid` (required) and `authzid` option to test the mechanism.

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=DIGEST-MD5 \
  --saslOption authid=u:jdoe \
  --saslOption authzid=dn:uid=admin,dc=example,dc=com \
  --base "" \
  --searchScope base "(objectclass=*)"
  Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

Configure SASL EXTERNAL mechanism

Prior to the SASL EXTERNAL session exchange, the client should have successfully established a secure communication channel using SSL or StartTLS, and the client must have presented its own certificate to the server. The SASL EXTERNAL bind request does not contain any credentials. The server only uses the information contained in the provided client certificate to identify the target user.

The configuration settings for SASL EXTERNAL includes three required properties:

- `certificate-validation-policy` – Checks if the certificate presented by the client is present in the target user's entry. Possible values are:
  - `always` – Always require the peer certificate to be present in the user's entry. Authentication will fail if the user's entry does not contain any certificates, or if it contains one or more certificates and the certificate presented by the client is not included in the user's entry.
  - `ifpresent` – (Default) If the user's entry contains one or more certificates, require that one of them match the peer certificate. Authentication will succeed if the user's entry does not have any certificates, but will fail if the user's entry has one or more certificates that do not match the certificate provided by the client.
  - `never` – Do not look for the peer certificate to be present in the user's entry. Authentication will succeed if the user's entry does not contain any client certificates, or if it contains certificates that do not match the certificate provided by the client.
- `certificate-attribute` – Specifies the attribute that holds user certificates to be examined if the `certificate-validation-policy` attribute has a value of `ifpresent` or `always`. The name must be a valid attribute type defined in the server schema. The default value is `userCertificate`. LDAP generally requires certificate values to use the `;binary` attribute modifier. Certificates should be stored in user entries using the attribute `userCertificate;binary`.
- `certificate-mapper` – Specifies the certificate mapper that will be used to identify the target user based on the certificate presented by the client.

Perform the following to configure the EXTERNAL mechanism:

1. Change to the server root directory.

   ```
   $ cd /ds/PingData<server>
   ```

2. Determine the `certificate-validation-policy` property. If not storing the DER-encoded representation of the client's certificate in the user's entry, skip to the next step.

   If `always` is chosen, make sure that the user's entry has a valid value. If `ifpresent` is selected, the `userCertificate` attribute can also be present. The client's certificate can be stored in the user entry using `ldapmodify`.

   ```
   $ bin/ldapmodify

   dn: uid=jdoe,ou=People,dc=example,dc=com
   changetype: modify
   ```

```
add: userCertificate;binary
userCertificate;binary:<file:///path/to/client.der
```

3. If using an attribute other than `userCertificate`, specify it using the `certificate-attribute` property. Make sure that the schema is updated to support the attribute.
4. Determine the `certificate-mapper` property. For more information about certificate mappers, see Configure certificate mappers.
5. Use `dsconfig` to enable the SASL EXTERNAL mechanism if it is disabled. By default, the SASL mechanism is enabled. For this example, set the `certificate-mapper` property to Subject Attribute to User Attribute. All other defaults are kept.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
    --handler-name EXTERNAL \
    --set enabled:true \
    --set "certificate-mapper:Subject Attribute to User Attribute"
```

6. Use `ldapsearch` to test SASL EXTERNAL.

```
$ bin/ldapsearch --port 1636 \
    --useSSL \
    --keyStorePath /path/to/clientkeystore \
    --keyStorePasswordFile /path/to/clientkeystore.pin \
    --trustStorePath /path/to/truststore \
    --saslOption mech=EXTERNAL \
    --baseDN "" \
    --searchScope base "(objectClass=*)
```

Configure SASL GSSAPI mechanism

While the GSSAPI specification includes a provision for protecting client-server communication, PingData servers currently support GSSAPI only for the purpose of authenticating clients.

Kerberos configuration considerations

To implement GSSAPI authentication, a Kerberos V deployment must be configured. The Kerberos deployment should take the following into consideration:

- It is recommended that the KDC be configured to use "aes128-cts" as the TKT and TGS encryption type, which is supported by all Java VMs. In Kerberos environments using the MIT libraries, make sure that the following lines are present in the `[libdefaults]` section of the `/etc/krb.conf` configuration file on the KDC system:

```
default_tkt_enctypes = aes128-cts
default_tgs_enctypes = aes128-cts
permitted_enctypes = aes128-cts
```

- When a client uses Kerberos to authenticate to a server, the addresses of the target server and the KDC are used in cryptographic operations. Make sure that all systems agree on the addresses of the server and KDC systems. Make sure that DNS is configured so that the primary addresses for the KDC and server are addresses that clients will use to communicate.
- Kerberos authentication is time-sensitive. If system clocks are not synchronized, authentication may fail. Use NTP or some other form of time synchronization for all KDC, server, and client systems.

To authenticate itself to the Kerberos environment, the KDC should include both host and service principals for all servers. The host principal is in the form `host/directory.example.com`, and the service principal should generally be `ldap/directory.example.com`. In an MIT Kerberos environment, the `kadmin` utility can be used to create these principals, as follows:

```
# /usr/sbin/kadmin -p kws/admin
Authenticating as principal kws/admin with password.
Password for kws/admin@EXAMPLE.COM:
kadmin: add_principal -randkey host/directory.example.com
WARNING: no policy specified for host/directory.example.com@EXAMPLE.COM;
  defaulting to no policy
Principal "host/directory.example.com@EXAMPLE.COM" created.
kadmin: ktadd host/directory.example.com
Entry for principal host/directory.example.com with kvno 3, encryption type AES-128
  CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: add_principal -randkey ldap/directory.example.com
WARNING: no policy specified for ldap/directory.example.com@EXAMPLE.COM;
  defaulting to no policy
Principal "ldap/directory.example.com@EXAMPLE.COM" created.
kadmin: quit
```

On each server, the service principal for that instance must be exported to a keytab file, using a command such as:

```
# /usr/sbin/kadmin -p kws/admin
Authenticating as principal kws/admin with password.
Password for kws/admin@EXAMPLE.COM:
kadmin: ktadd -k /ds/PingData-<server>/config/server.keytab ldap/
directory.example.com
Entry for principal ldap/directory.example.com with kvno 4, encryption type AES-128
  CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/ds/PingDataMetricsServer/
config/
  server.keytab.
kadmin: quit
```

Because this file contains the credentials that the server will use to authenticate to the KDC, make sure that it is only accessible by the server.

GSSAPI mechanism handler options

The GSSAPI SASL mechanism handler provides the following configuration options:

- `enabled` – Indicates whether the GSSAPI SASL mechanism handler is enabled for use in the server. By default, it is disabled.
- `kdc-address` – Specifies the address that the server uses to communicate with the KDC. If this is not specified, the server uses the underlying system configuration.
- `server-fqdn` – Specifies the fully-qualified domain name that clients use to communicate with the server. If this is not specified, the server uses the underlying system configuration.
- `realm` – Specifies the Kerberos realm that clients use. If this is not specified, the server uses the underlying system configuration.
- `kerberos-service-principal` – Specifies the service principal that the server uses to authenticate with the KDC. If this is not specified, the service principal is `ldap/` followed by the fully-qualified server address.
- `keytab` – Specifies the path to the keytab file that holds the credentials for the Kerberos service principal that the server uses to authenticate with the KDC. If this is not specified, the server uses the system-wide keytab.
- `identify-mapper` – Specifies the identify mapper that the server uses to map a client's Kerberos principal to the entry of the corresponding user account in the server. In the default configuration, the server uses a regular expression identity mapper that looks for an entry with a `uid` value equal to the username portion of the Kerberos principal. For example, for a Kerberos principal of `jdoe@EXAMPLE.COM`, the identity mapper will perform an internal search with a filter of (`uid=jdoe`).
- `enable-debug` – Indicates whether the server should write debugging information about Kerberos-related processing (including JAAS processing) that the server performs. If enabled, this information is written to standard error in the `logs/server.out` log file.
- `jaas-config` file – Specifies the path to a JAAS configuration file that the server should use. If this is not specified, the server generates a JAAS configuration file based on the values of the other configuration properties. This should only be used when the server-generated JAAS configuration is not acceptable.

Configure SASL PLAIN mechanism

LDAP clients can use SASL PLAIN with the following SASL options:

- `authid` – Specifies the authentication ID to use for the bind. This must be provided.
- `authzid`– Specifies an optional alternate authorization ID to use for the bind.

Perform the following steps to configure SASL PLAIN:

1. Use `dsconfig` to enable the SASL PLAIN mechanism.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name PLAIN \
  --set enabled:true
```

2. Use `ldapsearch` to view the root DSE using the authentication ID (`authid`) with the username jdoe. Enter a password for the authentication ID.

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=PLAIN \
  --saslOption "authid=u:jdoe" --baseDN "" \
  --searchScope base "(objectclass=*)"
  Password for user 'u:jdoe':
```

   Or specify the full DN of the user:

```
$ bin/ldapsearch --port 1389 \
  --saslOption mech=PLAIN \
  --saslOption "authid=dn:uid=jdoe,ou=People,dc=example,dc=com" \
  --baseDN "" \
  --searchScope base "(objectclass=*)"
  Password for user 'dn:uid=jdoe,ou=People,dc=example,dc=com':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 59bab79d-4429-49c8-8a88-c74a86792f26
```

Configure the UNBOUNDID-CERTIFICATE-PLUS-PASSWORD mechanism

Perform the following to configure the UNBOUNDID-CERTIFICATE-PLUS-PASSWORD mechanism:

1. Change to the server root directory.

```
$ cd /ds/PingData<server>
```

2. Determine the `certificate-validation-policy` property. If not storing the DER-encoded representation of the client's certificate in the user's entry, skip to the next step.

   If `always` is chosen, make sure that the user's entry has a valid value. If `ifpresent` is selected, the `userCertificate` attribute can also be present. The client's certificate can be stored in the user entry using `ldapmodify`.

```
$ bin/ldapmodify

dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: userCertificate;binary
userCertificate;binary:<file:///path/to/client.der
```

3. If using an attribute other than `userCertificate`, specify it using the `certificate-attribute` property. Make sure that the schema is updated to support the attribute.
4. Determine the `certificate-mapper` property. For more information about certificate mappers, see Configure certificate mappers.
5. Use `dsconfig` to enable the SASL UNBOUNDID-CERTIFICATE-PLUS-PASSWORD mechanism if it is disabled. For this example, set the `certificate-mapper` property to Subject Attribute to User Attribute. All other defaults are kept.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
    --handler-name UNBOUNDID-CERTIFICATE-PLUS-PASSWORD \
    --set enabled:true \
    --set "certificate-mapper:Subject Attribute to User Attribute"
```

Configure SASL UNBOUNDID-TOTP mechanism

Perform the following steps to configure the UNBOUNDID-TOTP mechanism:

1. Configure the server so that `ds-auth-totp-shared-secret` is a sensitive attribute that can only be set over a secure connection and not retrieved from the server. Create the sensitive attribute and reference it from the global configuration using `dsconfig`:

```
$ bin/dsconfig create-sensitive-attribute \
  --attribute-name ds-auth-totp-shared-secret \
  --set attribute-type:ds-auth-totp-shared-secret \
  --set allow-in-returned-entries:suppress \
  --set allow-in-filter:reject \
  --set allow-in-compare:reject \
  --set allow-in-add:secure-only \
  --set allow-in-modify:secure-only

$ bin/dsconfig set-global-configuration-prop \
  --add sensitive-attribute:ds-auth-totp-shared-secret
```

2. Update a user entry so that it contains a `ds-auth-totp-shared-secret` attribute with a value that holds the base32-encoded shared secret that will be used for TOTP authentication. This should be done over a secure connection (SSL or StartTLS). There is no maximum limit for the `ds-auth-totp-shared-secret` string, but there is a minimum length of 16 base32-encoded characters.

```
dn: uid=user.0,ou=People,dc=example,dc=com
changetype: modify
add: ds-auth-totp-shared-secret
ds-auth-totp-shared-secret: ONSWG4TFORRW6ZDF
```

3. Use `ldapsearch` to test the configuration.

```
$ bin/ldapsearch --saslOption mech=UNBOUNDID-TOTP \
  --saslOption authID=u:user.0 \
  --saslOption totpPassword=628094 \
  --bindPassword password \
  --baseDN "" \
  --searchScope base \
  "(objectClass=*)"
```

Configure SASL UNBOUNDID-DELIVERED-OTP mechanism

Perform the following steps to configure the UNBOUNDID-DELIVERED-OTP mechanism:

1. Add support for one or more one-time password delivery mechanisms. The following commands enable an SMTP external server, associate it with the global configuration, and create the delivery mechanism.

```
$ bin/dsconfig create-external-server \
    --server-name "Intranet SMTP Server" \
    --type smtp \
    --set server-host-name:server.example.com
```

```
$ bin/dsconfig set-global-configuration-prop \
    --add "smtp-server:Intranet SMTP Server"
```

```
$ bin/dsconfig create-otp-delivery-mechanism \
    --mechanism-name E-Mail \
    --type email \
    --set enabled:true \
    --set 'sender-address:otp@example.com' \
    --set "email-address-attribute-type:mail" \
    --set "message-subject:Your one-time password" \
    --set "message-text-before-otp:Your one-time password: "
```

2. With a Twilio account, configure the server to deliver one-time passwords over SMS.

```
$ bin/dsconfig create-otp-delivery-mechanism \
    --mechanism-name SMS \
    --type twilio \
    --set enabled:true
    --set twilio-account-sid:xxxxx \
    --set twilio-auth-token:xxxxx \
    --set "sender-phone-number:xxxxx" \
    --set phone-number-attribute-type:mobile \
    --set "message-text-before-otp:Your one-time password: "
```

3. With OTP delivery mechanisms established, configure the extended operation handler.

```
$ bin/dsconfig create-extended-operation-handler \
    --handler-name "Deliver One-Time Password" \
    --type deliver-otp \
    --set enabled:true \
    --set "identity-mapper:Exact Match" \
    --set "password-generator:One-Time Password Generator" \
    --set default-otp-delivery-mechanism:SMS \
    --set default-otp-delivery-mechanism:E-Mail
```

4. Configure the SASL mechanism handler.

```
$ bin/dsconfig create-sasl-mechanism-handler \
    --handler-name UNBOUNDID-DELIVERED-OTP \
    --type unboundid-delivered-otp \
    --set enabled:true \
    --set "identity-mapper:Exact Match" \
    --set "otp-validity-duration:5 minutes"
```

5. Make sure the server contains a user account with the information needed to deliver the one-time password, such as a valid email address or mobile number.
6. Use the deliver one-time password extended operation to have the server generate and send a one-time password to the user. The Commercial Edition of LDAP SDK contains support for the extended request and response needed to do this, or use the `deliver-one-time-password` command-line tool:

   ```
   $ bin/deliver-one-time-password \
       --userName jdoe \
       --promptForBindPassword \
       --deliveryMechanism SMS

   Enter the static password for the user:

   Successfully delivered a one-time password via mechanism 'SMS' to '123-
   456-7890'
   ```

   If processed successfully, a text message is received:

   ```
   Your one-time password: 123456
   ```

7. Authenticate to the server using the UNBOUNDID-DELIVERED-OTP SASL mechanism. The Commercial Edition of the LDAP SDK can be used, or the `ldapsearch` tool:

   ```
   $ bin/ldapsearch \
       -o mech=UNBOUNDID-DELIVERED-OTP \
       -o authID=u:jdoe \
       -o otp=123456 \
       -b '' \
       -s base '(objectClass=*)' \
       ds-supported-otp-delivery-mechanism
   ```

   The search returns:

   ```
   dn:
   ds-supported-otp-delivery-mechanism: E-Mail
   ds-supported-otp-delivery-mechanism: SMS
   ```

Configuring the UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION mechanism

If not enabled, use `dsconfig` to enable this mechanism. By default, the UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
    --handler-name UNBOUNDID-EXTERNALLY-PROCESSED-AUTHENTICATION \
```

```
    --set enabled:true \
    --set "identity-mapper:Exact Match"
```

Configure the UNBOUNDID-YUBIKEY-OTP mechanism

Once the YubiKey client ID and API key are obtained, edit the server configuration for the UNBOUNDID-YUBIKEY-OTP SASL mechanism handler and make the following changes:

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name UNBOUNDID-YUBIKEY-OTP \
  --set enabled:true \
  --set yubikey-client-id:<12345> \
  --set "yubikey-api-key:<abcdefghijklmnopqrstuvwxyz=>"
```

The following properties can also be confgured:

- `identity-mapper` — Specifies the identity mapper used to determine which user is trying to authenticate for a provided username. The default Exact Match identity mapper assumes that the provided username exactly matches the value of either the `uid` or the `mail` attribute in the target user's entry.
- `require-static-password` — Indicates whether users are required to supply a static password when authenticating to the server with a YubiKey one-time password. By default, a static password is required for two-factor authentication. If this properly is set to `false`, the one-time password is permitted by itself for single-factor authentication. However, if an UNBOUNDID-YUBIKEY-OTP bind request includes a static password, that password is verified even if `require-static-password` is set to `false`.

If using a validation service other than the one provided by Yubico, configure the following properties:

- `yubikey-validation-server-base-url` — Specifies the HTTP or HTTPS URLs of any servers that will be used to validate the one-time passwords generated by YubiKey devices. Multiple values can be provided. In the event that the first server is unavailable or unresponsive, the next server is accessed until one is available.
- `key-manager-provider` — Specifies the key manager provider that should be used if the server is to communicate with the validation service over HTTPS, and must provide its own client certificate. In most deployments, this should be left blank.
- `trust-manager-provider` — Specifies the trust manager provider that should be used if the server is to communicate with the validation service over HTTPS, and the certificate presented by that validation service is not one that is automatically trusted by the JVM.

Configure YubiKey authentication for a user

There are three ways to register a YubiKey device for a particular user:

- `RegisterYubiKeyOTPDeviceExtendedRequest` – If using the Commercial Edition of the LDAP SDK for Java, this extended request is supported by the `com.unboundid.ldap.sdk.unboundidds.extensions.RegisterYubiKeyOTPDeviceExtendedRequest` class. If using a different API, consult the javadoc documentation for that class for the proper format for the value of the extended request and other information.
- `register-yubikey-otp-device` tool – Command line tool that uses the register YubiKey OTP device extended request provided with the server. To self register, use the `--authenticationID` argument to specify either "`dn:`" followed by the DN of your entry, or "`u:`" followed by username; use one of the `--userPassword`, `--userPasswordFile`, or `--promptForUserPassword` arguments to provide a password; and use the `--otp` argument to supply a one-time password generated by the YubiKey device that to register. To register a device on behalf of another user, the user's password is not needed, but the `--authenticationID` and `--otp` arguments must be provided, and you will also need to authenticate as a user with the `password-reset` privilege.
- LDAP modify operation or an add operation – If creating a user entry, directly store the desired public ID in the `ds-auth-yubikey-public-id` attribute in the target user's entry.

Retire a YubiKey device for a user

To retire a YubiKey device so that it can no longer be used for a specified user:

- Use the LDAP SDK for Java `DeregisterYubiKeyOTPDeviceExtendedRequest`.
- Use the `register-yubikey-otp-device` tool with the `--deregister` argument.
- Use a standard LDAP modify operation to remove the desired value(s) from the `ds-auth-yubikey-public-id` attribute in the target user's entry.

When removing support for a device, a one-time password generated by the YubiKey device can be supplied to deregister, or support for all devices that are registered for the specified user can be removed.

Configure certificate mappers

SASL EXTERNAL requires that a certificate mapper be configured in the server. The certificate mapper is used to identify the entry for the user to whom the certificate belongs. PingData servers support a number of certificate mapping options including:

- **Subject Equals DN** – Specifies that the subject of the certificate exactly match the distinguished name of the associated user entry. This option is not often practical as certificate subjects (`cn=jdoe,ou=Client Cert,o=Example Company,c=Austin,st=Texas,c=US`) are not typically in the same form as an entry (`cn=jdoe,ou=People,o=Example Company, or uid=jdoe,ou=People,dc=example,dc=com`).

- **Fingerprint** – Specifies that the user's entry contains an attribute (`ds-certficate-fingerprint` by default), with values that are the SHA-1, 256-bit SHA-2, or MD5 fingerprints of the certificate(s) that they can use to authenticate. This attribute must be indexed for equality.
- **Subject Attribute to User Attribute** – Used to build a search filter to find the appropriate user entry based on information contained in the certificate subject. For example, the default configuration expects the `cn` value from the certificate subject to match the `cn` value of the user's entry, and the `e` value from the certificate subject to match the `mail` value of the user's entry.
- **Subject DN to User Attribute** – Expects the user's entry to contain an attribute (`ds-certificate-subject-dn` by default), whose values are the subjects of the certificate(s) that they can use to authenticate. This multi-valued attribute can contain the subjects of multiple certificates. The attribute must be indexed for equality.

Configure the Subject Equals DN certificate mapper

The Subject Equals DN certificate mapper is the default mapping option for the SASL EXTERNAL mechanism. The mapper requires that the subject of the client certificate exactly match the distinguished name of the corresponding user entry. The mapper, however, is only practical if the certificate subject has the same format as the server's entries.

Perform the following to change the certificate mapper for the SASL EXTERNAL mechanism and configure the Subject Equals DN certificate mapper:

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
  --handler-name EXTERNAL \
  --set "certificate-mapper:Subject Equals DN"
```

Configure the Fingerprint certificate mapper

The Fingerprint mapper causes the server to compute an MD5, 256-bit SHA-2, or SHA-1 fingerprint of the certificate presented by the client and performs a search to find that fingerprint value in a user’s entry (`ds-certificate-fingerprint` by default). The `ds-certificate-fingerprint` attribute can be added to the user’s entry together with the `ds-certificate-user` auxiliary object class. For multiple certificates, the attribute can have separate values for each of the acceptable certificates. Make sure this attribute is indexed, if used.

The following example uses this certificate:

```
Alias name: client-cert
Creation date: Oct 29, 2011
Entry type: PrivateKeyEntry

Certificate chain length: 1 Certificate[1]:
Owner: CN=jdoe, OU=Client Cert, O=Example Company, L=Austin, ST=Texas, C=US
Issuer: EMAILADDRESS=whatever@example.com, CN=Cert Auth, OU=My Certificate Authority,
O=Example Company, L=Austin, ST=Texas, C=US
Serial number: e19cb2838441dbcd
Valid from: Thu Oct 29 13:07:10 CDT 2011 until: Fri Oct 29 13:07:10 CDT 2012
Certificate fingerprints:
   MD5: 40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   SHA1: 2A:89:71:06:1A:F5:DA:FF:51:7B:3D:2D:07:2E:33:BE:C6:5D:97:13
   Signature algorithm name: SHA1withRSA
   Version: 1
```

Perform the following steps to configure the Fingerprint certificate mapper:

1. Create an LDIF file to add the `ds-certificate-user object class` and `ds-certificate-fingerprint` attribute to the target user's entry.

   ```
   dn: uid=jdoe,ou=People,dc=example,dc=com
   changetype: modify
   add: objectClass
   objectClass: ds-certificate-user
   -
   add: ds-certificate-fingerprint
   ds-certificate-fingerprint:
   40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

2. Apply the change to the entry using `ldapmodify`:

   ```
   $ bin/ldapmodify --filename add-cert-attr.ldif
   ```

   ```
   dn: uid=jdoe,ou=People,dc=example,dc=com
   ds-certificate-
   fingerprint:40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

3. Check that the attribute was added to the entry using `ldapsearch`.

   ```
   $ bin/ldapsearch --baseDN dc=example,dc=com "(uid=jdoe)" \
     ds-certificate-fingerprint
     dn:uid=jdoe,ou=People,dc=example,dc=com
     ds-certificate-
   fingerprint:40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

4. Create an index for the `ds-certificate-fingerprint` attribute. If the server is configured with multiple data backends, the attribute should be indexed in each of those backends.

```
$ bin/dsconfig create-local-db-index --backend-name userRoot \
  --index-name ds-certificate-fingerprint \
  --set index-type:equality
```

5. Use the `rebuild-index` tool to cause an index to be generated for this attribute.

```
$ bin/rebuild-index --task --baseDN dc=example,dc=com \
  --index ds-certificate-fingerprint
```

```
[14:56:28] The console logging output is also available in
'/ds/PingDataMetricsServer/logs/tools/rebuild-index.log'
[14:56:29] Due to changes in the configuration, index
dc_example_dc_com_ds-certificate-fingerprint.equality is currently
operating in a degraded state and must be rebuilt before it can used
[14:56:29] Rebuild of index(es) ds-certificate-fingerprint started with
161 total records to process
[14:56:29] Rebuild complete. Processed 161 records in 0 seconds (average
rate 1125.9/sec)
```

6. Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
  --handler-name EXTERNAL \
  --set "certificate-mapper:Fingerprint Mapper"
```

Configure the Subject Attribute to User Attribute Certificate Mapper

The Subject Attribute to User Attribute certificate mapper maps common attributes from the subject of the client certificate to the user's entry. The generated search filter must match exactly one entry within the scope of the base distinguished name for the mapper. If no match is returned or if multiple machine entries are found, the mapping fails.

Given the subject of the client certificate:

```
Owner: CN=John Doe, OU=Client Cert, O=Example Company, L=Austin, ST=Texas, C=US
We want to match to the following user entry:
dn: uid=jdoe,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: jdoe
givenName: John
sn: Doe
cn: John Doe
mail: jdoe@example.com
```

Perform the following to change the certificate mapper for the SASL EXTERNAL mechanism and configure the Subject Attribute to User Attribute certificate mapper:

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
  --handler-name EXTERNAL \
  --set "certificate-mapper:Subject Attribute to User Attribute"
```

Configure the Subject DN to User Attribute certificate mapper

The Subject DN to User Attribute certificate mapper expects the user’s entry to contain an attribute (`ds-certificate-subject-dn` by default) whose values match the subjects of the certificates that the user can use to authenticate. The `ds-certificate-subject-dn` attribute can be added to the user’s entry together with the `ds-certificate-user` auxiliary object class. The attribute is multi-valued and can contain the subject distinguished names of multiple certificates. The certificate mapper must match exactly one entry, or the mapping will fail.

If using this attribute, add an equality index for this attribute in all data backends.

Perform the following steps to configure the Subject DN to User Attribute certificate mapper:

1. Create an LDIF file to add the `ds-certificate-user` object class and `ds-certificate-subject-dn` attribute to the target user’s entry.

```
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: ds-certificate-user
-
add: ds-certificate-subject-dn
ds-certificate-subject-dn:CN=John Doe,OU=Client Certificate,O=Example
Company,L=Austin,ST=Texas,C=US
```

2. Then, apply the change to the entry using `ldapmodify`:

```
$ bin/ldapmodify --filename add-cert-attr.ldif
```

3. Check that the attribute was added to the entry using `ldapsearch`.

```
$ bin/ldapsearch --baseDN dc=example,dc=com "(uid=jdoe)" \
  ds-certificate-subject-dn
  dn: uid=jdoe,ou=People,dc=example,dc=com
  ds-certificate-fingerprint:CN=jdoe, OU=Client Cert, O=Example Company,
  L=Austin, ST=Texas, C=US
```

4. Create an index to the `ds-certificate-subject-dn` attribute.

```
$ bin/dsconfig create-local-db-index --backend-name userRoot \
  --index-name ds-certificate-subject-dn \
  --set index-type:equality
```

5. Use the `rebuild-index` tool to ensure that the index is properly generated in all appropriate backends.

```
$ bin/rebuild-index --task --baseDN dc=example,dc=com \
  --index ds-certificate-subject-dn
```

```
[15:39:19] The console logging output is also available in
'/ds/PingDataMetricsServer/logs/ tools/rebuild-index.log'
[15:39:20] Due to changes in the configuration, index
dc_example_dc_com_ds-certificate-subject-dn.equality is currently
```

```
operating in a degraded state and must be rebuilt before it can used
[15:39:20] Rebuild of index(es) ds-certificate-subject-dn started with
161 total records to process
[15:39:20] Rebuild complete. Processed 161 records in 0 seconds (average
rate 2367.6/sec)
```

6. Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
  --handler-name EXTERNAL \
  --set "certificate-mapper:Subject DN to User Attribute"
```

Configuring pass-through authentication

Pass-through authentication (PTA) is a mechanism by which one server receives the bind request and can consult another server to authenticate the bind request. Implement this functionality by configuring a PTA plug-in that enables the server to accept simple password-based bind operations.

Perform the following steps to configure PTA:

1. Use `dsconfig` to define external servers to perform the authentication. The bind DN is set to `uid=pass-throughuser, dc=example,dc=com`, which is used to bind to the target LDAP server for simple authentication. The `verify-credentials-method` property ensures that a single set of connections for processing binds and all other types of operations is in place without changing the identity of the associated connection. Multiple external servers can be configured.

```
$ bin/dsconfig create-external-server \
  --server-name "ds-with-pw-1.example.com:389" \
  --type pingdatametricsserver \
  --set server-host-name:ds-with-pw-1.example.com \
  --set server-port:389 \
  --set "bind-dn:uid=pass-through-user,dc=example,dc=com" \
  --set authentication-method:simple \
  --set verify-credentials-method:retain-identity-control
```

2. Create an instance of the PTA plug-in that will use the external server(s). The server will first try to process a local bind as the target user (`try-local-bind:true`). The `try-local-bind:true` with `override-local-password:true` means that if the local bind fails, it will try sending the request to `ds-with-pw-1.example.com:389` or another server, if configured (`server-access-mode:round-robin`). If the bind succeeds against the remote server, the local entry is updated to store the password that was used (`update-local-password:true`). The number of connections to initially establish to the LDAP external server is set to 10. The maximum number of connections maintained to the LDAP external server is 10.

```
$ bin/dsconfig create-plugin \
  --plugin-name "Pass-Through Authentication" \
  --type pass-through-authentication \
  --set enabled:true \
  --set server:ds-with-pw-1.example.com:389 \
  --set server:ds-with-pw-2.example.com:389 \
  --set try-local-bind:true \
  --set override-local-password:true \
  --set update-local-password:true \
  --set server-access-mode:round-robin \
  --set initial-connections:10 \
  --set max-connections:10
```

Note

The `try-local-bind` property works with the `override-local-password` property. If `try-local-bind` is true and `override-local-password` is set to its default value of false, the server attempts a local bind first. If it fails because no password is set, it forwards the bind request to a remote server. If the password was set but still fails, the server will not send the request to the remote server.

If `try-local-bind` is true and `override-local-password` is true, a local bind is attempted. The server forwards the request to the remote server, if the local bind fails.

Prevent bind information leaks

For most operations, if a problem prevents the operation from completing successfully, the server attempts to return a detailed diagnostic message, appearing in the server's access log. However, for bind operations, returning a diagnostic message could be intercepted by an attacker. To avoid this, the server does not return diagnostic messages for a number of authentication failures. The information is included in access log messages in the `authFailureReason` element, so it is available to administrators, but not returned to the client.

If it is deemed that the value of providing this information to clients outweighs the risk of an attacker using the diagnostic information, the server can be configured to return those messages. This is controlled by the `return-bind-error-messages` property.

The following Global Configuration properties help prevent bind information leak:

- `return-bind-error-messages` – Indicates whether the server should include diagnostic messages in responses for unsuccessful bind operations. This feature has a value of `false` by default for a more secure configuration, but it can be changed to `true` if the benefit of providing these messages to clients is believed to outweigh their risk. Regardless of the setting, the reason for the authentication failure is indicated in the server access log.

- `bind-with-dn-requires-password` – Indicates whether the server should reject any simple bind request that contains a non-empty DN with an empty password. Although this is allowed by LDAP standards (as an anonymous simple bind), security problems can arise from poorly written clients that don't check whether an empty password is provided, and merely checks the bind operation result code. If this is enabled (default setting), the server rejects these type of bind requests. Simple bind requests with an empty DN and an empty password are still allowed, so this option should only be disabled if clients are allowed to perform legitimate anonymous binds that include a non-empty DN in the bind request.

Chapter 11: Monitoring, alerts, alarms, and notifications

The PingData servers support a flexible monitoring framework that enables administrators to detect unusual activity. Each server exposes its monitoring information under the `cn=monitor` entry, and provides interfaces through JMX, the Administrative Console, over LDAP, over SNMP, and through the PingDataMetrics Server.

PingData servers also provide delivery mechanisms for alarms, alerts, and notifications that can be sent to end users, operators, and directory administrators, such as account status notifications and administrative alerts using SMTP, JMX, SNMP or standard error logging.

Topics include:

Monitoring components

The PingData product family exposes its monitoring information under the `cn=monitor` entry for easy access to its information. Administrators can use various means to monitor the server's information including the Administrative Console, JConsole, LDAP commandline tools, JMX, through SNMP, and using thePingDataMetrics Server.

About the PingDataMetrics Server

The PingDataMetrics Server collects performance and event data from a set of PingData PingDirectory Server, PingDirectoryProxy Server, and/or PingDataSync Servers. A single PingDataMetrics Server instance can collect data from up to 50 servers. The PingDataMetrics Server normalizes and aggregates this data and makes it available to users through a RESTful API.

For more information, see the *PingData PingDataMetrics Server Administration Guide*.

PingDataMetrics Server Security

The PingDataMetrics Server can be secured by setting the `require-api-authentication` property of the Monitoring Configuration object using the `dsconfig` command-line tool.

The HTTPS Connection Handler can also be configured for accessing the API to encrypt traffic over the wire.

Monitoring using SNMP

The PingDirectory Server supports real-time monitoring using SNMP. The PingDirectory Server provides an embedded SNMPv3 subagent plugin that, when enabled, sets up the server as a managed device and exchanges monitoring information with a master agent based on the AgentX protocol.

MIBS

The PingDirectory Server provides SMIv2-compliant MIB definitions (RFC 2578, 2579, 2580) for distinct monitoring statistics. These MIB definitions are found in text files under `resource/mib` directory under the server root directory.

Each MIB provides managed object tables for each specific SNMP management information as follows:

- **LDAP Remote Server MIB** – Provides information related to the health and status of the LDAP servers to which the PingDirectoryProxy Server connects, and statistics about the operations invoked by the PingDirectoryProxy Server on those LDAP servers.
- **LDAP Statistics MIB** – Provides a collection of connection-oriented performance data that is based on a connection handler in the PingDirectory Server. A server typically contains only one connection handler and therefore supplies only one table entry.
- **Local DB Backend MIB** – Provides key metrics related to the state of the local database backends contained in the server.
- **Processing Time MIB** – Provides a collection of key performance data related to the processing time of operations broken down by several criteria but reported as a single aggregated data set.
- **Replication MIB** – Provides key metrics related to the current state of replication.
- **System Status MIB** – Provides a set of critical metrics for determining the status and health of the system in relation to its work load.

For information on the available monitoring statistics for each MIB available on the PingDirectory Server and the PingDirectoryProxy Server, see the text files provided in the `resource/mib` directory in the server installation.

The PingDirectory Server generates an extensive set of SNMP traps for event monitoring. The traps display the severity, description, name, object ID, and summary. For information about the available alert types for event monitoring, see the `resource/mib/UNBOUNDID-ALERT-MIB.txt file`.

Monitoring with JMX

The PingDirectory Server supports monitoring the JVM through a Java Management Extensions (JMX) management agent, which can be accessed using JConsole or a JMX client. The JMX interface provides JVM performance and resource utilization information for applications running Java. Generic metrics exposed by the JVM itself can be monitored, including memory pools, threads, loaded classes, and MBeans, as well as all the monitor information that the PingDirectory Server provides. JMX notifications for any administrative alerts that are generated within the server can also be received.

Monitoring using the LDAP SDK

Use the monitoring API to retrieve monitor entries. For example, retrieve all monitor entries published by the PingDirectory Server and print the information contained in each using the generic API for accessing monitor entry data as follows:

```
for (MonitorEntry e : MonitorManager.getMonitorEntries(connection))
  {
    System.out.println("Monitor Name: " + e.getMonitorName());
    System.out.println("Monitor Type: " + e.getMonitorDisplayName());
    System.out.println("Monitor Data:");
    for (MonitorAttribute a : e.getMonitorAttributes().values())
    {
      for (Object value : a.getValues())
      {
        System.out.println(" " + a.getDisplayName() + ": " + String.valueOf(value));
      }
    }
    System.out.println();
  }
```

For more information about the LDAP SDK and the methods in this example, see the *LDAP SDK* documentation.

Monitoring over LDAP

The PingData servers expose a majority of their information under the `cn=monitor` entry. Access these entries over LDAP using the `ldapsearch` tool.

```
$ bin/ldapsearch --hostname server1.example.com \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --baseDN "cn=monitor" "(objectclass=*)"
```

Profile server performance using the Stats Logger Plugin

Each server ships with a built-in Stats Logger Plugin (disabled by default) that is useful for profiling server performance for a given configuration. At a specified interval, the Stats logger writes server statistics to a log file in a comma-separated format (.csv), which can be read by spreadsheet applications. The logger has a negligible impact on server performance unless the log-interval property is set to a very small value (less than 1 second). The statistics that are logged and their verbosity can be configured with the `dsconfig` tool.

Working with administrative alert handlers

PingData servers provide mechanisms to send alert notifications to administrators when significant problems or events occur. Several alert handler implementations are available, including:

- **Error Log Alert Handler** – Sends administrative alerts to the configured server error logger(s).
- **Exec Alert Handler** – Executes a specified command on the local system if an administrative alert matching the criteria for this alert handler is generated by the PingDirectory Server. Information about the administrative alert is made available to the executed application as arguments provided by the command.
- **Groovy Scripted Alert Handler** – Provides alert handler implementations defined in a dynamically-loaded Groovy script that implements the `ScriptedAlertHandler` class defined in the Server SDK.
- **JMX Alert Handler** – Sends administrative alerts to clients using the Java Management Extensions (JMX) protocol. PingData uses JMX for monitoring entries and requires that the JMX connection handler be enabled.
- **SMTP Alert Handler** – Sends administrative alerts to clients by email using SMTP. The server requires that one or more SMTP servers be defined in the global configuration.
- **SNMP Alert Handler** – Sends administrative alerts to clients using the Simple Network Monitoring Protocol (SNMP). The server must have an SNMP agent capable of communicating via SNMP 2c.
- **SNMP Subagent Alert Handler** – Sends SNMP traps to a master agent in response to administrative alerts generated within the server.
- **Third Party Alert Handler** – Provides alert handler implementations created in third-party code using the Server SDK.

The Alerts backend

PingData servers store recently generated administrative alerts under the `cn=alerts` branch. The backend makes it possible to obtain alert information over LDAP for use with remote monitoring. The backend's primary job is to process search operations for alerts. It does not support add, modify, or modify DN operations of entries.

The alerts persist on disk in the `config/alerts.ldif` file so that they can survive server restarts. By default, the alerts remain on disk for seven days before being removed. However, administrators can configure the number of days for alert retention using the `dsconfig` tool. The administrative alerts of Warning level or worse that have occurred in the last 48 hours are viewable from the output of the `status` command-line tool and in the Administrative Console.

View information in the Alerts backend

Use `ldapsearch` to view the administrative alerts:

```
$ bin/ldapsearch --port 1389 --bindDN "cn=Directory Manager" \
  --bindPassword secret --baseDN cn=alerts "(objectclass=*)"
```

```
dn: cn=alerts
objectClass: top
objectClass: ds-alert-root
cn: alerts

dn: ds-alert-id=3d1857a2-e8cf-4e80-ac0e-ba933be59eca,cn=alerts
objectClass: top
objectClass: ds-admin-alert
ds-alert-id: 3d1857a2-e8cf-4e80-ac0e-ba933be59eca
ds-alert-type: server-started
ds-alert-severity: info
ds-alert-type-oid: 1.3.6.1.4.1.32473.2.11.33
ds-alert-time: 20110126041442.622Z
ds-alert-generator: com.unboundid.directory.server.core.metrics.server
ds-alert-message: The Data Metrics Server has started successfully
```

Modify the alert retention time

Use `dsconfig` to change the maximum time information about generated alerts retained in the alerts backend. After this time, the information is purged from the server. The minimum retention time is 0 milliseconds, which immediately purges the alert information.

```
$ bin/dsconfig set-backend-prop \
  --backend-name "alerts" \
  --set "alert-retention-time: 2 weeks"
```

View the property using `dsconfig`:

```
$ bin/dsconfig get-backend-prop \
  --backend-name "alerts" \
  --property alert-retention-time
```

```
Property : Value(s)
---------------------:---------
alert-retention-time : 2 w
```

Configure duplicate alert suppression

Use dsconfig to configure the maximum number of times an alert is generated within a particular time frame for the same condition. The duplicate-alert-time-limit property specifies the length of time that must pass before duplicate messages are sent over the administrative alert framework and the maximum number of messages should be sent.

```
$ bin/dsconfig set-global-configuration-prop \
  --set duplicate-alert-limit:2 \
  --set "duplicate-alert-time-limit:3 minutes"
```

System alarms and gauges

An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server unavailable' or ‘server unavailable’ (Indicator Gauge). Gauges generate alarms, when the gauge’s severity changes due to changes in the monitored value. Like alerts, alarms have severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a Condition property, and may have a Specific Problem or Resource property. If surfaced through SNMP, a Probable Cause property and Alarm Type property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

There are two alert types supported by the server - standard and alarm-specific. The server constantly monitors for conditions that may attention by administrators, such as low disk space. For this condition, the standard alert is low-disk-space-warning, and the alarm-specific alert is alarm-warning. The server can be configured to generate alarm-specific alerts instead of, or in addition to, standard alerts. By default, standard alerts are generated for conditions internally monitored by the server. However, gauges can only generate alarm-alerts.

The server installs a set of gauges that are specific to the product and that can be cloned or configured through the dsconfig tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

The PingData servers are compliant with the International Telecommunication Union CCITT Recommendation X.733 (1992) standard for generating and clearing alarms. If configured, entering or exiting an alarm state can result in one or more alerts. An alarm state is exited when the condition no longer applies. An `alarm_cleared` alert type is generated by the system when an alarm's severity changes from a non-normal severity to any other severity. An `alarm_cleared` alert will correlate to a previous alarm when the Condition property is the same. The Alarm Manager, which governs the actions performed when an alarm state is entered, is configurable through the `dsconfig` tool and Administrative Console.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool. As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

Test alerts and alarms

After alarms and alert handlers are configured, verify that the server takes the appropriate action when an alarm state changes by manually increasing the severity of a gauge. Alarms and alerts can be verified with the `status` tool.

1. Configure a gauge with `dsconfig` and set the override-severity property to critical. The following example uses the CPU Usage (Percent) gauge.

```
$ dsconfig set-gauge-prop \
  --gauge-name "CPU Usage (Percent)" \
  --set override-severity:critical
```

2. Run the `status` tool to verify that an alarm was generated with corresponding alerts. The `status` tool provides a summary of the server’s current state with key metrics and a list of recent alerts and alarms. The sample output has been shortened to show just the alarms and alerts information.

```
$ bin/status
```

```
                        --- Administrative Alerts ---
Severity : Time          : Message
---------:---------------:------------------------------------------------------
Info     : 11/Aug/2014   : A configuration change has been made in the
```

```
         : 15:48:46-0500 : PingDirectory Server:
         :               : [11/Aug/2014:15:48:46.054 -0500]
         :               : conn=17 op=73 dn='cn=Directory Manager,cn=Root
         :               : DNs,cn=config' authtype=[Simple] from=127.0.0.1
         :               : to=127.0.0.1 command='dsconfig set-gauge-prop
         :               : --gauge-name 'Cleaner Backlog (Number Of Files)'
         :               : --set warning-value:-1'
Info     : 11/Aug/2014   : A configuration change has been made in the
         : 15:47:32-0500 : PingDirectory Server: [11/Aug/2014:15:47:32.547 -0500]
         :               : conn=4 op=196 dn='cn=Directory Manager,cn=Root
         :               : DNs,cn=config' authtype=[Simple] from=127.0.0.1
         :               : to=127.0.0.1 command='dsconfig set-gauge-prop
         :               : --gauge-name 'Cleaner Backlog (Number Of Files)'
         :               : --set warning-value:0'
Error    : 11/Aug/2014   : Alarm [CPU Usage (Percent). Gauge CPU Usage
         : 15:41:00-0500 : for Host System Recent CPU and Memory has
         :               : a current value of '18.583333333333332'.
         :               : The severity is currently OVERRIDDEN in the
         :               : Gauge's configuration to 'CRITICAL'.
         :               : The actual severity is: The severity is
         :               : currently 'NORMAL', having assumed this severity
         :               : Mon Aug 11 15:41:00 CDT 2014. If CPU use is high,
         :               : check the server's current workload and make any
         :               : needed adjustments. Reducing the load on the system
         :               : will lead to better response times.
         :               : Resource='Host System Recent CPU and Memory']
         :               : raised with critical severity
Shown are alerts of severity [Info,Warning,Error,Fatal] from the past 48 hours
Use the --maxAlerts and/or --alertSeverity options to filter this list
```

```
                            --- Alarms ---
Severity : Severity   : Condition : Resource    : Details
         : Start Time :           :             :
---------:------------:-----------:-------------:---------------------------
Critical : 11/Aug/2014: CPU Usage : Host System : Gauge CPU Usage (Percent) for
         : 15:41:00   : (Percent) :             : Host System
         :  -0500     :           :             : has a current value of
         :            :           :             : '18.785714285714285'.
         :            :           :             : The severity is currently
         :            :           :             : 'CRITICAL', having assumed
         :            :           :             : this severity Mon Aug 11
         :            :           :             : 15:49:00 CDT 2014. If CPU use
         :            :           :             : is high, check the server's
         :            :           :             : current workload and make any
         :            :           :             : needed adjustments. Reducing
         :            :           :             : the load on the system will
         :            :           :             : lead to better response times

Shown are alarms of severity [Warning,Minor,Major,Critical]
Use the --alarmSeverity option to filter this list
```

Working with account status notifications

PingData servers support notification handlers that can be used to notify users and/or administrators of significant changes related to password policy state for user entries. The following two notification handlers are available:

- **Error Log Account Status Notification Handler** – Enabled by default. The handlers send alerts to the error log when an account event occurs.
- **SMTP Account Status Notification Handler** – Sends notifications to designated email addresses, when enabled. The SMTP Handler can be enabled with the `dsconfig` command.

Account status notification types

The handlers send alerts when one of the account status events described in the following table occurs during password policy processing.

Account Status Notification Types

Account Status Notification Types	Description
`account-disabled`	Generates a notification when a user account is disabled by an administrator.
`account-enabled`	Generates a notification when a user account is enabled by an administrator.
`account-expired`	Generates a notification when a user authentication attempt fails because the account has expired.
`account-idle-locked`	Generates a notification when a user authentication attempt fails because the account has been locked after idling for too long.
`account-permanently-locked`	Generates a notification when a user account is permanently locked (requiring administrative action to unlock the account) after too many failed attempts.
`account-reset-locked`	Generates a notification when an authentication attempt fails because the user account is locked due to a failure to change the password within the required interval set by the administrator.
`account-temporarily-locked`	Generates a notification whenever a user account is temporarily locked after too many failed attempts.
`account-unlocked`	Generates a notification whenever a user account is unlocked by an administrator.
`password-changed`	Generates a notification whenever a user changes his or her own password.
`password-expired`	Generates a notification whenever a user authentication fails because the password has expired.
`password-expiring`	Generates a notification the first time that a password expiration warning is encountered for a user password.
`password-reset`	Generates a notification whenever a user's password is reset by an administrator.

Chapter 12: Logging security

PingData servers provide logging capabilities to parse and analyze any situational event or problem that may occur. This chapter summarizes the logging features available on the servers.

Topics include:

Configure log rotation and retention policies

Because disks do not have unlimited space, file-based loggers provide options for log file rotation and retention. Log file rotation is the process by which the active log file is closed and renamed, and a new file is created in its place. For example, the default access logger uses a file named access. When rotation occurs, the current access file is renamed to include a timestamp such as `access.20110102030405Z`, and a new empty access file is started. The primary purpose of log file rotation is to ensure that no individual log file grows too large.

There are a few different kinds of log rotation policies, including:

- **Size limit rotation policy** – Starts rotation when the log file reaches a given size.
- **Time limit rotation policy** – Starts rotation based on the length of time since the last rotation.
- **Fixed time rotation policy** – Starts rotation at specified times in the day.
- **Never rotate policy** – Prevents log rotation from occurring.

Each file-based logger must have at least one rotation policy. If there are multiple policies, any of them can trigger a rotation. For example, the default access logger is configured with two rotation policies: one that will trigger a rotation if the log file reaches 100MB in size, and another that will trigger a rotation if it's been 24 hours since the previous rotation. Therefore, there will be one rotation per day, or more if more than 100MB is written in the course of a day.

Log retention policies are used to determine when rotated log files should be removed from the system (with older files deleted before newer files). Available types of log retention policies include:

- **File count retention policy** – Deletes rotated log files as necessary to ensure that the number of rotated files does not exceed a given count.
- **Size limit retention policy** – Deletes rotated log files as necessary to ensure that the total size of rotated files (for a particular logger) does not exceed a given threshold.
- **Free disk space retention policy** – Deletes rotated log files if the amount of remaining usable disk space on the volume holding those files drops below a given threshold.
- **Never delete retention policy** – Causes a log file deletion to never be triggered. Each file-based logger must have at least one retention policy.

Log signing

Logs can be cryptographically signed to ensure that they have not been modified. For example, financial institutions require audit logs for all transactions to check for correctness. Tamper-proof files are needed to ensure that these transactions can be properly validated and that they have not been modified by any third-party. Use the `dsconfig` tool to enable the `sign-log` property on a Log Publisher to turn on cryptographic signing.

When enabling signing for a logger that already exists and was enabled without signing, the first log file will not be completely verifiable because it still contains unsigned content. Only log files whose entire content was written with signing enabled is considered completely valid. For the same reason, if a log file is still open for writing, then signature validation will not indicate that the log is completely valid because the log will not include the necessary "end signed content" indicator at the end of the file.

To validate log file signatures, use the `validate-file-signature` tool provided in the `bin` directory of the server (or the `bat` directory for Windows systems).

Once this property is enabled, disable and then re-enable the Log Publisher for the changes to take effect.

Configure access logging

Access loggers can be used to record information whenever a connection is established and/or closed, when the server receives a request from a client, and/or when the server sends a response to a client. Access loggers are a useful way of understanding the processing that the server has actually performed.

By default, the server configuration includes these access loggers:

- **File-Based Access Logger** – Logs information about all operations processed by the server (one message per operation combining both request and response details), as well as connects and disconnects. This logger is enabled by default and writes to the `logs/access` file.
- **Failed Operations Access Logger** – Logs information about operations that did not complete successfully. It does this using a result criteria object configured to only match operations with a result code other than success, compare true, compare false, referral,

SASL bind in progress, and no operation. This logger is enabled by default and writes to the `logs/failed-ops` file.

- **Expensive Operations Access Logger** – Logs operations that took at least 1000 milliseconds to complete. This is useful to determine if a client is issuing requests the server isn't optimally configured to handle, if searches are returning an unusually large number of entries, or if the server is under exceptionally heavy load. This logger is disabled by default. When enabled it writes to the `logs/expensive-ops` file.
- **File-Based Audit Logger** – Writes information about successful add, delete, modify, and modify DN operations. The content of the changes are represented in LDIF form, which makes it possible to determine exactly what change was requested by the client. Alternately, it can be configured to log changes in reversible form, which enables a change to be undone if it was made in error. This logger is disabled by default. If enabled, it writes to the `logs/audit` file.
- **Successful Searches with no Entries Found** – Writes information about successful search requests including requestor information, search information, and request and result criteria. This logger is disabled by default. If enabled, it writes to the `logs/searches-returning-no-entries` file.

The Server SDK can be used to create additional access loggers. All types of access loggers provide a number of common options, including:

- `suppress-replication-operations` – Indicates whether the logger should be used to record information about operations initiated by replication in addition to those requested by external clients.
- `log-connects` – Indicates whether the logger should record information about new connections established to the server.
- `log-disconnects` – Indicates whether the logger should record information about existing connections that are closed.
- `log-client-certificates` – Indicates whether the logger should record information about certificates that clients present to the server during SSL or StartTLS negotiation.
- `log-requests` – Indicates whether the logger should record information about operation requests sent to the server.
- `log-forwards` – Indicates whether the logger should record information about requests forwarded on to one or more backend servers. This is primarily applicable to the PingDirectoryProxy Server.
- `log-forward-failures` – Indicates whether the logger should record information about failures encountered while attempting to process an operation in a backend server. This is primarily applicable to the PingDirectoryProxy Server.

- `log-results` – Indicates whether the logger should record information about the outcome of operation processing.
- `log-search-entries` – Indicates whether the logger should record information about each search result entry returned to clients.
- `log-search-references` – Indicates whether the logger should record information about each search result reference returned to clients.
- `log-intermediate-responses` – Indicates whether the logger should record information about each intermediate response returned to clients.

Several loggers other configuration properties to further customize their behavior, including:

- `include-request-details-in-result-messages` – Indicates whether the server should include all of the content that it would provide in request messages in the log message for the result of that operation. When this is combined with setting `log-requests` to false, this makes it possible to write only a single log message per operation rather than separate messages for the request and the result. This also makes it easier to interpret log messages, because information between request and result messages doesn't need to be correlated.
- `include-request-details-in-search-entry-messages` – Same effect as `include-request-details-in-result-messages`, except that it applies to log messages generated for search result entries returned to clients.
- `include-request-details-in-search-reference-messages` – Same effect as `include-request-details-in-result-messages`, except that it applies to log messages generated for search result references returned to clients.
- `include-request-details-in-intermediate-response-messages` – Same effect as `include-request-details-in-result-messages`, except that it applies to log messages generated for intermediate response messages returned to clients.
- `include-extended-search-request-details` – Indicates whether log messages for search requests should include additional information about the request, including the requested size limit, time limit, alias dereferencing behavior, and types only flag.
- `include-add-attribute-names` – Indicates whether log messages for add requests (and/or add result messages if request details should be included in result messages) should include a field with the names of the attributes included in the add request.
- `include-modify-attribute-names` – Indicates whether log messages for modify requests (and/or modify result messages if request details should be included in result messages) should include a field with the names of the attributes targeted by the modify request.

- `include-search-entry-attribute-names` – Indicates whether search result entry messages should include a field with the names of the attributes in the entry returned to the client.
- `include-product-name` – Indicates whether log messages should include the name of the product that logged the message. This is helpful for logging messages from multiple products, which may be combined.
- `include-instance-name` – Indicates whether log messages should include the name of the server instance that logged the message. The instance name can be specified in the global configuration, but the server can generate its own instance name (which will generally contain the address and port on which it is listening for client connections). This can be helpful for cases in which log messages from multiple instances are combined.
- `include-startup-id` – Indicates whether log messages should include a compact unique identifier that is generated at the time the server is started. This can help differentiate log messages from the same instance across server restarts. When the server is restarted, connection IDs are reset to zero, so without a startup ID it may be difficult to distinguish between operations with the same connection ID and operation ID before and after the restart.
- `include-requester-ip-address` – Indicates whether log messages should include the IP address of the client from which the request was received. The client address will be included in the message logged when a connection is established, but including the IP address in request and result messages can avoid the need to locate the connect message to determine the address of a given client.
- `include-requester-dn` – Indicates whether log messages should include the DN of the user authenticated on the connection on which the request was received. The DN of the authenticated user is included in bind result messages, but it can be useful to include the requester DN in other kinds of log messages as well.
- `include-request-controls` – Indicates whether log messages should include the OIDs of any controls included in the request received from the client.
- `include-response-controls` – Indicates whether log messages should include the OIDs of any controls included in responses returned to the client.
- `include-replication-change-id` – Indicates whether log messages for write operations should include the replication change ID for an operation. This can be used for debugging and correlating a replicated change as it is processed across multiple servers.
- `max-string-length` – Specifies the maximum length of any string allowed for a field included in an access log message. This ensures that long log elements are truncated (with an indication of the number of bytes removed) to save space.

- `timestamp-precision` – Indicates whether to log timestamps with an accuracy of seconds or milliseconds. Although log message timestamps have traditionally only used second-level accuracy, when servers are capable of processing hundreds of thousands of operations per second per instance, timestamp precision can be useful.
- `compression-mechanism` – Indicates whether the contents of the log file should be compressed. A compressed log file consumes less space, which makes it possible to store more data. This setting cannot be changed after a logger has been created. To use compressed logging, create a new logger and enable compression.

Configure filtered logging

Servers under heavy load can easily generate hundreds of megabytes or more of log content every minute. While it is useful to have a full log of all operations processed by the server, the sheer volume of content (and the frequency with which files may be rotated or removed) can make it difficult to debug certain problems in real time. Further, storage space constraints may make it difficult to archive the entire history of operations.

PingData servers provide a criteria subsystem that make it easy to filter log contents. When this is combined with the server's ability to have any number of active access loggers, this makes it possible to have loggers dedicated to a particular purpose.

Many access loggers (including those that don't log to files) support filtering. The kinds of messages to include can be customized. For example, to create an access log with only operations requested by root users, use the `"Requests by Root Users"` connection criteria with a change like:

```
$ bin/dsconfig create-log-publisher \
  --publisher-name "Operations by Root Users" \
  --type file-based-access \
  --set enabled:true \
  --set "connection-criteria:Requests by Root Users" \
  --set log-file:logs/root-operations \
  --set include-requester-ip-address:true \
  --set include-requester-dn:true \
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --set "rotation-policy:Size Limit Rotation Policy" \
  --set "retention-policy:File Count Retention Policy" \
  --set "retention-policy:Size Limit Retention Policy"
```

A similar process can be used to log operations from a particular client (based on its address). In that case, choose a different connection criteria. For example, the following criteria can be used to match any request from client with IP address "1.2.3.4":

```
$ bin/dsconfig create-connection-criteria \
  --criteria-name "Clients from IP 1.2.3.4" \
  --type simple \
  --set included-client-address:1.2.3.4
```

To create an access logger that records every time the server returns a search result entry containing the `userPassword` attribute, create a search result entry criteria object that will match those entries, and then create a logger to use that criteria and configured to log only search result entry messages, such as:

```
$ bin/dsconfig create-search-entry-criteria \
  --criteria-name "Search Entries Containing Passwords" \
  --type simple \
  --set "any-included-entry-filter:(userPassword=*)"

$ bin/dsconfig create-log-publisher \
  --publisher-name "Password Retrieval" \
  --type file-based-access \
  --set enabled:true \
  --set log-client-certificates:false \
  --set log-results:false \
  --set log-search-entries:true \
  --set "search-entry-criteria:Search Entries Containing Passwords" \
  --set include-request-details-in-search-entry-messages:true \
  --set include-search-entry-attribute-names:true \
  --set include-requester-ip-address:true \
  --set include-requester-dn:true \
  --set log-file:logs/password-retrieval \
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --set "rotation-policy:Size Limit Rotation Policy" \
  --set "retention-policy:File Count Retention Policy" \
  --set "retention-policy:Size Limit Retention Policy
```

Configure change logging

The PingDirectory Server provides an audit log (which is implemented as a specialized access log) that records information about changes processed in the server using LDIF representations. The PingDirectory Server also provides support for an LDAP changelog, which makes this information available to LDAP clients in a form that can be consumed using APIs such as the LDAP SDK for Java. This information can be used to help synchronize changes between multiple systems, and it can also provide additional information about entries that have been updated but not included in the audit log.

The LDAP changelog is implemented as a special backend in the PingDirectory Server. The server configuration includes a changelog backend, which is disabled by default. The configuration object provides a number of properties that can be used to customize its behavior, including:

- `changelog-maximum-age` – Specifies the maximum length of time for which the changelog should hold records. The changelog automatically purges records older than this to ensure that the database does not grow too large. By default, changelog records are kept for two days.
- `changelog-include-attribute` – Restricts the set of changelog entries created for add and modify operations. If one or more include attributes are defined, changelog entries are only created for add operations, if the entry to add contains one or more of the specified attributes. Changelog entries are created for modify operations if one or more of those attributes was updated by the change. Only those attributes are listed in the `changes` attribute of the changelog entry. This setting does not impact modify or modify DN operations.
- `changelog-exclude-attribute` – Restricts the set of changelog entries created for add and modify operations. It is similar to the `changelog-include-attribute` property, except that it excludes the named attributes.
- `changelog-deleted-entry-include-attribute` – Indicates that changelog entries should contain the values of the specified attributes from entries that have been deleted. If no include or exclude attributes are specified, then no deleted entry attribute information is included.
- `changelog-deleted-entry-exclude-attribute` – Indicates that changelog entries should contain the values of all except the specified attributes from entries that have been deleted. If no include or exclude attributes are specified, no deleted entry attribute information is included.
- `changelog-include-key-atttribute` – Indicates that changelog entries should include the values of the specified attributes at the time of the update, regardless of whether those attributes were altered by the operation. For add, modify, and modify DN operations, this reflects the values of those attributes after the operation completes. For delete operations, this reflects the values of those attributes just before the entry was removed.
- `changelog-max-before-after-values` – Indicates that changelog entries should include the values of attributes updated by the operation, both before and after the operation is processed. This applies to both modify and modify DN operations. This option specifies the maximum number of values to report, which prevents including too many entries for bulk operations.
- `index-include-attribute` – Indicates that the changelog should maintain indexes for each of the specified attributes. This tracks changelog records in which the specified

attribute was included in the change that was processed. This can improve performance with the get changelog batch extended operation when change filtering is requested.

- `index-exclude-attribute` – Indicates that the changelog should maintain indexes for all attributes except those specified. This cannot be used with the `index-include-attribute` property.
- `use-reversible-form` – Indicates whether changelog entries for modify operations should record information about the change that enable it to be reverted. If enabled, delete changelog records include all deleted entry attributes.
- `include-virtual-attributes` – Indicates whether to include information about virtual attributes held in the entry at the time the change was made. Values for this property include:
    - `add-attributes` – Include information about virtual attributes as they would appear in the resulting entry after the add completed.
    - `before-and-after-values` – Include information about virtually-generated values that would be included in the entry before and after the change was applied, for modify and modify DN operations.
    - `deleted-entry-attributes` – Include virtual attribute values for the entry at the time it was removed).
    - `key-attribute-values` – Include virtual attribute values for key attributes in the entry).
- `apply-access-controls-to-changelog-entry-contents` – Indicates whether the server should apply access control restrictions to information contained in changelog entries before they are returned to clients. If true, this removes references to any attributes that the requester does not have permission to see from the changelog entries, before returning them to the client. This can be useful if changelog entries are accessible to non-administrators.
- `report-excluded-changelog-attributes`– Indicates whether changelog entries returned to the client should include information about any attributes that were removed as a result of access control processing. Values include:
    - `none` – Include no information about excluded attributes.
    - `attribute-counts` – Include the number of user and operational attributes that were excluded.
    - `attribute-names` – Include the names of the user and operational attributes that were excluded.

The LDAP SDK for Java supports parsing the information contained in changelog entries. The `com.unboundid.ldap.sdk.ChangeLogEntry` class interacts with changelog entries using the specification in the *draft-good-ldap-changelog IETF draft*, while the `com.unboundid.ldap.sdk.unboundidds.UnboundIDChangeLogEntry` class provides enhanced support for changelog entries in the PingDirectory Server, including key attributes, before and after values, and virtual attributes.

Configure error logging

Error loggers publish information about warnings, errors, and significant events encountered during processing. In addition to Server SDK support for creating custom error loggers, servers provide error loggers that can write messages to local files, a relational database (JDBC), or to a syslog server.

All error loggers provide support for the following configuration properties:

- `default-severity` – Specifies the log severities for messages that should be published by the error logger for all categories for which no `override-severity` is defined. Values include `fatal-error`, `severe-error`, `mild-error`, `severe-warning`, `mild-warning`, `notice`, `info`, `debug`, `all`, and `none`. Severities are not inherently hierarchical. Specify all severities for messages that should be included.
- `override-severity` – Indicates that log messages with a given category should use a set of severities that differ from those specified by the `default-severity` property. Values have a format of `category=severity-list`, where `category` is the name of a log message category (such as access-control, admin, or backend), and `severity-list` is a comma-separated list of the severities that should be used for that category. For example, a value of `third-party:fatal-error,severe-error,mild-error` indicates that all errors from third-party components should be logged.

Any number of error loggers can be configured. Logging can be enabled for multiple targets (log to both local files and to a remote database), and for short-term debugging purposes. To diagnose a problem, create a temporary error logger with a broader range of severities, without polluting the primary error log with a greater volume of less important content.

Configure debug logging

The debug logging subsystem can access detailed information about internal processing within a server. This content is useful for developers with access to the underlying source code. However, if the server is running with one or more custom extensions written with the Server SDK, then the debugging framework may be useful for diagnosing problems within that code.

By default, debug logging is disabled. Enabling debug logging for a long period of time may degrade performance due to the volume of debug code. It is recommended that debug logging remain disabled unless it is needed to solve a particular problem.

Note

Unlike other loggers, the server only provides the ability to record debug information to local files. There is no support for debugging to targets such as syslog or relational databases, nor is it possible to implement custom debug loggers in the Server SDK.

The file-based debug logger includes the following configuration properties:

- `default-debug-level` – Specifies the level of debug messages to be published. Levels are hierarchical, with the following values from least verbose to most verbose: `disabled`, `error`, `warning`, `info`, `verbose`, and `all`.
- `default-debug-category` – Specifies the categories for debug messages to be published. By default, messages from all categories are eligible for publishing. Categories include:
  - `caught` – For exceptions caught within the server.
  - `constructor` – For new object creation.
  - `data` – For data read or written.
  - `database-access` – For reads from and writes to a database.
  - `enter` – For method entry.
  - `exit` – For method return.
  - `message` – For general-purpose debugging.
  - `protocol` – For parsed communication with clients.
  - `thrown` – For exceptions thrown within the server.
- `default-omit-method-entry-arguments` – Indicates whether debug messages for constructor and method invocation should exclude information about the arguments provided.
- `default-omit-method-return-value` – Indicates whether debug messages for a method return should exclude the return value for that method.
- `default-include-throwable-cause` – Indicates whether debug messages for exceptions and errors should include information about exceptions caught that triggered the exception.
- `default-throwable-stack-frames` – Specifies the number of stack frames that should be included in debug messages for exceptions and errors.

The debug level and category options offer only a coarse level of control over what is published. The server also offers a debug target mechanism that provides fine-grained control, down to the package, class, or even method from which the debug messages are generated. The debug scope controls the code locations to which the debug target applies, and may be a fully-qualified class or package name or a fully-qualified class name followed by an octothorpe (#) and the name of a method within that class (such as "`com.unboundid.directory.server.core.DirectoryServer#startUp`" covers only debug messages generated from the `startUp` method in the `DirectoryServer` class).

Each debug target has its own level and category configuration, and those settings override the settings of the associated debug logger for messages matching that scope. For example, the "Server SDK Extension Debug Logger" is configured so that it will not generate any debug messages, but has a debug target that matches all messages generated from Server SDK extensions.

Note

Effective use of debug logging requires specific knowledge of the server source code. Unless debugging custom extensions written with the Server SDK, debug logging be used with the assistance of Ping Identity support.

Configure PingDataSync Server logging

The PingDataSync Server provides two loggers used to keep track of the synchronization operations. The first of these is the sync logger, a file-based sync log publisher that provides a general record of all synchronization activity for the following events:

- `change-detected` – Provides general information about a change detected in a Sync Source.
- `change-detected-detailed` – Provides detailed information about a change detected in a Sync Source.
- `change-applied` – Provides general information about a change applied to a Sync Destination.
- `change-applied-detailed` – Provides detailed information about a change applied to a Sync Destination.
- `change-failed` – Provides general information about a failure encountered while attempting to apply a change to a Sync Destination that will not be re-tried by the PingDataSync Server.
- `change-failed-detailed` – Provides detailed information about a failure encountered while attempting to apply a change to a Sync Destination that will not be re-tried by the PingDataSync Server.

- `intermediate-failure` – Provides information about a failure encountered while attempting to apply a change to a Sync Destination, but that will be re-tried by the PingDataSync Server.
- `synchronizing-out-of-date-change` – Indicates that the server synchronized a stale change that no longer reflects the current state of the Sync Source and may be updated by a later change that has already been applied. By default, the PingDataSync Server does not synchronize these changes.
- `no-change-needed` – Indicates that the server did not synchronize a change made in a Sync Source because the Sync Destination already had that change applied.
- `dropped-out-of-scope` – Indicates that a change detected in a Sync Source will not be applied to a destination because it is out of the scope of any Sync Class.
- `dropped-op-type-not-synchronized` – Indicates that a change detected in a Sync Source will not be applied to a destination because its change type is not one that should be synchronized.
- `entry-mapping-details` – Provides detailed information about any attribute and/or DN mapping applied to an entry in the course of preparing it to be applied to a Sync Destination.
- `plugin` – Provides a general message generated by a synchronization plugin.
- `plugin-error` – Provides information about an error encountered during processing within a synchronization plugin.
- `aborted-by-plugin` – Indicates that a synchronization plugin has aborted processing for a change.

The PingDataSync Server also provides a "Sync Failed Ops Log Publisher" logger that records information about failures encountered during synchronization processing. This primarily contains the DN of the source entry (or an entry constructed from data in the Sync Source), and may include additional information about the problem encountered.

Options for centralized logging

Servers are configured so that all logging is written to files on the local filesystem. In some environments, it may be convenient to have content from multiple servers appear in the same place for easier analysis. Centralized logging can be accomplished with one of the following options:

- **File-based logging to a network filesystem** – Each server instance can be configured to use a separate directory, or can be configured to use a different filename in the same directory. In either case, each instance maintains its own separate set of files, but those files are in the same location for easier analysis.

- **Logging to a relational database through JDBC** – All servers can be configured to log to separate databases, separate tables in the same database, or the same table in the same database.
- **Logging to a syslog server** – The server does not provide any native support for a secure syslog mechanism. If this option is used, each instance should be configured to log to a local daemon configured to act as a secure syslog relay.
- **Custom Logging using the Server SDK** – Use the Server SDK to create custom loggers to send messages to a centralized system.

If not using centralized logging, or if log files from separate instances can be mixed, configure those loggers so that the product name and instance name are included. This ensures that each message can be identified. The startup ID field can also be included, so that messages coming from the same server instance, with the same connection and operation ID values, can still be distinguished.

If centralized logging is enabled, local logging should also be enabled. If a problem occurs with the centralized system, that content is still recorded in local files.

Parse and Analyze log messages

Log messages generated by PingData servers are intended to be easy to read and understand, and easy to parse by tools for more automated analysis. The LDAP SDK for Java includes APIs (in the `com.unboundid.ldap.sdk.unboundidds.logs` package) for parsing access and error log messages generated by the PingDirectory Server, PingDirectoryProxy Server, and PingDataSync Server. In addition, because audit log records are in LDIF form, the LDAP SDK's LDIF support (in the `com.unboundid.ldif` package) can be used to consume those messages.

Note

Because of the nature of messages written to the sync logger or the failed ops sync logger, there are currently no APIs capable of parsing their content.

The `summarize-access-log` tool, which is provided with the PingDirectory Server and PingDirectoryProxy Servers, can be used to parse log content and identify a number of interesting elements, including:

- The total number of operations processed (overall and per operation type), the percentage of the total each operation type constitutes, and the average rate per second for those operations.

- The average duration for operations processed (overall and per operation type), in milliseconds with microsecond accuracy. Processing times are broken out into a histogram with buckets below 1ms, 1-2ms, 2-3ms, 3-5ms, 5-10ms, 10-20ms, 20-30ms, 30-50ms, 50-100ms, 100-1000ms, and over 1000ms.
- The most popular result codes for each type of operation.
- The number of unindexed search attempts, as well as the numbers of successful and failed unindexed searches.
- The most common search result entry counts.
- The most common filters used in non-base search requests. These filters are represented in generic form, like "`(uid=?)`" for any equality filter targeting the `uid` attribute with any value.

The source code for the `summarize-access-log` tool is provided as an example in the LDAP SDK for Java and can be used as the starting point for writing a tool.

Chapter 13: Network security

Client-server communication is one of the most critical points in securing a directory environment. This chapter addresses this issue.

Topics include:

SSL and StartTLS

The most popular way of securing network communication is through the use of SSL. The protection that StartTLS offers is the same as SSL, except the time in which the negotiation is performed.

When an SSL-based connection is established, the client and server immediately begin the negotiation process so that there is never any unencrypted communication. With StartTLS, the client establishes an initially-insecure connection, and may optionally issue unencrypted requests over that connection (such as a request to retrieve the server's root DSE to determine StartTLS extended operation support). When the client wishes to convert the insecure connection to a secure one, it sends a StartTLS extended request to the server. If the server returns a response of "success," the negotiation will start the same way as an SSL-based connection.

Note

Once a connection has been secured using StartTLS, it will generally remain encrypted for the duration of that connection. While it is technically possible to end an SSL session without terminating the connection, many servers (including the PingData server products) do not support this, because there is no standard way for either the client or the server to indicate that they want to end the secure communication phase but continue with unencrypted communication.

Configure SSL

If SSL was not configured during installation, it can be enabled with the following steps. This procedure assumes that a certificate is available in a JKS-formatted keystore.

Perform the following steps to configure SSL:

1. Change to the server root directory.

   ```
   $ cd /ds/PingData<server>
   ```

2. Create a text file containing the password for the certificate keystore. The file permissions (or filesystem ACLs) should be configured so that the file is only readable by the server user account.

   ```
   $ echo ’changeit’ > config/keystore.pin
   $ chmod 0400 config/keystore.pin
   ```

3. Run the `dsconfig` command in interactive mode (`bin/dsconfig`).
4. Enter the connection parameters when prompted.
5. On the main menu, switch to the Advanced menu.
6. Enter the option for the **Key Manager Provider**.
7. On the Key Manager Provider menu, select the option to view and edit an existing key manager.
8. On the Key Manager Provider menu, enter the option for **JKS**.
9. Make any necessary changes to the JKS key manager provider for the keystore. The provider must be enabled, and the locations of the `key-store-file` and `key-store-pin-file` must be set.

```
>>>> Configure the properties of the File Based Key Manager Provider

Property Value(s)
-----------------------------------------
1) description -
2) enabled true
3) key-store-file config/keystore
4) key-store-type              JKS
5) key-store-pin               -
6) key-store-pin-file config/keystore.pin
7) private-key-pin             -
8) private-key-pin-file        -
```

10. Type **f** to save and apply the changes.
11. Return to the main menu, and enter the option for the **Trust Manager Provider**.
12. On the Trust Manager Provider menu, enter the option to view and edit an existing trust manager provider.
13. On the Trust Manager Provider menu, enter the option for **JKS**.
14. Make sure that the JKS trust manager provider is enabled and that the `trust-store-file` property has a value that reflects the path to the truststore file.
15. Type **f** to save and apply the changes.
16. Return to the main menu, and enter the option for the **Connection Handler** option.
17. On the Connection Handler menu, enter the option to view and edit and existing connection handler.
18. On the Connection Handler menu, enter the option for **LDAPS Connection Handler**.
19. On the LDAP Connection Handler menu, make sure that the handler is enabled, the `listen-port` property reflects the port on which to listen for SSL-based connections. The `ssl-cert-nickname` property should reflect the alias for the target certificate in the selected keystore.

20. Type **f** to save and apply the changes.
21. Verify that the server is properly configured to accept SSL-based client connections using `ldapsearch`. For example:

    ```
    $ bin/ldapsearch \
      --port 1636 \
      --useSSL \
      --baseDN "" \
      --searchScope base "(objectclass=*)"
    ```

    ```
    The server is using the following certificate:
      Subject DN: CN=179.13.201.1, OU=Server
      Certificate, O=Example Company, L=Austin, ST=Texas,
      C=US Issuer DN: EMAILADDRESS=whatever@example.com,
      CN=Cert Auth, OU=My Certificate Authority, O=Example
      Company, L=Austin, ST=Texas, C=US
      Validity: Fri Sep 25 15:21:10 CDT 2011 through Sat Sep 25 15:21:10 CDT
    2012
    Do you wish to trust this certificate and continue connecting to the
    server?
    Please enter 'yes' or 'no':yes
    ```

If necessary, disable the LDAP connection handler so only the LDAPS connection handler will accept connections.

Configure StartTLS

The StartTLS extended operation is used to initiate a TLS-secured communication channel over a clear-text connection, such as an LDAP connection. StartTLS provides a way to use a single connection handler for both secure and insecure communication, rather than requiring a dedicated connection handler for secure communication.

1. Use `dsconfig` to configure the Connection Handler to allow StartTLS. The `allow-starttls` property cannot be set if SSL is enabled. The connection handler must also be configured with a key manager provider and a trust manager provider.

    ```
    $ bin/dsconfig set-connection-handler-prop \
      --handler-name "LDAP Connection Handler" \
      --set allow-start-tls:true \
      --set key-manager-provider:JKS \
      --set trust-manager-provider:JKS
    ```

2. Use `ldapsearch` to test StartTLS.

    ```
    $ bin/ldapsearch -p 1389 --useStartTLS -b "" -s base "(objectclass=*)"
    ```

    ```
    The server is using the following certificate:
       Subject DN: CN=Server Cert, OU=Server Certificate,
         O=Example Company, L=Austin, ST=Texas, C=US
       Issuer DN: EMAILADDRESS=whatever@example.com, CN=Cert Auth,
         OU=My Certificate Authority, O=Example Company, L=Austin, ST=Texas,
    ```

```
C=US
   Validity: Thu Oct 29 10:29:59 CDT 2013 through Fri Oct 29 10:29:59
CDT 2014

Do you wish to trust this certificate and continue connecting to the
server?
Please enter 'yes' or 'no':yes
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 6fa8f196-d112-40b4-b8d8-93d6d44d59ea
```

Key manager providers

When a server needs to provide a certificate to another system, it will use a key manager provider to access that certificate. The server offers a key manager provider type for each of the supported keystore types, and the Server SDK can also be used to add support for accessing certificates in other ways if desired.

For each of the key manager provider types provided with the server, a PIN is required to access the keystore content. That PIN can be made available using one of the following properties:

- `key-store-pin` – Specifies the PIN used to access the keystore contents. It will be obscured, but a dedicated attacker with access to the configuration may be able to determine the clear-text value.
- `key-store-pin-file` – Specifies the path to a file containing the keystore PIN. The PIN must be stored in clear text, but filesystem permissions and/or access controls can be used to limit access.
- `key-store-pin-property` – Specifies the name of a Java property that holds the keystore PIN in clear text. This is not recommended, because anyone with access to JVM information or server monitor output may be able to determine the keystore PIN.
- `key-store-pin-environment-variable` – Specifies the name of a system environment variable that will hold the clear-text keystore PIN. This is not recommended, because anyone with access to the JVM process or server monitor output may be able to determine the keystore PIN.

For the PKCS#11 key manager provider, the keystore PIN is the only configuration element that needs to be provided. For the JKS and PKCS#12 key manager providers, it is also necessary to specify the path to the keystore file, and it may also be necessary to specify a PIN to use to access the private key (also specified using one of the four methods listed above). It is not necessary to specify a private key PIN if the value is the same as the keystore PIN.

Trust manager providers

When a server is presented with a certificate, it must determine whether that certificate should be trusted. This determination is made by a trust manager provider. The server provides support for three trust manager providers by default:

- **Blind trust** – Automatically accepts any certificate that is presented. This can be helpful for testing and/or debugging purposes, but is discouraged in production environments.
- **JKS** – Consults a JKS-format truststore file in order to determine whether to accept a given certificate. In order for a presented certificate to be trusted, either that certificate, or a certificate in its chain of issuers, must be present in the truststore file.
- **PKCS#12** – Operates in much the same way as the JKS trust manager provider, except that it consults a file in PKCS#12 format rather than a file in JKS format.
- **PKCS#11** – Interacts with certificates stored in some other repository, such as a hardware security module (HSM) or cryptographic accelerator. If security requirements require hardware protection for certificates, use PKCS#11. It may be necessary to use specialized tools provided by the vendor of the PKCS#11 token to manage certificates for use with that token.

The Server SDK can be used to create additional trust manager providers.

Configure the Key and Trust manager providers

Ping Identity servers support the following trust and key managers:

- JKS Key Manager Provider and Trust Manager Provider.
- PKCS#11 Key Manager Provider and Trust Manager Provider.
- PKCS#12 Key Manager Provider and Trust Manager Provider.

Perform the following steps to enable a key manager and trust manager and assign a connection handler with `dsconfig`:

1. Change location to the server root:

   ```
   $ cd /PingData<server>
   ```

2. Create a text file containing the password for the certificate keystore. It is recommended that file permissions (or filesystem ACLs) be configured so that the file is only readable

by the server user.

```
$ echo ’changeit’ > config/keystore.pin
$ chmod 0400 keystore.pin
```

3. Use the `dsconfig` to enable the key manager provider.

```
$ bin/dsconfig set-key-manager-provider-prop \
  --provider-name <JKS, PKCS11, or PKCS12> \
  --set enabled:true \
  --set key-store-file:/config/<Keystore.jks, keystore.p11 or
keystore.p12> \
  --set key-store-type:<JKS, PKCS11 or PKCS12> \
  --set key-store-pin-file:/config/keystore.pin
```

4. Use `dsconfig` to enable the trust manager provider.

```
$ bin/dsconfig set-trust-manager-provider-prop \
  --provider-name <JKS, PKCS11, or PKCS12> \
  --set enabled:true \
  --set trust-store-file:/config/<truststore.jks, truststore.p11, or
truststore.p12>
```

5. Use `dsconfig` to enable the LDAPS connection handler. Port 636 is typically reserved for LDAPS. If the certificate alias differs from the default `server-cert`, use the `--set ssl-cert-nickname:<aliasname>` option to set it, or use the `--reset sslcert-nickname` option for the server to set the alias.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "LDAPS Connection Handler" \
  --set listen-port:1636 \
  --set enabled:true \
  --set ssl-cert-nickname:1 \
  --set key-manager-provider:<JKS, PKCS11, or PKCS12> \
  --set trust-manager-provider:<JKS, PKCS11, or PKCS12>
```

6. Test the listener port for SSL-based client connection on port 1636 to return the Root DSE. Type **yes** to trust the certificate.

```
$ bin/ldapsearch --port 1636 --useSSL --baseDN "" --searchScope base \
  "(objectclass=*)"
```

```
The server is using the following certificate:
Subject DN: CN=179.13.201.1, OU=Server Certificate, O=Example Company,
L=Austin, ST=Texas, C=US
Issuer DN: EMAILADDRESS=whatever@example.com, CN=Cert Auth, OU=My
Certificate Authority, O=Example Company, L=Austin, ST=Texas, C=US
Validity: Fri Sep 25 15:21:10 CDT 2013 through Sat Sep 25 15:21:10 CDT
2014

Do you wish to trust this certificate and continue connecting to the
server?
Please enter 'yes' or 'no':yes
```

7. If necessary, disable the LDAP Connection Handler so that communication can only pass through SSL.

   ```
   $ bin/dsconfig set-connection-handler-prop \
     --handler-name "LDAP Connection Handler" \
     --set enabled:false
   ```

Secure LDAP communication

There are four primary ways to secure communication with PingData servers:

- Provide an LDAP connection handler configured to accept SSL-based connections.
- Provide an LDAP connection handler configured to allow StartTLS.
- Configure an alternate mechanism, like IPSec, for securing communication between client and server systems.
- Run clients on the same systems as the target server so that communication can occur over the loopback interface.

Note

This section discusses configuring the server for SSL or StartTLS security. Configuring IPSec or other forms of network encryption are beyond the scope of this documentation. It is recommended that the number of processes running on server be limited to minimize risks from a local attack. In addition, SSL and StartTLS are the only ways to ensure end-to-end encryption between the client and server.

To configure an LDAP connection handler to require all incoming connections to use SSL, set its `use-ssl` property to `true`. Or, to allow it to support the use of StartTLS, set `allow-start-tls` to `true`. The same connection handler cannot be configured to use both SSL and StartTLS. However, multiple LDAP connection handlers are supported to allow both SSL and StartTLS.

If a connection handler is configured for either SSL or StartTLS, the following properties are used to customize its behavior:

- `key-manager-provider` – Specifies the key manager provider to access certificates that are presented to clients. This is required for either SSL or StartTLS.
- `trust-manager-provider` – Specifies the trust manager provider to determine whether to trust client certificates that are presented to the server. This is required for either SSL or StartTLS.
- `ssl-cert-nickname` – Specifies the nickname of the certificate that the key manager should use for SSL or StartTLS communication. If this is not provided, the key manager picks the first suitable certificate it finds in the keystore.

- `ssl-client-auth-policy` – Specifies whether the server will ask clients to provide their own certificates, and whether to continue communication with clients if they don't provide a certificate. Allowed values are:
    - `disabled` – The server will not request a client certificate.
    - `optional` – The server will request a client certificate, but will allow clients that don't provide one). The default value is `optional`.
    - `required` – The server will request a client certificate, and will terminate the connection of any client that does not provide one.
- `ssl-protocol` – Specifies the names of the SSL protocol versions that the server will accept. The set of supported protocols depends on the underlying JVM. Protocol names may include `SSLv3`, `TLSv1`, or `SSLv2Hello`. If no values are specified, the JVM's default set of supported protocols is used.
- `ssl-cipher-suite` – Specifies the names of the SSL cipher suites that the server will accept. The set of supported cipher suites depends on the underlying JVM. If no values are specified, the server will attempt to automatically determine the best cipher suites to use.
- `disable-tls-renegotiation` – Indicates whether to allow clients to request TLS renegotiation. This enables a client to request repeating the process of negotiating the SSL protocol, cipher, and symmetric key. This option is rarely used, and may present security vulnerabilities in some SSL implementations.
- `auto-authenticate-using-client-certificate` – Indicates whether the connection handler should attempt to authenticate the client connection if the client provides a certificate during SSL or StartTLS negotiation. Normally, a client certificate is not used for LDAP authentication unless the client explicitly sends a SASL EXTERNAL bind request.

In addition to accepting connections from LDAP clients, servers can attempt to establish LDAP connections to other servers. This is particularly true for the PingDirectoryProxy Server and PingDataSync Server, but it may also be the case for the PingDirectory Server. LDAP external server configuration objects are used to provide the settings to use for communicating with those servers, and they have a set of properties for configuring communication security, including:

- `connection-security` – Specifies the mechanism to secure communication with the target server. Values are `none`, `SSL`, or `StartTLS`.
- `key-manager-provider` – Specifies the key manager provider used to obtain a client certificate to present to the server, if one is requested during SSL or StartTLS negotiation.

- `trust-manager-provider` – Specifies the trust manager provider used to determine whether to trust the server's certificate during SSL or StartTLS negotiation.

Configure LDAP connection handlers

To configure an LDAP connection handler to require all incoming connections to use SSL, set its `use-ssl` property to `true`. To support the use of StartTLS, set `allowstart-tls` to `true`.

If a connection handler is configured for either SSL or StartTLS, then the following properties can be used to customize its behavior:

- `key-manager-provider` – Specifies the key manager provider to access the certificates presented to clients. This is required for either SSL or StartTLS.
- `trust-manager-provider` – Specifies the trust manager provider used to determine whether to trust client certificates presented to the server. This is required for either SSL or StartTLS.
- `ssl-cert-nickname` – Specifies the nickname of the certificate that the key manager should use for SSL or StartTLS communication. If this is not provided, the key manager picks the first suitable certificate it finds in the keystore.
- `ssl-client-auth-policy` – Specifies whether the server will ask clients to provide their own certificates, and whether to continue communication with clients if they don't provide a certificate. Allowed values are:
    - `disabled` – The server will not request a client certificate.
    - `optional` – The server will request a client certificate, but will allow clients that don't provide one. The default value is `optional`.
    - `required` – The server will request a client certificate, and will terminate the connection of any client that does not provide one.
- `ssl-protocol` – Specifies the names of the SSL protocol versions that the server will accept. The set of supported protocols depends on the underlying JVM. If no values are specified, the JVM's default set of supported protocols is used.
- `ssl-cipher-suite` – Specifies the names of the SSL cipher suites that the server will accept. The set of supported cipher suites depends on the underlying JVM. If no values are specified, the server will attempt to automatically determine the best cipher suites to use.
- `disable-tls-renegotiation` – Indicates whether to allow clients to request TLS renegotiation. This enables a client to request repeating the process of negotiating the SSL protocol, cipher, and symmetric key. This option is rarely used, and may present security vulnerabilities in some SSL implementations.

- `auto-authenticate-using-client-certificate` – Indicates whether the connection handler should attempt to authenticate the client connection if the client provides a certificate during SSL or StartTLS negotiation. Normally, a client certificate is not used for LDAP authentication, unless the client explicitly sends a SASL EXTERNAL bind request.

Configure external server communication

In addition to accepting connections from LDAP clients, servers can attempt to establish LDAP connections to other servers. This is particularly true for the PingDirectoryProxy Server and PingDataSync Server, but it may also be the case for the PingDirectory Server in certain circumstances. LDAP external server configuration objects are used to provide the settings to use for communicating with those servers. Properties for configuring communication security include:

- `connection-security` – Specifies the mechanism used to secure communication with the target server. Allowed values are `none`, `SSL`, or `StartTLS`.
- `key-manager-provider` – Specifies the key manager provider used to obtain a client certificate to present to the server if one is requested during SSL or StartTLS negotiation.
- `trust-manager-provider` – Specifies the trust manager provider used to determine whether to trust the server's certificate during SSL or StartTLS negotiation.

Preventing communication over insecure connections

The server should be configured to accept connections from clients that communicate with the server only over a secure connection. There are two simple ways to accomplish this:

- **Use LDAPS** – If all clients support the ability to use LDAP over SSL, disable any LDAP connection handlers not configured to use SSL communication.
- **Use StartTLS** – If some clients only support the ability to use StartTLS over an initially insecure connection, use the `reject-insecure-requests` global configuration property to reject any request other than a StartTLS extended request received over an insecure connection. If the server does not need to accept any requests from insecure clients, then this should be enabled. For more granular control, set the `allowed-insecure-request-criteria` global configuration property, which specifies a set of criteria to match LDAP requests that may be permitted over an insecure connection, even if `reject-insecure-requests` is `true`. Some types of requests will always be permitted, including StartTLS and start administrative session requests.

There may be some cases in which clients either cannot communicate securely or require insecure communication before using StartTLS. In these instances, quarantine those connections using a custom Client Connection Policy that only allows a minimal set of operations, and another policy that allows a broader range of operations once that client has used StartTLS. Also, consider the use of sensitive attribute definitions to prevent access to certain attributes over insecure connections, or block their access entirely.

Allow or Deny connections from specific clients

Three mechanisms can be used to configure the set of clients that are allowed to establish connections to the server:

- Client Connection Policies can be associated with connection criteria, and simple connection criteria objects provide `included-client-address` and `excluded-client-address` specify the set of clients that match that criteria. If no policy has criteria that matches a given connection, or if policy matches a client and has the `terminate-connection` property set to `true`, any connection for which that policy is selected is terminated.
- Each connection handler provides `allowed-client` and `denied-client` properties that can be used to restrict the set of clients allowed to establish connections to that connection handler.
- User entries can include a `ds-auth-allowed-address` operational attribute that can be used to specify the addresses of client systems from which that user is allowed to authenticate.

In each of these cases, address masks are used to target client systems. Address masks can be used to specify clients in the following ways:

- An individual IPv4 or IPv6 address, such as "`1.2.3.4`" or "`1234:5678:90ab:cdef:1234:5678:90ab:cdef`."
- An IPv4 address with one or more elements replaced with an asterisk as a wildcard, such as "`1.2.3.*`."
- An IPv4 address range using CIDR notation, which follows a base address with a slash to specify the number of significant bits, such as "`1.2.3.0/24`."
- An IPv4 address followed by a slash and a subnet mask, such as "`1.2.3.0/255.255.255.0`".
- As individual resolvable hostname, such as "`host.example.com`."

- As a resolvable name with one or more components replaced with an asterisk, such as "`*.example.com`." The asterisk will match exactly one component, so "`*.example.com`" will match "`a.example.com`," but not "`a.b.example.com`."
- As a resolvable domain (or sub-domain) name preceded by a period, such as "`.example.com`." This will match any number of components before the given domain, so "`.example.com`" will match both "`a.example.com`" and "`a.b.example.com`."

Secure replication communication

Replication between PingDirectory Servers requires SSL authentication and encryption on a separate port (default 8989), on which the PingDirectory Server replication server component listens. Each server has a private key created at startup and stored in the `config/ads-truststore` JKS KeyStore. This key is used to authenticate to other replication servers.

Secure HTTP communication

Some components of the directory environment, including the Administrative Console, and the SCIM server, use HTTP for communication. The most common way to secure this communication is to use HTTP over SSL (HTTPS) with the following two properties:

- `ssl-protocol` – Specifies the names of the SSL protocol versions that the server will accept. The set of supported protocols depends on the underlying JVM. Protocol names may include `SSLv3`, `TLSv1`, or `SSLv2Hello`. If no values are specified, the JVM's default set of supported protocols is used.
- `ssl-cipher-suite` – Specifies the names of the SSL cipher suites that the server will accept. The set of supported cipher suites depends on the underlying JVM. If no values are specified, the server will attempt to automatically determine the best cipher suites to use.

Another security mechanism, HTTP Strict Transport Security (HSTS), can be enabled. HSTS is a web security policy mechanism used to protect websites against protocol downgrade attacks and cookie hijacking. It enables web servers to declare that web clients should only interact with it using secure HTTPS connections. HSTS is an IETF standards track protocol and is specified in RFC 6797.

The HSTS Policy is communicated by the server to the client through the "Strict-Transport-Security" HTTPS response header field. The HTTP Connection Handler can be used to set response headers across all servlets and web applications. The HTTP Servlet Extension's response-header configuration property can be used to add or override defined response-headers. The following is the command for enabling an HSTS header:

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler"  \
  --set response-header:Strict-Transport-Security:max-age=31536000
```

Securing SNMP communication

The PingDirectory Server can expose some monitoring information over SNMP, and generate alerts as SNMP traps. If secure SNMP communication is needed, configure the server to operate as an SNMP subagent, and communicate with a master agent on the same system over the loopback interface. The SNMP master agent should also be configured to require SNMPv3 using the authPriv security level, which provides authentication and encryption for SNMP clients.

Securing JMX communication

The PingDirectory Server can also make monitor information and administrative alerts available over JMX. If JMX is used, it can be secured with SSL. It will also require authentication, and only users with the jmx-read privilege will be allowed to retrieve any information over JMX. Only users with the jmx-notify privilege will be allowed to subscribe to receive administrative alerts as JMX notifications.

Secure database communication

All products can be configured to write access and error log messages to a relational database, and the PingDataSync Server can use them as Sync Sources or Destinations.

When communicating with a relational database, the security features used to protect that communication depend on the type of database being used, and the JDBC driver used to interact with it. Many JDBC drivers support the use of SSL, which can be configured using arguments provided in the JDBC URL. See the documentation for the specific database and JDBC driver for details on how to configure this or other security features.

Securing syslog communication

The PingDirectory Server can be configured to deliver access and/or error log messages to a network syslog server over the standard UDP-based protocol. This communication does not allow for encryption, so the server should be configured to use a syslog server running on the local system where communication only occurs over the loopback interface.

To have the log messages delivered to a remote system, use loopback communication, but have the local syslog daemon act as an encrypted relay to a remote server. Open source and commercial syslog software (including `rsyslog` and `syslog-ng`) provide the ability to act as a syslog relay for this purpose.

Other network security configuration options

Some of the other configuration options related to securing network communication include:

- **Limit the Max Time for JVM Cache** – The global configuration includes a `network-address-cache-ttl` property, which can be used to control the maximum length of time that the JVM should cache the IP address for which a given hostname resolves. Setting a reasonable timeout (such as one hour) allows the server to recognize network changes which assign a different IP address to a given name in a timely manner.
- **Limit the Max Number of Connections** – The global configuration includes a number of properties that can be used to control the maximum number of connections that can be established to the server. This includes `maximum-concurrent-connections` (the absolute maximum number of connections allowed to the server at any time), `maximum-concurrent-connections-per-ip-address` (the maximum number of connections allowed from any individual IP address at any time), and `maximum-concurrent-connections-per-bind-dn` (the maximum number of connections allowed to be authenticated as any individual user at any time). If any connection limit has already been reached, then any subsequent connections are terminated.
- **Use Custom Post-Connect and Post-Disconnect Plug-ins** – The Server SDK can be used to develop custom post-connect and post-disconnect plug-ins. Post-connect plug-ins are invoked when the server accepts a new client connection, and may be used to terminate that connection if it is determined that it should not be allowed. Post-disconnect plug-ins are invoked just after an existing connection is closed, whether that closure is initiated by a client or by the server.

Limit the maximum time for JVM cache

The global configuration includes a `network-address-cache-ttl` property, which specifies the maximum length of time that the JVM is allowed to cache the IP address associated with a system hostname. Setting a reasonable time-to-live value allows the server to detect cases in which a network administrator changes the IP address with which a given hostname is associated. If no time-out is defined, the JVM can cache these mappings indefinitely, and it may be necessary to restart the server to detect such changes.

The global configuration also includes a number of properties that can be used to control the maximum number of connections established to the server. This includes `maximum-concurrent-connections` (the maximum number of connections allowed to the server at one time), `maximum-concurrent-connections-per-ip-address` (the maximum number of connections allowed from any individual IP address at any time), and `maximum-concurrent-connections-per-bind-dn` (the maximum number of connections allowed to be authenticated as any individual user at one time). If any connection limit is reached, any subsequent connections are terminated.

Appendix A: SSL details

SSL provides a relatively simple way for clients to establish a secure connection to servers without ever having communicated with those systems in the past.

Topics include:

Asymmetric and symmetric encryption

There are two basic kinds of encryption: symmetric encryption, and asymmetric encryption. With symmetric encryption, the same key is used for both encryption and decryption. Asymmetric encryption uses a different key to encrypt data than it does to decrypt it. Symmetric encryption is generally less expensive than asymmetric, but it requires both the sender and receiver to have the same key. It also requires that no one else have that key. Asymmetric encryption is more expensive, but the encryption key (the public key) can be made available to anyone as long as the decryption key (the private key) is carefully protected and known only to its owner. Anyone can use the public key to encrypt a message, but only the one holding the private key can decrypt it.

When using asymmetric encryption, the encryption and decryption keys are mathematically related, but in a way that makes it extremely difficult to derive one from the other. One interesting property of some kinds of asymmetric encryption is that not only is it possible to encrypt messages using the public key in a way that can only be decrypted with the private key, but it is also possible to encrypt messages using the private key in a way that can only be decrypted with the public key. Since the public key can be widely available, this isn't useful for protecting the encrypted data from unintended observers, but it does make it possible to prove that it was encrypted by the private key, and therefore it can be used as a type of digital signature.

SSL uses a combination of symmetric and asymmetric encryption. When a client establishes an SSL-based connection to a server, there is an initial negotiation in which the following occurs:

- The client tells the server that it wants to use SSL and provides information about how that communication should proceed, including information about the SSL protocol version and cipher types that it supports.
- The server compares what the client supports with what the server supports, and informs the client what SSL version and cipher should be used for the rest of the communication.
- If the client and server can't agree on an SSL version and cipher suite, the negotiation will fail.

Certificates

The server sends information about its certificate to the client. This includes the public key, the subject (which is like a DN for the certificate), the time period for which that the certificate should be considered valid, and information about the certification authority (CA) that issued the certificate. The client can look at this information to determine whether to trust the certificate presented by the server. If not trusted, the server can cancel the negotiation.

The server can request that the client provide its own certificate to the server. If the client receives this type of request, it can send its certificate to the server. If the client does send a certificate to the server, the server uses it to decide whether to trust the client. If the client does not send a certificate, the server can decide to continue communicating with the client or not.

The client generates a symmetric encryption key, and then encrypts that through asymmetric encryption using the server's public key. This ensures that only the client and server know that key. The client and server will then switch to symmetric encryption using that newly-generated key.

As described, SSL has an element of trust in addition to providing encryption. Encryption isn't very useful if communicating with the wrong system, particularly when SSL is designed to make it easy for clients to communicate with servers with minimal knowledge of the server ahead of time. Although it is possible to configure clients so that they trust only the specific certificates configured for use by servers in the directory environment, much of the time this trust is based on a combination of the following elements:

- **The certification authority (CA) that issued the certificate** – Unless a certificate is self-signed, it will contain information about the CA that issued the certificate. Most clients are configured so that if they trust a certification authority, they will trust any certificate issued by that authority. Clients can be configured with information about a small number of CAs that are considered trustworthy, and have some process so that they will only issue a certificate for an organization after confirming that it was requested by an authorized representative.
- **The validity dates for the server certificate** – Nearly all clients will reject a certificate if it is expired (or not yet valid). It is important for administrators to be aware

of when their server certificates expire so that a replacement certificate can be installed prior to the expiration.

- **Agreement between the address of the system to which the connection has been established and the address contained in the certificate** – Most server certificates include information about the address of the system for which it is intended, either in the CN attribute of the certificate's subject, or in a `subjectAltName` extension. If the connection does not match an address contained in the certificate, many clients will reject that certificate because it may have come from an alternate system. Many clients do support wildcard certificates in which the server address contains a wildcard (such as "`*.example.com`") that can legitimately be used across multiple systems in the same organization, but these certificates are often very expensive.

Some clients may use a validation service, like checking certificate revocation lists (CRLs) or using the online certificate status protocol (OCSP), to determine whether a previously-valid certificate was revoked. If a certificate is compromised, mechanisms like CRLs or OCSP may be the easiest way to indicate that clients should no longer trust it. It is important to carefully protect the private portion of all server certificates to prevent the need to revoke them.

The process that the client uses to determine whether to trust the certificate presented by the server is called "server authentication." If the client presents its own certificate to the server, the server can also decide whether to trust that certificate and continue communicating with the client ("client authentication"). This doesn't necessarily mean that the client's certificate will actually be associated with a user in the directory and used for the purpose of LDAP authentication. It is possible to use the client certificate as a means of performing LDAP authentication using SASL EXTERNAL, or by configuring the connection handler to try to automatically authenticate the client using the certificate, but this is not done by default.

Appendix B: About the Java Keytool

Java Keytool is a key and certificate management utility, allowing users to manage their own public/private key pairs and certificates. The keytool utility comes with the standard JDK distribution and is located in the `JAVA_HOME/bin` directory.

Topics include:

Java Keytool utility use

If using a Java JKS KeyStore to hold server certificates, obtain a certificate to include. Most deployments will want to use a certificate that is signed by a certification authority so that clients can merely trust that CA and trust certificates signed by that CA.

Maintaining a CA can provide the greatest degree of flexibility, and can be significantly cheaper than using a commercial CA. However, it can also have a notable management overhead, and may require updating every client to trust the private CA certificate. A commercial certification authority can be used, which is relatively straightforward and likely already trusted by most clients.

Regardless of which certification authority used, a certificate signing request (CSR) must be generated that can be signed by the CA.

Create a server certificate

The Keytool utility enables management of public/private key pairs, x509 certificate chains and trusted certificates. The keys and certificates are stored in a keystore, which is a password-protected file with a default format of JKS. Each key and trusted certificate in the keystore is accessed by its unique alias.

The following procedure creates a keystore, generates a public/private key pair, and creates a self-signed certificate based on the key pair. This certificate can be used as the server certificate or it can be replaced by a CA-signed certificate chain with additional Keytool commands.

The `–dname` option is used to specify the certificate's subject, which is usually a CN attribute with a value equal to the fully-qualified name of the server. If the `–dname` option is omitted, the utility prompts for input. The certificate is valid for 180 days.

Perform the following steps to create a server certificate using Keytool:

1. Change to the directory where the certificates will be stored.

   ```
   $ cd /ds/PingData<server>/config
   ```

2. Use the keytool utility to create a private/public key pair and a keystore. The keytool utility is part of the Java SDK (`${JAVA_HOME}/bin`).

```
$ keytool -genkeypair \
  -dname "CN=server.example.com,ou=Data Metrics Server Certificate,
     O=Example Company,C=US"\
  -alias server-cert \
  -keyalg rsa \
  -keystore keystore \
  -keypass changeit \
  -storepass changeit \
  -storetype JKS \
  -validity 180 \
  -noprompt
```

The `-keypass` and `-storepass` arguments can be omitted to cause the tool to interactively prompt for the password. Also, the key password should match the keystore password.

3. View the keystore. The entry type is `privateKeyEntry`, which indicates that the entry has a private key associated with it, which is stored in a protected format to prevent unauthorized access. Also note that the **Owner** and **Issuer** are the same, indicating that this certificate is self-signed.

```
$ keytool -list -v -keystore keystore -storepass changeit
```

```
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: server-cert
Creation date: Sep 30, 2011
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=server.example.com, OU=Server Certificate, O=Example Company,
C=US
Issuer: CN=server.example.com, OU=Server Certificate, O=Example Company,
C=US
Serial number: 4ac3695f
Valid from: Wed Sep 30 09:21:19 CDT 2011 until: Mon Mar 29 09:21:19 CDT
2012
Certificate fingerprints:
MD5: 3C:7B:99:BA:95:A8:41:3B:08:85:11:91:1B:E1:18:00
SHA1: E9:7E:38:0F:1C:68:29:29:C0:B4:8C:08:2B:7C:DA:14:BF:41:DE:F5
Signature algorithm name: SHA1withRSA
Version: 3
```

4. If having a certificate signed by a Certificate Authority, skip to step 7. Otherwise export the self-signed certificate. Then examine the certificate.

```
$ keytool -export -alias server-cert -keystore keystore -rfc -file
server.crt
```

```
Enter keystore password:
Certificate stored in file <server.crt>
```

5. Import the self-signed certificate into a truststore. When prompted, type **yes** to trust the certificate.

```
$ keytool -importcert -alias server-cert -file server.crt \
  -keystore truststore -storepass changeit
```

6. View the truststore with the self-signed certificate. If using this certificate as the server certificate, this is the final step.

```
$ keytool -list -v -keystore truststore -storepass changeit
```

7. To create a production-ready certificate, continue by creating the Certificate Signing Request (CSR) by writing to the file server.csr. Follow the instructions of the third-party CA, and submit the file to a CA. The CA authenticates then returns a certificate reply, which can be saved as `signed.crt`.

```
$ keytool -certreq -v -alias server-cert -keystore keystore \
  -storepass changeit -file server.csr

Certification request stored in file <server.csr>
Submit this to your CA
```

8. If working with a third-party CA, both the key and trust stores should include information about the CA's root certificate as well as any intermediate certificates used to sign the server certificate. Obtain the CA root and any intermediate certificates to set up a chain of trust in your keystore. View the trusted CA and intermediate certificates to check that the displayed certificate fingerprints match the expected ones.

```
$ keytool -v -printcert -file root.crt
$ keytool -v -printcert -file intermediate.crt
```

9. Import the CA's root certificate in the keystore and truststore. If there are other intermediate certificates, then import them using the same commands, giving them each different aliases in the key and trust stores.

```
$ keytool -importcert -v -trustcacerts -alias cacert \
  -keystore keystore -storepass changeit -file root.crt
$ keytool -importcert -v -trustcacerts -alias cacert -keystore
truststore \
  -storepass changeit -file root.crt
```

10. Import the server certificate signed by the CA into the keystore, which will replace the existing self-signed certificate. When prompted, type **yes** to trust the certificate.

```
$ keytool -importcert -v -trustcacerts -alias server-cert -keystore
keystore
  -storepass changeit -file signed.crt
```

11. Add the certificate to the truststore.

    ```
    $ keytool -importcert -v -trustcacerts -alias server-cert \
      -keystore truststore -storepass changeit -file signed.crt
    ```

Create a client certificate

Client certificates can be used when stronger client authentication is desired, but is not required for SSL connections to be established. There are two important considerations when using client certificates:

- If a client presents its own certificate to the server, the server must be configured to trust that certificate.
- If the client certificates are used for LDAP authentication through SASL EXTERNAL, the certificate must contain enough information to allow the server to associate it with exactly one user entry. The requirements for this are dependent upon the certificate mapper configured for use in the server.

To create a PKCS#12 formatted client certificate with the Keytool utility, follow the steps in Create a server certificate and use the following command:

```
$ keytool -genkeypair \
  -dname "CN=server.example.com,ou=Certificate,O=Example Company,C=US"\
  -alias server-cert -keyalg rsa -keystore keystore.p12 -keypass changeit \
  -storepass changeit -storetype pkcs12 -validity 180 -noprompt
```

Appendix C: Understanding Criteria

The criteria subsystem provides a simple and powerful mechanism for classifying connections and operations. Understanding how to use criteria is an integral part of maintaining a secure directory environment.

Topics include:

Criteria overview

Simple connection criteria

Simple request criteria

Simple result criteria

Simple search entry criteria

Simple search reference criteria

Aggregate criteria

Criteria overview

The criteria subsystem is an integral part of many security-related features of PingData server products. Client Connection Policies use connection criteria to classify clients, and request criteria in the course of determining which requests should be allowed. The access logging subsystem uses all types of criteria to provide filtering support, which provides control over the kinds of messages that should be handled by each logger. Extensions like plug-ins, change subscription handlers, and virtual attributes can use criteria to identify connections and operations for which processing should be performed.

There are a number of criteria types in the server, including:

- **Connection criteria** – Used to classify client connections.
- **Request criteria** – Used to classify operation requests.
- **Result criteria** – Used to classify operation results.
- **Search entry criteria** – Used to classify search result entries encountered while processing a search.
- **Search reference criteria** – Used to classify search result references encountered while processing a search.

For each kind of criteria, there are multiple subtypes that can be used. Each kind of criteria has two subtypes:

- Simple criteria objects that provide a number of properties for use in the classification.
- Aggregate criteria objects that provide the ability to combine other criteria objects with Boolean logic.

Simple connection criteria

Simple connection criteria objects provide support for a number of properties that can be used to classify client connections. Some aspects deal with the method in which the client is communicating with the server, while others are based on the authenticated identity of the client.

Those properties dealing with the way the client communicates with the server include:

- `included-client-address` – Defines the client's IP address or resolved name that must match one of the given patterns.
- `excluded-client-address` – Defines the client's IP address or resolved name that must not match any of the given patterns.
- `included-connection-handler` – Specifies that the client's connection must have been accepted by one of the specified connection handlers.
- `excluded-connection-handler` – Specifies that the client's connection must not have been accepted by any of the specified connection handlers.
- `included-protocol` – Specifies that the name of the protocol that the client is using to communicate with the server must match one of the given values.
- `excluded-protocol` – Specifies that the protocol that the client is using to communicate with the server must not match any of the given values.
- `communication-security-level` – If defined, it may be used to perform matching based on whether the client is communicating with the server in a secure manner. Values include:
  - `secure-only` – The client must use secure communication.
  - `insecure-only` – The client must not use secure communication.
  - `any` – The client can use either secure or insecure communication.

The simple connection criteria properties that deal with the client's authentication state are listed below. All except `user-auth-type` are evaluated for authenticated client connections, and will be ignored for unauthenticated clients.

- `user-auth-type` – Performs matching based on whether, and possibly how, the client has authenticated. Values are `none` (matches unauthenticated clients), `simple` (matches clients authenticated with simple authentication), and `sasl` (matches clients authenticated with SASL authentication). To match only authenticated clients, include values `simple` and `sasl` but not `none`.
- `internal-authentication-security-level` – Performs matching based on whether the client authenticated in a secure manner. Values are `secure-only` (the client must have authenticated in a secure manner), `insecure-only` (the client must have authenticated in an insecure manner), or `any` (in which the client may have authenticated in either a secure or insecure manner).
- `included-user-sasl-mechanism` – If the client used SASL authentication, it will only match client connections in which the client authenticated using one of the specified SASL mechanisms. This is ignored for clients that have not performed SASL authentication.
- `excluded-user-sasl-mechanism` – If the client used SASL authentication, it will only match client connections in which the client did not authenticate using one of the

specified SASL mechanisms. This is ignored for clients that have not performed SASL authentication.

- `included-user-base-dn` – Matches client connections in which the authenticated user's entry is equal to or subordinate to one of the provided DNs.
- `excluded-user-base-dn` – Matches client connections in which the authenticated user's entry is not equal to or subordinate to one of the provided DNs.
- `all-included-user-group-dn` – Matches client connections in which the authenticated user is a member of all of the specified groups.
- `any-included-user-group-dn` – Matches client connections in which the authenticated user is a member of at least one of the specified groups.
- `not-all-included-user-group-dn` – Matches client connections in which the authenticated user is not a member of at least one of the specified groups. The authenticated user can be a member of zero or more of the groups, but must not be a member of all of them.
- `none-included-user-group-dn` – Matches client connections in which the authenticated user is not a member of any of the specified groups.
- `all-included-user-filter` – Matches client connections in which the authenticated user's entry matches all of the provided filters.
- `any-included-user-filter` – Matches client connections in which the authenticated user's entry matches at least one of the provided filters.
- `not-all-included-user-filter` – Matches client connections in which the authenticated user's entry does not match at least one of the provided filters. The authenticated user's entry may match zero or more of the provided filters, but must not match all of them.
- `none-included-user-filter` – Matches client connections in which the authenticated user's entry does not match any of the provided filters.
- `all-included-user-privilege` – Matches client connections in which the authenticated user has all of the specified privileges.
- `any-included-user-privilege` – Matches client connections in which the authenticated user has at least one of the specified privileges.
- `not-all-included-user-privilege` – Matches client connections in which the authenticated user does not have all of the specified privileges. The user may have zero or more of the privileges, but not all of them.
- `none-included-user-privilege` – Matches client connections in which the authenticated user does not have any of the specified privileges.

Simple request criteria

Simple request criteria objects provide support for matching a number of different kinds of requests. Some of the properties are based on the entry targeted by the requested operation.

- For add operations, this is the entry to be added.
- For bind operations, this is the specified bind DN (it will not look at SASL credentials to attempt to determine the target identity).
- For compare operations, this is the entry to be compared.
- For delete operations, this is the entry to be deleted.
- For modify operations, this is the original entry before any changes have been applied.
- For modify DN operations, this is the original entry before the DN has been altered.
- For search operations, this is the entry specified as the base DN.

Any properties referencing the target entry are ignored for abandon, extended, and unbind operations (and no attempt is made to look inside any extended request value).

Some properties reference a target attribute.

- For add operations, this is any of the attributes included in the entry to be added.
- For compare operations, this is the target attribute type.
- For modify operations, this is any of the attributes to be altered.
- For modify DN operations, this is any of the attributes included in the new RDN.
- For search operations, this is any of the attributes included in the search filter.

Any properties referencing the target attribute are ignored for abandon, bind, delete, extended, and unbind operations (and no attempt is made to look inside any extended request value).

- `operation-type` – Matches requests based on the type of operation requested.
- `operation-origin` – Matches requests based on the way the request was initiated. Values include `external-request` for requests initiated by an external client, `replicated-operation` for requests received through replication, or `internal-operation` for internal operations invoked by a plugin, or some other type of extension.
- `connection-criteria` – Matches requests based on information about the client that issued the request. At most, one connection criteria can be provided, but it may be an aggregate connection criteria, which combines multiple connection criteria objects.

- `all-included-request-control` – Matches requests in which the client included request controls with all of the specified object IDs. The request can include additional controls not included in this list.
- `any-included-request-control` – Matches requests in which the client included at least one request control with one of the specified object IDs. The request can include additional controls not included in this list.
- `not-all-included-request-control` – Matches requests in which the client did not include request controls with all of the specified object IDs. The request can include controls with zero or more of the specified object IDs, but not all of them.
- `none-included-request-control` – Matches requests in which the client did not include any request control with any of the specified object IDs. It can include control.
- `included-target-entry-dn` – Matches requests in which the target entry has a DN equal to or subordinate to one of the given values.
- `excluded-target-entry-dn` – Matches requests in which the target entry does not have a DN equal to or subordinate to any of the given values.
- `all-included-target-entry-filter` – Matches requests in which the target entry matches all of the provided filters.
- `any-included-target-entry-filter` – Matches requests in which the target entry matches at least one of the provided filters.
- `not-all-included-target-entry-filter` – Matches requests in which the target entry does not match all of the provided match filters.
- `none-included-target-entry-filter` – Matches requests in which the target entry does not match any of the provided filters.
- `all-included-target-entry-group-dn` – Matches requests in which the target entry is a member of all of the specified groups.
- `any-included-target-entry-group-dn` – Matches requests in which the target entry is a member of at least one of the specified groups.
- `not-all-included-target-entry-group-dn` – Matches requests in which the target entry is not a member of all of the specified groups. The target entry may be a member of zero or more of the specified groups, but not all of them.
- `none-included-target-entry-group-dn` – Matches requests in which the target entry is not a member of any of the specified groups.
- `target-bind-type` – Matches bind requests in which the authentication type matches one of the given values. Values are `simple` and `sasl`. This property is ignored for all operation types except bind.

- `included-target-sasl-mechanism` – Matches SASL bind requests in which the specified SASL mechanism is equal to one of the given values. This property will be ignored for non-bind requests, as well as for simple bind requests.
- `excluded-target-sasl-mechanism` – Matches SASL bind requests in which the specified SASL mechanism is not equal to any of the given values.
- `included-target-attribute` – Matches requests that target at least one of the specified attributes.
- `excluded-target-attribute` – Matches requests that do not target any of the specified attributes.
- `included-extended-operation-oid` – Matches extended requests in which the request object ID is equal to one of the given values. This property is ignored for non-extended requests.
- `excluded-extended-operation-oid` – Matches extended requests in which the request object ID is not equal to any of the given values. This property is ignored for non-extended requests.
- `using-administrative-session-worker-thread` – Performs matching based on whether the request is being processed using a dedicated administrative session worker thread. Values include:
  - `true` – Only match requests processed using an administrative session worker thread.
  - `false` – Only match requests not processed using an administrative session worker thread.
  - `any` – Use of an administrative session worker thread is not considered relevant.

Simple result criteria

Simple result criteria objects can be used to perform matching based on the result code of the operation, the length of time required to process that operation, the length of time the request remained on the work queue before being picked up for processing by a worker thread, controls included in the response, attempts to use privileges, and any entries or references returned during processing.

- `request-criteria` – Matches results for operations matching the provided request criteria. Only one request criteria object can be specified, but it may be an aggregate request criteria object, which combines multiple request criteria objects.
- `result-code-criteria` – Matching is performed based on the result code for the associated operation. Values include:

    - `all-result-codes` – The result code is not considered.
    - `non-failure-result-codes` – The associated operation must have completed successfully.
    - `failure-result-codes` - The associated operation must not have completed successfully.
    - `selected-result-codes` – The result code must match one of the values of the `result-code-value` property. For this property, the following result codes are considered successful: `success`, `compare-true`, `compare-false`, `referral`, `sasl-bind-in-progress`, and `no-operation`.
- `result-code-value` – Matches only operations with one of the specified result codes. This is only used if the `result-code-criteria` property has a value of `selected-result-codes`.
- `processing-time-criteria` – Matching is performed based on the length of time required for the worker thread to process the operation. Values include:
    - `any` – The processing time is not considered.
    - `less-than-or-equal-to`– The processing time must be less than or equal to the `processing-time-value`.
    - `greater-than-or-equal-to`– The processing time must be greater than or equal to the `processing-time-value`.
- `processing-time-value` – Performs matching based on the worker thread processing time for an operation. It is only used if a `processing-time-criteria value` of `less-than-or-equal-to` or `greater-than-or-equal-to` was specified.
- `queue-time-criteria` – Matching is performed based on the length of time the request was required to wait in the work queue before being picked up for processing by a worker thread. If this property has a value other than `any`, queue time monitoring must be enabled. Values include:
    - `any` – The queue time is not considered.
    - `less-than-or-equal-to` – The queue time must be less than or equal to the `queue-time-value`.
    - `greater-than-or-equal-to` – The queue time must be greater than or equal to the `queue-time-value`.
- `queue-time-value` – Matching is based on the queue time for an operation. It is only used if a `queue-time-criteria` value of `less-than-or-equal-to` or `greater-than-or-equal-to` is set.
- `referral-returned` – Matching is performed based on whether any referral URLs were included in the result. Values are:

  - `required` – The result must include one or more referral URLs.
  - `prohibited` – The result must not include any referral URLs.
  - `optional` – The inclusion of referral URLs is not considered.
- `all-included-response-control` – Matches results which contained response controls with all of the specified object IDs.
- `any-included-response-control` – Matches results that contain at least one response control with one of the given object IDs.
- `not-all-included-response-control` – Matches results that do not contain response controls with all of the given object IDs. It may contain response controls with zero or more of the given object IDs, but not all of them.
- `none-included-response-control` – Matches results that do not contain response controls with any of the given object IDs.
- `used-alternate-authzid` – Matching is performed based on whether the operation was processed using an authorization identity that differs from the authentication identity (the client used the proxied authorization or intermediate client controls, or a SASL alternate authorization identity). Values include:
  - `required` – The operation must have been processed with an alternate authorization identity.
  - `prohibited` – The operation must not have been processed with an alternate authorization identity.
  - `optional` – The use of an alternate authorization identity is not considered.
- `used-any-privilege` – Matching is performed based on whether the client made use of any privileges during processing. Values include `required` (the client must have used at least one privilege), `prohibited` (the client must not have used any privileges), or `optional` (the use of privileges is not considered).
- `used-privilege` – The client must have used at least one of the specified privileges.
- `missing-any-privilege` – Matching is performed based on whether the client attempted to perform any operation for which it did not have at least one required privilege. Values include:
  - `required` – The client must have been missing at least one privilege needed for the operation.
  - `prohibited` – The client must not have been missing any of the required privileges.
  - `optional` – Missing privileges are not considered.
- `missing-privilege` – At least one of the specified privileges was required for processing the operation, but the client did not have the necessary privilege.

- `search-entry-returned-criteria` – Matching is performed based on the number of matching entries returned to the client during search processing. This is ignored for non-search operations. Values include:
    - `any` – The number of entries returned is not considered.
    - `equal-to` – The number of entries returned must match the `search-entry-returned-count` value.
    - `not-equal-to` – The number of entries returned must not match the `search-entry-returned-count` value.
    - `less-than-or-equal-to` – The number of entries returned must be less than or equal to the `search-entry-returned-count` value.
    - `greater-than-or-equal-to` – The number of entries returned must be greater than or equal to the `search-entry-returned-count` value.
- `search-entry-returned-count` – Specifies the number of search result entries to use when performing matching based on the `search-entry-returned-criteria` property.
- `search-reference-returned-criteria` – Matching is performed based on the number of search result references returned to the client during search processing. This is ignored for non-search operations. Values include:
    - `any` – The number of references returned is not considered.
    - `equal-to` – The number of references returned must match the `search-reference-returned-count` value.
    - `not-equal-to` – The number of references returned must not match the `search-reference-returned-count` value.
    - `less-than-or-equal-to` – The number of references returned must be less than or equal to the `search-reference-returned-count` value.
    - `greater-than-or-equal-to` – The number of references returned must be greater than or equal to the `search-reference-returned-count` value.
- `search-reference-returned-count` – Specifies the number of search result references to use when performing matching based on the `search-reference-returned-criteria` property.

Simple search entry criteria

Simple search entry criteria objects may be used to perform matching based on the contents of search result entries returned to the client. Note that for properties used to perform matching based on a filter, that filter will be evaluated against the entry actually being returned to the client rather than the complete entry contained in the server.

- `request-criteria` – If specified, only matches search result entries for search operations matching the provided request criteria. Only one request criteria object can be specified, but it may be an aggregate request criteria object, with multiple request criteria objects.
- `all-included-entry-control` – If specified, only matches search result entries containing controls with all of the specified object IDs.
- `any-included-entry-control` – If specified, only matches search result entries containing at least one control with one of the specified object IDs.
- `not-all-included-entry-control` – If specified, only matches search result entries that do not contain controls with all of the specified object IDs. It may contain controls with zero or more of the specified object IDs, but not all of them.
- `none-included-entry-control` – If specified, only matches search result entries that do not contain any controls with any of the specified object IDs.
- `included-entry-base-dn` – If specified, only matches search result entries in which the DN of that entry is equal to or subordinate to one of the given base DN values.
- `excluded-entry-base-dn` – If specified, only matches search result entries in which the DN of that entry is not equal to or subordinate to one of the given base DN values.
- `all-included-entry-filter` – If specified, only matches search result entries in which the pared-down entry matches all of the provided filters.
- `any-included-entry-filter` – If specified, only matches search result entries in which the pared-down entry matches at least one of the provided filters.
- `not-all-included-entry-filter` – If specified, only matches search result entries in which the pared-down entry does not match all of the provided filters. It can match zero or more of the provided filters, but must not match all of them.
- `none-included-entry-filter` – If specified, only matches search result entries in which the pared-down entry does not match any of the provided filters.
- `all-included-entry-group-dn` – If specified, only matches search result entries in which the entry is a member of all of the specified groups.
- `any-included-entry-group-dn` – If specified, only matches search result entries in which the entry is a member of at least one of the specified groups.
- `not-all-included-entry-group-dn` – If specified, only matches search result entries in which the entry is not a member of at least one of the specified groups. The entry may be a member of zero or more of the specified groups, but not all of them.
- `none-included-entry-group-dn` – If specified, only matches search result entries in which the entry is not a member of any of the specified groups.

Simple search reference criteria

Simple search reference criteria objects can perform matching based on the contents of search result references returned to the client. Properties for this type of criteria include:

- `request-criteria` – Matches search result references for search operations matching the provided request criteria. Only one request criteria object can be specified, but it may be an aggregate, with multiple request criteria objects.
- `all-included-reference-control` – Matches search result references containing controls with all of the specified object IDs.
- `any-included-reference-control` – Matches search result references containing at least one control with one of the specified object IDs.
- `not-all-included-reference-control` – Matches search result references that do not contain controls with all of the specified object IDs. It may contain controls with zero or more of the specified object IDs, but not all of them.
- `none-included-reference-control` – Matches search result references that do not contain any controls with any of the specified object IDs.

Aggregate criteria

Each kind of criteria has an aggregate subtype that can be used to create logical ANDs, ORs, and NOTs of other criteria objects. For example, an aggregate connection criteria type can include the following properties:

- `all-included-connection-criteria` – Identifies client connections that match all of the referenced connection criteria objects. If one or more of the referenced criteria objects do not match a client connection, the aggregate connection criteria will not match that connection.
- `any-included-connection-criteria` – Identifies client connections that match at least one (but possibly more) of the referenced connection criteria objects. If none of the referenced criteria objects do not match a client connection, the aggregate connection criteria will not match that connection.
- `not-all-included-connection-criteria` – Identifies client connections that do not match at least one (and possibly none of) the referenced connection criteria objects. Connections may match one or more of the referenced connection criteria objects, as long as at least one of the referenced connection criteria objects does not match the connection.

- `none-included-connection-criteria` – Identifies client connections that do not match any of the referenced connection criteria objects. If one or more of the referenced criteria objects do match a client connection, then the aggregate connection criteria will not match that connection.

Other criteria types have an aggregate subtype with similar sets of `all-included`, `any-included`, `not-all-included`, and `none-included` properties.

Index