BigMemory Max Security Guide

Innovation Release

Version 4.3.5

April 2018

TERRACOTTA

This document applies to BigMemory Max Version 4.3.5 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: BMM-SG-435-20180417

Table of Contents

1 Overview of BigMemory Max Security

Security can be applied to both authentication (such as login credentials) and authorization (the privileges of specific roles).

We recommend that you plan and implement a security strategy that encompasses all of the points of potential vulnerability in your environment, including, but not necessarily limited to, your application servers (Terracotta clients), Terracotta servers in the TSA, the Terracotta Management Console (TMC), and any BigMemory .NET or C++ clients.

Note: Terracotta does not encrypt the data on its servers, but applying your own data encryption is another possible security measure.

Scope of the SSL documentation

SSL and Java Security configuration is complex and very environment specific. This documentation assumes that you already have a working SSL configuration, and that you wish to add Terracotta to that configuration. Introducing SSL and Java Security into an environment where there was previously no SSL or Java security is outside the scope of this documentation.

The documentation assumes that you have a solid understanding of SSL, Java Security, and related concepts. There are many freely accessible documents on the web to guide you in learning and understanding SSL and Java Security; typical terms to search for are *public key certificate*, *transport layer security (TLS)* and the *keytool utility*.

Some of the descriptions in the following sections give examples of how you can use third party tools to help you set up your environment. These tools are widely used in the context of Java Security and are extensively documented on the web site of the tool supplier. In such cases, we do not attempt to document all possible options of the tools and limit ourselves to mentioning just the options required.

Note: All commands or sequences of commands in the following descriptions for setting up the security configuration are intended as OUTLINES ONLY that describe the basics of getting SSL configured. The setups will generally NOT work out-of-the-box, since each customer has unique requirements. If you try to copy and paste the examples, your setup will probably not be valid. Therefore you should take the outlines only as a rough guide to what you need to do, and tailor the outlines to suite your own particular configuration.

Securing the Terracotta Cluster and Components

- Terracotta Server Array (TSA) using SSL, LDAP, JMX. See:
  - "Introduction to Security " on page 7
  - "About Security in a Cluster" on page 13

  - "Setting Up Server Security" on page 17
  - "Setting Up a TSA to Use the Terracotta Management Server" on page 33
  - "Using LDAP or Active Directory for Authentication" on page 43
  - "Using Encrypted Keychains" on page 49
- Terracotta Client (your application). See:
  - "Enabling SSL on Terracotta Clients" on page 27
  - "Using Encrypted Keychains" on page 49
  - "Serialization: Securing Against Untrusted Clients" on page 31
- Terracotta Management Console (TMC). See:
  - The *Terracotta Management Console User Guide*.
- BigMemory Max security using JMX Authentication. See:
  - "Configuring Security Using JMX Authentication" on page 9.
- BigMemory .NET and C++ clients. See:
  - The *Cross-Language Clients User Guide*.

2 Introduction to Security

Authentication Mechanisms

The Enterprise Edition of the Terracotta kit provides standard authentication mechanisms to control access to Terracotta servers. Enabling one of these mechanisms causes a Terracotta server to require credentials before allowing a JMX connection.

You can choose one of the following to secure servers:

- SSL-based security - Authenticates all nodes (including clients) and secures the entire cluster with encrypted connections. Includes role-based authorization.
- LDAP-based authentication - Uses your organization's authentication database to secure access to Terracotta servers.
- JMX-based authentication - provides a simple authentication scheme to protect access to Terracotta servers.

Note that Terracotta scripts cannot be used with secured servers without "passing credentials to the script" on page 10.

Configuring Security Using LDAP (via JAAS)

Lightweight Directory Access Protocol (LDAP) security is based on JAAS and requires Java 1.6. Using an earlier version of Java does not prevent Terracotta servers from running, but security will *not* be enabled.

To configure security using LDAP, follow these steps:

1. Save the following configuration to the file .java.login.config :

```
Terracotta {
com.sun.security.auth.module.LdapLoginModule REQUIRED
java.naming.security.authentication="simple"
userProvider="ldap://orgstage:389"
authIdentity="uid={USERNAME},ou=People,dc=terracotta,dc=org"
authzIdentity=controlRole
useSSL=false
bindDn="cn=Manager"
bindCredential="****"
bindAuthenticationType="simple"
debug=true;
};
```

   Edit the values for `userProvider` (LDAP server), `authIdentity` (user identity), and `bindCredential` (encrypted password) to match the values for your environment.

2. Save the file .java.login.config to the directory named in the Java property `user.home`.

3. Add the following configuration to each <server> block in the Terracotta configuration file:

```
<server host="myHost" name="myServer">
...
```

```
 <authentication>
   <mode>
     <login-config-name>Terracotta</login-config-name>
   </mode>
 </authentication>
...
</server>
```

4. Start the Terracotta server and look for a log message containing "INFO - Credentials: loginConfig[Terracotta]" to confirm that LDAP security is in effect.

> **Note:** If security is set up incorrectly, the Terracotta server can still be started. However, you might not be able to shut down the server using the shutdown script (stop-tc-server).

Configuring Security Using JMX Authentication

Terracotta can use the standard Java security mechanisms for JMX authentication, which relies on the creation of .access and .password files with correct permissions. The default location for these files for JDK 1.5 or higher is $JAVA_HOME/jre/lib/management.

To configure security using JMX authentication, follow these steps:

1. Ensure that the desired usernames and passwords for securing the target servers are in the JMX password file jmxremote.password and that the desired roles are in the JMX access file jmxremote.access.
2. If both jmxremote.access and jmxremote.password are in the default location ($JAVA_HOME/jre/lib/management), add the following configuration to each <server> block in the Terracotta configuration file:

```
<server host="myHost" name="myServer" jmx-enabled="true">
...
 <authentication />
...
</server>
```

3. If jmxremote.password is not in the default location, add the following configuration to each <server> block in the Terracotta configuration file:

```
<server host="myHost" name="myServer" jmx-enabled="true">
...
 <authentication>
   <mode>
     <password-file>/path/to/jmx.password</password-file>
   </mode>
 </authentication>
...
</server>
```

4. If jmxremote.access is not in the default location, add the following configuration to each <server> block in the Terracotta configuration file:

```
<server host="myHost" name="myServer" jmx-enabled="true">
...
 <authentication>
   <mode>
```

```
      <password-file>/path/to/jmxremote.password</password-file>
    </mode>
    <access-file>/path/to/jmxremote.access</access-file>
  </authentication>
...
</server>
```

If the JMX password file is not found when the server starts up, an error is logged stating that the password file does not exist.

Configuring SSL-based Security

To learn how to use Secure Sockets Layer (SSL) encryption and certificate-based authentication to secure enterprise versions of Terracotta clusters, see the following sections:

- "About Security in a Cluster" on page 13
- "Setting Up Server Security" on page 17
- "Enabling SSL on Terracotta Clients" on page 27

Note that using SSL to a Terracotta cluster reduces performance due to the overhead introduced by encrypting inter-node communication.

User Roles

There are two roles available for Terracotta servers and clients:

- admin - The user with the "admin" role is the initial user who sets up security. Thereafter, the "admin" user can perform system functions such as shutting down servers, clearing or deleting caches and cache managers, and reloading configurations.
- terracotta - This is the operator role. The default username for the operator role is "terracotta". The "terracotta" user can connect to the TMC and access the read-only areas. In addition, the "terracotta" user can start a secure server. But a user must have the "admin" role in order to run the stop-tc-server script.

Using Scripts Against a Server with Authentication

A script that targets a secured Terracotta server must use the correct login credentials to access the server. If you run a Terracotta script such as backup-data or server-stat against a secured server, pass the credentials using the `-u` (followed by username) and `-w` (followed by password) flags.

For example, if Server1 is secured with username "user1" and password "password", run the server-stat script by entering the following:

UNIX/LINUX

```
[PROMPT]${TERRACOTTA_HOME}/server/bin/server-stat.sh -s Server1 -u user1
-w password
```

MICROSOFT WINDOWS

```
[PROMPT]%TERRACOTTA_HOME%\server\bin\server-stat.bat -s Server1 -u user1
-w password
```

Extending Server Security

JMX messages are not encrypted. Therefore, server authentication does not provide secure message transmission after valid credentials are provided by a listening client. To extend security beyond the login threshold, consider the following options:

- Place Terracotta servers in a secure location on a private network.
- Restrict remote queries to an encrypted tunnel, such as one provided by SSH or stunnel.
- If using public or outside networks, use a VPN for all communication in the cluster.
- If using Ehcache, add a cache decorator to the cache that implements your own encryption and decryption.

3 About Security in a Cluster

Introduction

Terracotta clusters can be secured using authentication, authorization, and encryption. You can use:

- the built-in authentication and authorization
- external directory services
- certificate-based Secure Sockets Layer (SSL) encryption for communications between nodes

Security in a Terracotta cluster includes both server-server connections and client-server connections. Security must be enabled globally in the cluster. This ensures that each and every connection is secure, including connections in the Terracotta Management Server.

Security is set up using the Terracotta configuration, tools provided in the Terracotta kit, standard Java tools, and public key infrastructure (via standard digital X.509 digital certificates).

Security Related Files

Each Terracotta server uses the following types of files to implement security:

- Java keystore - Contains the server's *private key and public-key* certificate. The keystore is protected by a keystore/certificate-entry password.
- Truststore - A keystore file containing *only the public keys* of the certificates. This file is needed only if you are using self-signed certificates rather than a Certificate Authority (CA).
- Keychain - Stores passwords, including the passwords to the server's keystore and to entries in other files. The tools for creating and managing the Terracotta keychain file are provided with the Terracotta kit.
- Authorization - A .ini file with password-protected user accounts and their roles for servers and clients that connect to the server.

Note that Microsoft Active Directory and standard LDAP authentication/authorization are available options; see "Using LDAP or Active Directory for Authentication" on page 43 for related information.

Tip: The standard Java cacerts file, located in ${JAVA_HOME}java.home/lib/security by default, is a system-wide repository for CA root certificates included with the JDK. These certificates can play a part in certificate chains.

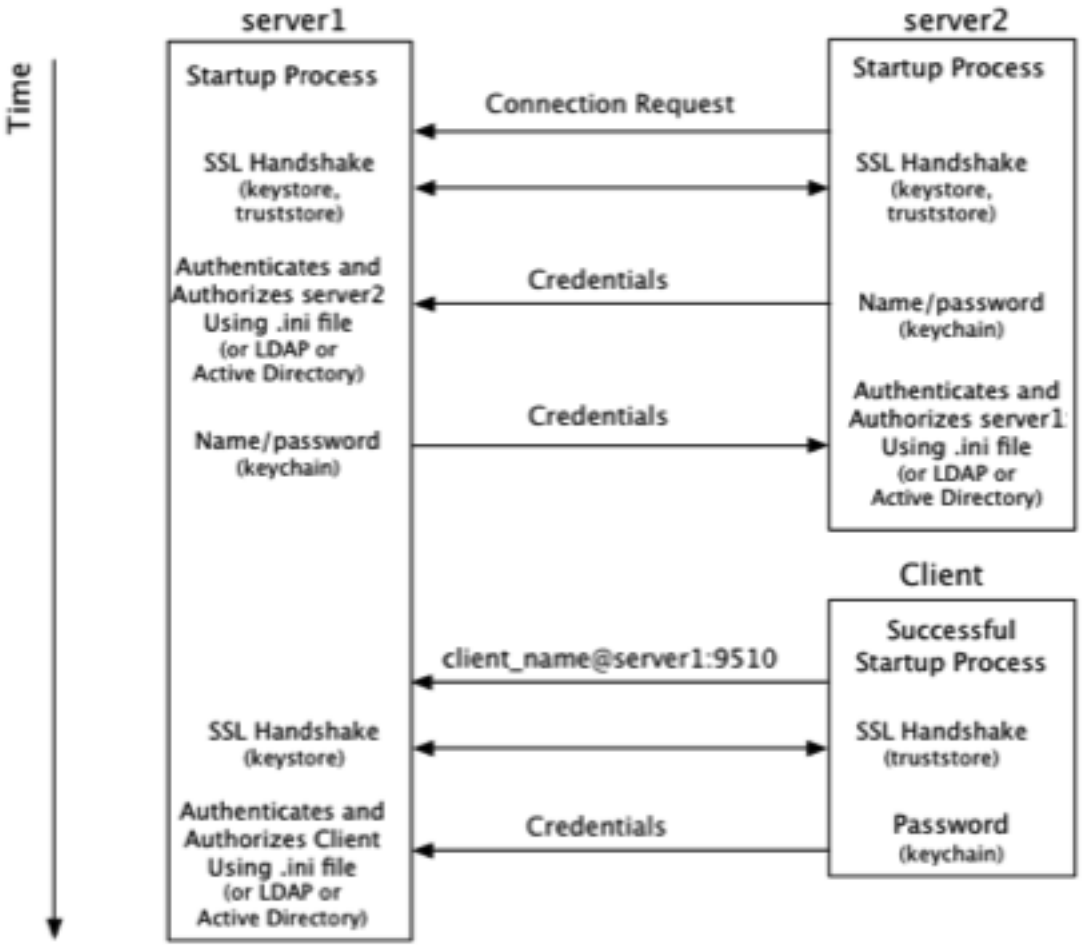
"Java documentation" recommends that the cacerts file be protected by changing its default password and file permissions.

Each Terracotta client also has a keychain file that stores the password it uses to authenticate with the server.

All files are read on startup. Changes made to the files after startup cannot be read unless the cluster is restarted.

Process Diagram

The following diagram illustrates the flow of security information during initial cluster connections. It also shows which security-related file originates the security information:



From a Terracotta *server* point of view, security checks take place at the time a connection is made with another node on the cluster:

1. After startup, servers can make connection requests to servers named in the configuration.
2. A connection request from server2 initiates the process of establishing a secure connection using SSL.

3. Server1 authenticates server2 using stored credentials. Credentials are also associated with a role that authorizes server2. The process is symmetrical: server2 authenticates and authorizes server1.
4. A connection request from a Terracotta client initiates the process of establishing a secure connection using SSL.
5. Server1 authenticates and authorizes the client using stored credentials and associated roles.

Because a client might communicate with any active server in the cluster during its lifetime, the client must be able to authenticate with any active server. Clients should be able to authenticate against *all* servers in the cluster because active servers might fail over to mirror servers.

From a Terracotta *client* point of view, security checks occur at the time the client attempts to connect to an active server in the cluster:

1. The client uses a server URI that includes the client username.

   A typical (non-secure) URI is <server-address>:<port>. A URI that initiates a secure connection takes the form <client-username>@<server-address>:<port> .
2. A secure connection using SSL is established with the server.
3. The client sends a password fetched from a local keychain file. The password is associated with the client username.

4 Setting Up Server Security

Basic Steps to Set Up Server Security

To set up security on a Terracotta server, follow the steps in the following procedures.

Note: Script names in the examples given below are for UNIX or Linux systems. Equivalent scripts are available for Microsoft Windows in the same locations. For Windows, replace the .sh extension with .bat and convert path delimiters as appropriate.

Creating the Server Certificates

Each Terracotta server must have a keystore file containing a digital certificate and the associated private key. This document assumes that you will use self-signed certificates.

IMPORTANT SECURITY CONSIDERATION! : Self-signed certificates might be less safe than CA-signed certificates because they lack third-party identity verification and do not carry a digital signature from an official CA. Your organization might already have policies and procedures in place regarding the generation and use of digital certificates and certificate chains, including the use of certificates signed by a Certificate Authority (CA). To follow your organization's policies and procedures regarding using digital certificates, you might need to adjust the procedures outlined in this document.

When used for a Terracotta server, the following conditions must be met for certificates and their keystores:

- The keystore must be a Java keystore (JKS) compatible with JDK 1.6 or higher.
- The certificate must be keyed with the alias named in the value of the `<certificate>` element of the server's configuration. See "Configure Server Security" on page 24 for details.
- The Common Name (CN) field in the Distinguished Name must contain the hostname of the server, as configured in "Configure Server Security" on page 24.
- The password securing the certificate must match the keystore's main password. In other words, the store password and key passwords must be identical.
- When using a self-signed certificate (not one signed by a trusted CA), create a custom truststore for storing public keys. See the section "Exporting and Importing Certificates" on page 19 for details.

  Note: When using a self-signed certificate (not one signed by a trusted CA), use the -k option for stopping the server or running server scripts.

If you have a keystore in place, but the server certificate is not already stored in the keystore, you must import it into the keystore. If the keystore does not already exist, you must create it.

Creating Self-Signed Certificates Using Java Keytool

For testing purposes, or if you intend to use self-signed certificates, use the Java keytool command to create the public-private key pair. You also use this command to create keystores and truststores, but note that keytool refers to truststores as "keystores" since there is only a logical difference.

Note: Specifying a Custom Truststore - Note that if you are *not* using cacerts, the default Java truststore, the custom truststore must be specified with the `javax.net.ssl.trustStore` system property. In this case, you can choose to reset the custom truststore's default password with `javax.net.ssl.trustStorePassword`.

The following could be used to create both public-private keys (including a certificate) and a keystore file for the server called "server1" in the configuration example above:

```
keytool -genkey -keystore keystore-file.jks
-dname "CN=172.16.254.1, OU=Terracotta, O=SAG, L=San Francisco, S=California,
C=US" -alias server1alias -storepass server1pass -keypass server1pass
```

Note that the values passed to `-storepass` and `-keypass` match. Also, the field designating the Common Name (CN) must match the server's hostname, which matches the value entered in the server's configuration. This hostname can be an IP address or a resolvable domain name. If the `-dname` option is left out, a series of identity prompts (distinguished-name fields based on the X.500 standard) will appear before the server's entry is created in the keystore. The CN prompt appears as shown:

```
What is your first and last name?
   [Unknown]:
```

There are a number of other keytool options to consider, including `-keyalg` (cryptographic algorithm; default is DSA) and `-validity` (number of days until the certificate expires; default is 90). These and other options are dependent on your environment and security requirements. For more information on using the keytool, see the JDK documentation.

Create a keystore and entry on each Terracotta server.

Exporting and Importing Certificates

Each server should have a copy of each other server's public-key certificate in its truststore.

The following could be used to export the certificate of the server called "server1" in the configuration example above.

```
keytool -export -alias server1alias -keystore keystore-file.jks \
    -file server1SelfSignedCert.cert
```

This "cert" file can now be used to import server1's certificate into the truststore of every other server. For example, to create a truststore and import server1's certificate on server2, copy the cert file to the working directory on server2 and use the following command:

```
keytool -import -alias server1alias -file server1SelfSignedCert.cert \
    -keystore truststore.jks
```

After the password prompt, information about the certificate appears, and you are prompted to trust it. You must repeat this process until every server has a truststore containing the public-key certificate of every other server in the cluster.

Tip: Use a Single Truststore - Instead of recreating the truststore on every server, create a single truststore containing every server's public key, then copy that to every server. This same truststore can also be used for clients. See "Using a Client Truststore" on page 29.

When you use the keytool utility, you can maintain additional certificates for the chain of trust in a file cacerts. If you wish to use these additional certificates for the import, refer to the use of the option `-trustcacerts` in the documentation of the keytool utility.

Tip: As an alternative to using the command line tool *keytool*, you might want to try the open source graphical tool *KeyStore Explorer*, available at "http://www.keystore-explorer.org/index.html".

Setting up the Server Keychain

The keystore and each certificate entry are protected by passwords stored in the server keychain file. The location of the keychain file is specified in the value of the `<url>` element under the `<keychain>` element of the server's configuration file.

For example, with this "server configuration" on page 24, when the server starts up, the keychain file would be searched for in the user's (process owner's) home directory. In the configuration example, a keychain file called server1keychain.tkc is searched for when server1 is started.

The keychain file should have the following entries:

- An entry for the local server's keystore entry.
- An entry for every server that the local server will connect to.

Entries are created using the keychain script found in the Terracotta kit's tools/security/bin directory.

Creating an Entry for the Local Server

Create an entry for the local server's keystore password:

```
tools/security/bin/keychain.sh -O <keychain-file> <certificate-URI>
```

where <keychain-file> is the file named in the server configuration's <keychain>/<url> element (including correct path), and <certificate-URI> is the URI value in the server configuration's <ssl>/<certificate> element.

Note: The `<certificate-URI>` must match the server configuration's `<ssl>/<certificate>` element exactly, including the path to the keystore.

By default, the keychain file stores passwords using an obfuscation scheme, requiring the use of `-O` (hyphen capital letter O) with the keychain script for *any* operation on the file. To switch a more secure encryption-based scheme, see "Using Encrypted Keychains" on page 49.

If the keychain file does not exist, add the `-c` option to create it:

```
tools/security/bin/keychain.sh -O -c <keychain-file> <certificate-URI>
```

You will be prompted to enter a password to associate with the URI. *You must enter the same password used to secure the server's certificate in the keystore.*

For example, to create an entry for server1 from the configuration example above, enter:

```
tools/security/bin/keychain.sh -O server1keychain.tkc
   jks:server1alias@/the/path/keystore-file.jks
Terracotta Management Console - Keychain Client
Enter the password you want to associate with this URL:   server1pass
Confirm the password to associate with this URL:   server1pass
Password for jks:server1alias@/the/path/keystore-file.jks successfully stored
```

Creating Entries for Remote Servers

Entries for remote servers have the format `tc://<user>@<host>:<group-port>`. Note that the value of `<user>` is specified in each server configuration's `<security>/<auth>/<user>` and is *not* related to the user running as the process owner. If a value for `<security>/<auth>/<user>` is not specified, the username "terracotta" is used by default.

For example, to create an entry for server2 in server1's keychain, use:

```
tools/security/bin/keychain.sh -O server1keychain.tkc
   tc://server2username@172.16.254.2:9530
```

If the keychain file does not exist, add the `-c` option:

```
tools/security/bin/keychain.sh -O -c server1keychain.tkc
   tc://server2username@172.16.254.2:9530
```

You will be prompted to enter a password to associate with the entry `server2username@172.16.254.2:9530`.

An entry for server1 must also be added to server2's keychain:

```
tools/security/bin/keychain.sh -O server2keychain.tkc
   tc://server1@172.16.254.1:9530
```

Setting up Authentication/Authorization

Servers and clients that connect to a secured server must have credentials (usernames/passwords) and roles (authorization) defined. This section discusses the authentication/authorization mechanism based on using a .ini file. To use LDAP or Microsoft Active Directory instead, see "LDAP and Active Directory setup page" on page 43.

Authentication and authorization are set up using the usermanagement script, located in the Terracotta kit's tools/security/bin directory. This script also creates the .ini file that

contains the required usernames and roles. The associated passwords are stored in the keychain file.

All nodes in a secured Terracotta cluster must have an entry in the server's .ini file:

- The local server itself
- All other servers
- All clients

Use the usermanagement script with the following format:

```
tools/security/bin/usermanagement.sh -c <file> <username> terracotta
```

where `-c` is required only if the file does not already exist. For servers, the `<username>` will be used as the value configured in `<security>/<auth>/<user>`. For clients, the username must match the one used to start the client.

Note: While the "terracotta" role is appropriate for Terracotta servers and clients, the "admin" role is necessary for performing system functions such as stopping servers. For more information about roles, refer to "User Roles" on page 10.

For example:

```
# Create the .ini file and add a server username and role.
tools/security/bin/usermanagement.sh -c my_auth.ini server1username terracotta
# Add another server.
tools/security/bin/usermanagement.sh my_auth.ini server2username terracotta
# Add a client.
tools/security/bin/usermanagement.sh my_auth.ini client1username terracotta
# Add a user with an "admin" (read/write) role.
tools/security/bin/usermanagement.sh my_auth.ini admin1username admin
# Add a user with a "terracotta" (read) role.
tools/security/bin/usermanagement.sh my_auth.ini console1username operator
```

The correct Apache Shiro Realm must be specified in the "server configuration" on page 24, along with the path to the .ini file:

```
...
<auth>
  <realm>com.tc.net.core.security.ShiroIniRealm</realm>
  <url>file:///%(user.dir)/my_auth.ini</url>
  <user>server1username</user>
</auth>
...
```

Setting up Authorization for TMC Queries

The Terracotta Management Console allows you to execute SQL-like queries in the query field of the **Application Data > Contents** panel. Initially, all users who can access the TMC can also use this query feature.

You may want to restrict the usage of the query feature by disabling it for certain user roles/identities. You can do this as follows:

Using Simple Account-Based Authentication (Ini-File) security

If you are using simple account-based authentication security, the authorization setup for disabling the query feature is defined in the security.ini file, which is located in the mgmt folder under your user account path at the operating system level. The file is created when you use the TMC to specify that you want to use Ini-File authentication. If you have not yet done so, you can use the **Settings** menu of TMC to specify the required authentication method. On Windows systems, the location of the mgmt folder could be, for example, C:\Users\MyUserName\.tc (where *MyUserName* is your Windows user name), and on Linux it could be ~/.tc.

Use the following steps to disable the query feature:

1. Open the file security.ini in a text editor and go to the line where the user that you want to modify is defined.
2. Append the `nobmsql` role at the end of the line in order to disable the query panel for that user.
3. Restart TMS, then log in using the user's credentials, and ensure that the query field is no longer visible in the **Application Data > Contents** panel.

Here is an example of security.ini, with `nobmsql` applied to the "operator" user:

```
[users]
admin=$shiro1$SHA-1$1000000$pibMTfX7zzyKTy57DLcSvw==$ENBPZPwB//L5fbVZ+/jeKJ4Fm/4=,operator,admin
operator=$shiro1$SHA-1$1000000$3mYdIqq2gjldlii7qaadsg==$tIMdM92xA6UXwXZn/MeH2AH7N8A=,operator,nob
```

There are 2 users in this configuration: "admin" and "operator". The long string behind '=' is the encrypted password, which is automatically generated the first time you configure the password through TMC. Currently there are 3 roles available: "admin", "operator" and "nobmsql". An administrator user needs to be assigned both "admin" and "operator" roles. An operator user needs to be assigned the "operator" role. If you want to hide the query panel from the administrator user or operator user or both, you can simply add the "nobmsql" role to that user.

Using LDAP-based security

If you are using LDAP based security, use the following steps to disable the query feature for a particular user:

1. Using a text editor, open the file shiro.ini in the mgmt folder (location as described above for Ini-File security).
2. Find the entry `ldapRealm.groupRolesMapAsString =`.

   This is the mapping string between TMC roles and LDAP groups. It is formatted as:

   ```
   "LDAP group":"TMC role[s]..."
   ```

   for example:

   ```
   "tmcopstgroup2":"admin,operator";
   ```

3. If you want to disable the query ability for an LDAP group, add ",nobmsql" behind the mapping.

4. Restart TMS, then log in using the credentials of a user who belongs to the LDAP group. Ensure that the query field is no longer visible in the **Application Data > Contents** panel.

Configuring Server Security

Set up the security for the Terracotta Server Array in the Terracotta configuration file, which is named tc-config.xml by default. For example:

```
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
...
  <servers secure="true">
    <server host="172.16.254.1" name="server1">
    ...
      <security>
        <ssl>
          <certificate>jks:server1alias@/the/path/keystore-file.jks</certificate>
        </ssl>
        <keychain>
          <url>file:///%(user.dir)/server1keychain.tkc</url>
        </keychain>
        <auth>
          <realm>com.tc.net.core.security.ShiroIniRealm</realm>
          <url>file:///%(user.dir)/my_auth.ini</url>
          <user>server1username</user>
        </auth>
      </security>
   ...
   </server>
   ...
 </servers>
...
</tc:tc-config>
```

Every server participating in an SSL-based secured cluster must have a <security> block in which the security-related information is encapsulated and defined. The keystore, keychain, and .ini files named in the configuration must be available to every server in the cluster. "LDAP or Microsoft Active Directory" on page 43 can be configured in place of file-based authentication and authorization.

The following table defines some of the security-related elements and attributes shown in the configuration example.

Name	Definition	Notes
secure	Attribute in <servers> element. Enables SSL security for the cluster. DEFAULT: false.	Enables/disables SSL-based security globally.
certificate	Element specifying the location of the server's authentication certificate and its containing keystore file. The format for	Only the JKS type of keystore is supported.

Name	Definition	Notes
	the certificate-keystore location is `jks:alias@/path/to/keystore`. "alias" must match the value used to key the certificate in the keystore file.	
url	The URI for the keychain file (when under <keychain>) or for the authentication/authorization mechanism (when under <auth>). These URIs are passed to the keychain or realm class to specify the keychain file or authentication/authorization source, respectively.	These files are created and managed with the "keychain" on page 20 and "usermanagement" on page 21 scripts. If using Microsoft Active Directory or LDAP, an LDAP or LDAPS connection is specified. The configured URL for locating the keychain file can be overridden with the property `com.tc.security.keychain.url`.
realm	The Shiro security realm that determines the type of authentication/authorization scheme being used: file-based (.ini), Microsoft Active Directory, or standard LDAP.	This element's value is specified in the section covering the setup for the chosen authentication/authorization scheme.
user	The username that represents this server and is authenticated by other servers. This name is part of the server's credentials. Default username is "terracotta"	

5 Enabling SSL on Terracotta Clients

How to Enable SSL Securing on the Client

Terracotta clients do not require any specific configuration to enable SSL connections to a Terracotta Server Array.

Note: Script names in the examples given below are for UNIX and Linux systems. Equivalent scripts are available for Microsoft Windows in the same locations. Replace the .sh extension with .bat and convert the path delimiters as appropriate.

To enable SSL security on the client:

- Prepend the client username to the address used by the client to connect to the cluster.

  This should be the username that will be authenticated followed by an "at" sign ("@") and the address of an active server running in secure mode. The format is `<client-username>@<host>:<tsa-port>`. Prepending the username automatically causes the client to initiate an SSL connection.

  If the client has username `client1`, for example, and attempts to connect to the server in the configuration example, the address would be:

  ```
  client1@172.16.254.1:9510
  ```

  This URI replaces the address `<host>:<tsa-port>` used to start clients in non-SSL clusters.

- Verify that the client username and its corresponding password match those in the "server's .ini file" on page 21 or credentials in "LDAP or Active Directory" on page 43. The username is included in the URI, but the password must come from a "local keychain entry" on page 28 that you create.

  The client credentials must be associated with the role "terracotta" or "admin".

- If Terracotta servers are using self-signed certificates (not certificates signed by a well-known CA), then you must "specify a truststore for the client" on page 29 that contains the public key of every server in the cluster.

Creating a Keychain Entry

The Terracotta client should have a keychain file with an entry for every Terracotta server in the cluster. The format for the entry uses the "tc" scheme:

```
tc://<client-username>@<host>:<tsa-port>
```

An entry for the server in the example configuration should look like:

```
tc://client1@172.16.254.1:9510
```

Use the keychain script in the Terracotta kit to add the entry:

```
tools/security/bin/keychain.sh -O clientKeychainFile
tc://client1@172.16.254.1:9510
```

By default, the keychain file stores passwords using an obfuscation scheme, requiring the use of `-O` (hyphen capital letter O) with the keychain script for *any* operation on the file. To switch a more secure encryption-based scheme, see "Using Encrypted Keychains" on page 49.

If the keychain file does not already exist, use the -c flag to create it:

```
tools/security/bin/keychain.sh -O -c clientKeychainFile
tc://client1@172.16.254.1:9510
```

You will be prompted to enter a client password to associate with the URI.

This entry in the client's keychain file serves as the key for the client's password and is provided to the server along with the client username ("client1" in the example). *These credentials must match those in the* "server's .ini file" on page 21 *or* "LDAP or Active Directory credentials" on page 43.

The Terracotta client searches for the keychain file in the following locations:

- %(user.home)/.tc/mgmt/keychain
- %(user.dir)/keychain.tkc
- The path specified by the system property `com.tc.security.keychain.url`

Example Using the Keychain Script

When you run the keychain script, the following prompt should appear:

```
Terracotta Management Console - Keychain Client
KeyChain file successfully created in clientKeychainFile
Enter the password you wish to associate with this URL:
Password for tc://client1@172.16.254.1:9510 successfully stored
```

Note that the script does not verify the credentials or the server address.

Using a Client Truststore

If Terracotta servers are using self-signed certificates (not certificates signed by a well-known CA), create a truststore on the client and import each server's public-key certificate into that truststore.

If you have already "created a truststore" on page 19 for a server in the TSA, you can copy that file to each client after first importing that server's public-key certificate into the copy.

For the client to find the truststore, you must set the Java system property `javax.net.ssl.trustStore` to the location of the truststore file. In this case, note the existing secrets for opening the truststore and accessing each certificate.

Tip: Changing the Truststore Password - To change the existing truststore master password, use the Java system property `javax.net.ssl.trustStorePassword`.

6 Serialization: Securing Against Untrusted Clients

Typically, Java objects are serialized when writing to cache and then deserialized by clients reading from cache.

Where all cache clients are trusted (a common deployment pattern), Java deserialization poses no security issue.

However, in cases where a client could be an attacker, deserialization could be used to inject malicious code into another client. Specially crafted objects can be included in the serialized stream of bytes that, when deserialized by the Java deserialization process, lead to arbitrary code execution.

For the cache this, by nature, is not a security issue per se as no deserialization happens. But on the client side, such attacks should be mitigated.

The security issue with deserialization is a known issue in the technical community. As a result there are various solutions (such as blacklists/whitelists for classes) to address this issue.

Examples are NotSoSerial ("https://github.com/kantega/notsoserial/"), ikkisoft SerialKiller ("https://github.com/ikkisoft/SerialKiller/"), or "ValidatingObjectInputStream in Apache Commons IO".

7 Setting Up a TSA to Use the Terracotta Management Server

Required Configuration

Before you can connect the Terracotta Management Server (TMS) to your secured Terracotta Server Array (TSA), you must configure your TSA as described in the following sections.

Configuring Identity Assertion

Add the following to each server's <security> block:

```
<security>
...
  <management>
    <ia> https://my-tms.mydomain.com:9443/tmc/api/assertIdentity</ia>
    <timeout>10000</timeout>
    <hostname>my-l2.mydomain.com</ hostname >
  </management>
</security>
```

where:

- `<timeout>` is the timeout value in milliseconds for connections from the server to the TMS.
- `<ia>` is the HTTPS (or HTTP) URL with the domain of the TMS, followed by the port 9443 and the path /tmc/api/assertIdentity.

  If you are using HTTPS, which is recommended, export a public key from the TMS and import it into the server's truststore. You must also export a public key from the server and import it into the TMS's truststore, or copy the server's truststore (including the local server's public key) to the TMS.
- `<management><hostname>` is used only if the DNS hostname of the server does not match server hostname used in its certificate. If there is a mismatch, enter the DNS address of the server here.

You must export a public key from the TMS.

JMX Authentication Using the Keychain

The following is *required* for server-to-client REST-agent authorization. Every node in the cluster must have the following entry in its keychain, all locked with the identical secret:

```
jmx:net.sf.ehcache:type=RemoteAgentEndpoint
```

In addition, server-server REST-agent communication must also be authorized using a keychain entry with the following format:

```
jmx://<user>@<host>:<group-port>
```

Note that the value of `<user>` is specified in each server configuration's `<security>/<auth>/<user>` and is *not* related to the user running as process owner.

For example, to create an entry for server2 in server1's keychain, use:

```
tools/security/bin/keychain.sh -O server1keychain.tkc
jmx://server2username@172.16.254.2:9530
```

Each server must have an entry for itself and an entry for each other server in the TSA.

Setting up the Security on the TMS

An unsecured TMS cannot connect to a secured TSA. To learn how to set up security on the TMS, see the *Terracotta Management Console User Guide*.

Securing TSA Access using a Permitted IP List

The IP white-listing feature enables you as the cluster administrator to ensure that only clients from certain explicitly named IP addresses can access the TSA. You can use this feature, for example, to secure the TSA from malicious clients attempting to connect to the TSA. The term "clients" here refers to caching clients, JMX clients and HTTP clients. The so-called white-list is a list of IPs, and clients running on these IPs are allowed to access the TSA; any client whose IP is not in the white-list will not be allowed to access the TSA. You maintain the white-list of known client IPs in a plain text file. CIDR notations can also be used to cover a range of IPs.

Note: It should be understood that usage of this feature (on its own) does not provide a strong level of security for the TSA. Features such as SSL encryption and authentication should be enabled for true security. Additionally, the ideal way to enforce connection restrictions based on IP addresses would be to use host-level firewalls rather than this feature.

If you want to use white-listing, you need to enable it at server startup. Once the server has started with white-listing enabled, white-listing cannot be turned back off while the server is running. However, you can change the existing IP/CIDR entries in the white-list file while the server is running. Also you can add and delete entries in the white-list file while the server is running, in order to modify the set of clients that need access to the cluster.

If you do not switch on white-listing at server startup, you cannot switch on white-listing while the server is running.

In a multi-stripe cluster, you need to start up all servers (both actives and passives) with the same copy of the white-list file, and when there are updates to the white-list file, you need to ensure that the same changes are mirrored across all stripes. Note that the TSA does not do any cross-stripe validation on the contents of the white-list file, so it is your responsibility as the cluster administrator to make sure that this happens.

Usage

To enable white-listing, you need to start the server with the following tc-config property set:

```
ip.white.list="<location>/<file>"
```

where <location> is the path containing the white-list file, and <file> is the name of the white-list file.

White-listing is enabled/disabled based on the presence of this tc-config property entry. The value of this property is the complete path to the white-list file. If the file is not found at the specified location, the server startup will fail with an appropriate error message. If the file is present but there is an error reading the file, the server startup will continue with white-listing disabled, and the failure will be logged in the server log. If the property is not specified, white-listing is disabled.

The server IPs specified in the tc-config file of the server are always white-listed. If hostnames are used in the tc-config entries instead of IPs, the server will attempt to resolve these hostnames to IPs. If the resolution fails, server startup will continue with white-listing disabled. So when this IP white-listing feature is used, it is recommended to have only IPs configured for servers in the tc-config file. Similarly, localhost IPs are always white-listed too.

White-list file

The white-list file is a simple text file. You can choose any name for this file, for example white-list.txt. The entries can be raw IP addresses in IPv4 format or in CIDR notation to represent ranges. IPv6 entries are not supported. Each line in the file can contain either a single IP address or a comma-separated set of IP addresses. Any entry that is not a valid IPv4 address or a valid CIDR will be ignored. Lines starting with a # are skipped. Blank lines are also skipped . Here is a sample white-list file:

```
# The white-list for my cluster
# Caching clients
192.168.5.28, 192.168.5.29, 192.168.5.30
10.60.98.0/28
# Other clients
192.168.10.0/24
```

The white-list file must be kept in a directory where you have write permissions. Some files will be created by the server in the same directory for book-keeping purposes. If the server fails to create these files, the server startup will also fail.

Dynamic updates

Once a server is started with white-listing enabled, entries can be dynamically added/removed from the white-list file. The updates to the white-list file are processed by the server only when you signal the server to do so. When you have finished making the changes to the white-list file, you must execute the "update-white-list.sh" script packaged with the kit. The script takes the white-list file path as the argument. After you run the script the updates will take effect in a few moments and the corresponding log statements can be found in the server log.

As already mentioned, every server node (active and passive) has its own copy of the white-list file. So you need to update the white-list file and execute the script on each node separately. When you run the script, the server receives the update signal and reloads the white-list file, and the updated white-list entries are logged in the server log. Thus, after every update operation, you should check the server logs to verify if the updates took effect. Make sure that the updates took effect in all the servers in the cluster.

On every dynamic update, the server reads the contents of the white-list file and the tc-config to update its in-memory white-list. Reading the white-list file involves a disk IO, and reading a tc-config file with hostnames in it involves DNS lookup for hostname resolution. In both cases, failures are very well possible. So if such a failure happens after a dynamic update, the updates will be ignored and the server will continue with the current white-list. No partial updates will be applied. The update won't be retried either until the user signals so by running the update-white-list script again.

Client behaviors

This section details different client behaviors with white-listing enabled.

Caching clients

When a client connects to a server on the tsa-port, the server first accepts the socket connection, then verifies if the IP of the incoming client is white-listed and closes the socket connection if the client is not white-listed. In this case, the client will get an EOF on trying to read from the socket connection established with the server. If a client was white-listed initially and was removed from the white-list on a dynamic update, it will not be removed immediately from the cluster. Instead, the client will remain connected to the cluster as long as there is no network disconnection between the client and server. The client will be rejected only on the next reconnect attempt.

HTTP clients

Non-white-listed clients that send HTTP requests to the management port will get a 403 Forbidden response.

Restricting Clients to Specified Servers (Optional)

By default, clients are not restricted to authenticate a specific set of servers when responding to REST requests. However, it is possible to explicitly list the servers that a client can respond to by using the `<managementRESTService>` element's `securityServiceLocation` attribute in the Ehcache configuration.

When this attribute is empty (or missing), no such restriction exists and the client will authenticate against any server in the cluster that meets the established security requirements. This is the recommended setting because SSL connections and the mechanism for authentication and authorization provide sufficient security.

In the case where an extra layer of security is required for the client's REST service, you can configure a list of allowed servers as follows:

```
<managementRESTService ...
  securityServiceLocation=" https://my-l2-node1/tmc/api/assertIdentity ,
     https://my-l2-node2/tmc/api/assertIdentity ">
```

where my-l2-node1 and my-l2-node2 are the servers' hostnames. However, any of the servers in a client's cluster can forward a REST request to that client at any time. Therefore, if this feature is used, all the servers should be listed.

8 Running a Secured Server

Introduction

Start a server in a secure Terracotta cluster using the start-tc-server script. If you are using encrypted keychains, a master password must be entered at the command line during server startup (or "set the server to automatically fetch the password" on page 55).

Confirming that Security is Enabled

You can confirm that a server's security is enabled in the following ways:

- Look for the startup message: "Security enabled, turning on SSL".
- Search for log messages containing "SSL keystore", "HTTPS Authentication enabled", and "Security enabled, turning on SSL".
- Attempt to make JMX connections to the server-these should fail.

Stopping a Secured Server

Stop a server in a secure Terracotta cluster using the stop-tc-server script with the following arguments:

- `-f <tc-config-file>` - A valid path to the self-signed certificate must have been specified in the server's configuration file.
- `-u <username>` - The user specified must have the "admin" role.
- `-w <password>`
- `-k` - This flag causes invalid TMS SSL certificates to be ignored. Use this option to accept self-signed certificates (ones not signed by a trusted CA).

Troubleshooting

You might encounter any of the following exceptions at startup:

TCRuntimeException: ... Wrong secret provided ?

The following exception indicates that the keychain file uses the default obfuscation scheme, but that the `-o` flag was not used with the keychain script:

```
com.tc.exception.TCRuntimeException:
com.terracotta.management.keychain.crypto.SecretMismatchException:
Wrong secret provided ?
```

Be sure to use the `-o` flag whenever using the keychain script.

No Configured SSL certificate

The following exception indicates that no SSL certificate was found for the server named "myServer":

```
Fatal Terracotta startup exception:
************************************************************************
Security is enabled but server myServer has no configured SSL certificate.
************************************************************************
```

Check that the expected SSL certificate was created for myServer and stored at the configured location.

IllegalStateException: Invalid cluster security configuration

This exception can occur when the security section in the Terracotta configuration file is not set up properly. However, this type of exception can also indicate problems elsewhere in the security setup. For example, an exception similar to the following can occur:

```
java.lang.IllegalStateException: Invalid cluster security configuration.
Unable to find connection credentials to server myOtherServer
```

This exception indicates that credentials cannot be found for the server named "myOtherServer". These credentials might be missing from or do not exist in the configured authentication source.

RuntimeException: Couldn't access a Console instance to fetch the password from!

This results from using "nohup" during startup. The startup process requires a console for reading password entry. You cannot run the startup process in the background if it requires manual password entry. For information on how to avoid having to manually enter the master keychain password, see "Reading the Keychain Master Password from a File" on page 55.

TCRuntimeException: Couldn't create KeyChain instance ...

The keychain file specified in the Terracotta configuration cannot be found. Check for the existence of the file at the location specified in `<keychain>/<url>` or the property `com.tc.security.keychain.url`.

RuntimeException: Couldn't read from file ...

This exception appears just after an incorrect password is entered for an "encrypted keychain file" on page 49.

RuntimeException: No password available in keyChain for ...

This exception appears if no keychain password entry is found for the server's certificate. You must explicitly "store the certificate password" on page 20 in the keychain file.

This exception can also appear if the resolved hostname or IP address is different from the one in the keychain entry:

- `tc://terracotta@localhost:9530` is the entry, but when the server configuration is read then `localhost` is resolved to an IP address. The entry searched for becomes `tc://terracotta@<a.certain.ip.address>:9530`.
- `tc://terracotta@<a.certain.ip.address>:9530` is the entry, but when the server configuration is read then <a.certain.ip.address> is resolved to a host name. The entry searched for becomes tc://terracotta@my.host.com:9530.

Two Active Servers (Split Brain)

Instead of an active-mirror 2-server stripe, both servers assert active status after being started. This exception can be caused by the failure of the SSL handshake. An entry similar to the following might appear in the server log:

```
2013-05-17 12:10:24,805 [L2_L2:TCWorkerComm # 1_W]
ERROR com.tc.net.core.TCConnection - SSL handshake error:
unable to find valid certification path to requested target, closing connection.
```

For each server, ensure that all keychain entries are accurate, and that the required certificates are available from the appropriate truststores.

No Messages Indicating Security Enabled

If servers start with no errors, but there are no messages indicating that security is enabled, ensure that the `<servers>` element contains `secure="true"` .

9 Using LDAP or Active Directory for Authentication

Introduction

Note: For a brief overview of Terracotta security with links to individual topics, see the "Overview of BigMemory Max Security" on page 5.

Instead of using the "built-in user management system" on page 13, you can set up authentication and authorization for the Terracotta Server Array (TSA) based on the Lightweight Directory Access Protocol (LDAP). This allows you to use your existing security infrastructure for controlling access to Terracotta clusters.

The two types of LDAP-based authentication supported are Microsoft Active Directory and standard LDAP. In addition, LDAPS (LDAP over SSL) is supported.

Note: Terracotta servers must be "configured to use SSL" on page 27 before any Active Directory or standard LDAP can be used.

Note: This topic assumes that the reader has knowledge of standard LDAP concepts and usage.

Configuration Overview

Active Directory and standard LDAP are configured in the <auth> section of each server's configuration block:

```
<servers secure="true">
  <server host="172.16.254.1" name="server1">
    ...
    <security>
      ...
      <auth>
        <realm>...</realm>
        <url>...</url>
        <user>...</user>
      </auth>
    </security>
...
</server>
```

Active Directory and standard LDAP are configured using the <realm> and <url> elements; the <user> element is used for "connections between Terracotta servers" on page 20 and is not required for LDAP-related configuration.

For presentation, the URLs used in this document use line breaks. Do not use line breaks when creating URLs in your configuration.

Realms and Roles

The setup for LDAP-based authentication and authorization uses Shiro realms to map user groups to one of the following two roles:

- admin - The user with the admin role is the initial user who sets up security. Thereafter, the user with the admin role performs system functions such as shutting down servers, clearing or deleting caches and cache managers, and reloading configuration.
- terracotta - The operator role is required to log in to the TMC, so even a user with the admin role must have the operator role. Thereafter, the person with the operator role can connect to the TMC and add connections.

URL Encoding

Certain characters used in the LDAP URL must be encoded, unless "wrapped in a CDATA construct" on page 48. Characters that may be required in an LDAP URL are described below:

- & (ampersand) - Encode as %26.
- { (left brace) - Encode as %7B.
- } (right brace) - Encode as %7D.
- Space - Encode as %20. *Spaces must always be encoded*, even if wrapped in CDATA.
- = (equals sign) - Does not require encoding.

Active Directory Configuration

Specify the realm and URL in the <security> section of the Terracotta configuration as follows:

```
<auth>
  <realm>com.tc.net.core.security.ShiroActiveDirectoryRealm</realm>
  <url>ldap://admin_user@server_address:server_port/searchBase=search_domain%26
       groupBindings=groups_to_roles</url>
  <user></user>
</auth>
```

Note the value of the <realm> element, which must specify the correct class (or Shiro security realm) for Active Directory. The components of the URL are defined in the following table.

Component	Description
ldap://	For the scheme, use either `ldap://` or `ldaps://`
admin_user	The name of a user with sufficient rights in Active Directory to perform a search in the domain specified by searchBase. The password for this user password must be stored in the Terracotta keychain used by the Terracotta server, using as key the root of the LDAP URI, `ldap://`

Component	Description
	`admin_user@server_name:server_port`, with no trailing slash ("/").
server_address: server_port	The IP address or resolvable fully qualified domain name of the server, and the port for Active Directory.
searchBase	Specifies the Active Directory domain to be searched. For example, if the Active Directory domain is `reggae.jamaica.org`, then the format is `searchBase=dc=reggae,dc=jamaica,dc=org`
groupBindings	Specifies the mappings between Active Directory groups and Terracotta roles. For example, `groupBindings=Domain%20Admins=admin,Users=terracotta` maps the Active Directory groups "Domain Admins" and "Users" to the "admin" and "terracotta" Terracotta roles, respectively. To be mapped, the named Active Directory groups must be part of the domain specified in searchBase; all other groups (including those with the specified names) in other domains are ignored.

For example:

```
<auth>
  <realm>com.tc.net.core.security.ShiroActiveDirectoryRealm</realm>
  <url>ldap://bmarley@172.16.254.1:389?searchBase=dc=reggae,dc=jamaica,dc=org%26
       groupBindings=Domain%20Admins=admin,Users=terracotta</url>
  <user></user>
</auth>
```

Standard LDAP Configuration

Specify the realm and URL in the <security> section of the Terracotta configuration as follows:

```
<auth>
  <realm>com.tc.net.core.security.ShiroLdapRealm</realm>
  <url>ldap://directory_manager@myLdapServer:636?
       userDnTemplate=cn=%7B0%7D,ou=users,dc=mycompany,dc=com%26
       groupDnTemplate=cn=%7B0%7D,ou=groups,dc=mycompany,dc=com%26
       groupAttribute=uniqueMember%26
       groupBindings=bandleaders=admin,bandmembers=terracotta</url>
  <user></user>
</auth>
```

Note the value of the <realm> element, which must specify the correct class (or Shiro security realm) for Active Directory. The components of the URL are defined in the following table.

Component	Description
ldap://	For the scheme, use either `ldap://` or `ldaps://`
directory_manager	The name of a user with sufficient rights on the LDAP server to perform searches. No user is required if anonymous lookups are allowed. If a user is required, the user's password must be stored in the Terracotta keychain, using as key the root of the LDAP URL, `ldap://admin_user@server_name:server_port`, with no trailing slash ("/").
server_address:server_port	The IP address or resolvable fully qualified domain name of the server, and the LDAP server port.
userDnTemplate	Specifies user-template values. See the example below.
groupDnTemplate	Specifies group-template values. See the example below.
groupAttribute	Specifies the LDAP group attribute whose value uniquely identifies a user. By default, this is "uniqueMember". See the example below.
groupBindings	Specifies the mappings between LDAP groups and Terracotta roles. For example, `groupBindings=bandleaders=admin, bandmembers=terracotta` maps the LDAP groups "bandleaders" and "bandmembers" to the "admin" and "terracotta" Terracotta roles, respectively.

For example:

```
<auth>
  <realm>com.tc.net.core.security.ShiroLdapRealm</realm>
  <url>ldap://dizzy@172.16.254.1:636?
       userDnTemplate=cn=%7B0%7D,ou=users,dc=mycompany,dc=com%26
       groupDnTemplate=cn=%7B0%7D,ou=groups,dc=mycompany,dc=com%26
       groupAttribute=uniqueMember%26
       groupBindings=bandleaders=admin,bandmembers=terracotta</url>
  <user></user>
</auth>
```

This implies the LDAP directory structure is set up similar to the following:

```
+ dc=com
    + dc=mycompany
        + ou=groups
            + cn=bandleaders
```

```
                  | uniqueMember=dizzy
                  | uniqueMember=duke
            + cn=bandleaders
                  | uniqueMember=art
                  | uniqueMember=bird
```

If, however, the the LDAP directory structure is set up similar to the following:

```
+ dc=com
    + dc=mycompany
        + ou=groups
            + cn=bandleaders
                  | musician=dizzy
                  | musician=duke
            + cn=bandleaders
                  | musician=art
                  | musician=bird
```

then the value of groupAttribute should be "musician".

Using the CDATA Construct

To avoid encoding the URL, wrap it in a CDATA construct as shown:

```
<url><![CDATA[ldap://dizzy@172.16.254.1:636?
     userDnTemplate=cn={0},ou=users,dc=mycompany,dc=com&
     groupDnTemplate=cn={0},ou=groups,dc=mycompany,dc=com&
     groupAttribute=uniqueMember&
     groupBindings=bandleaders=admin,bandmembers=terracotta]]></url>
```

10 Using Encrypted Keychains

Introduction

Note: For a brief overview of Terracotta security with links to individual topics, see "Overview of BigMemory Max Security" on page 5.

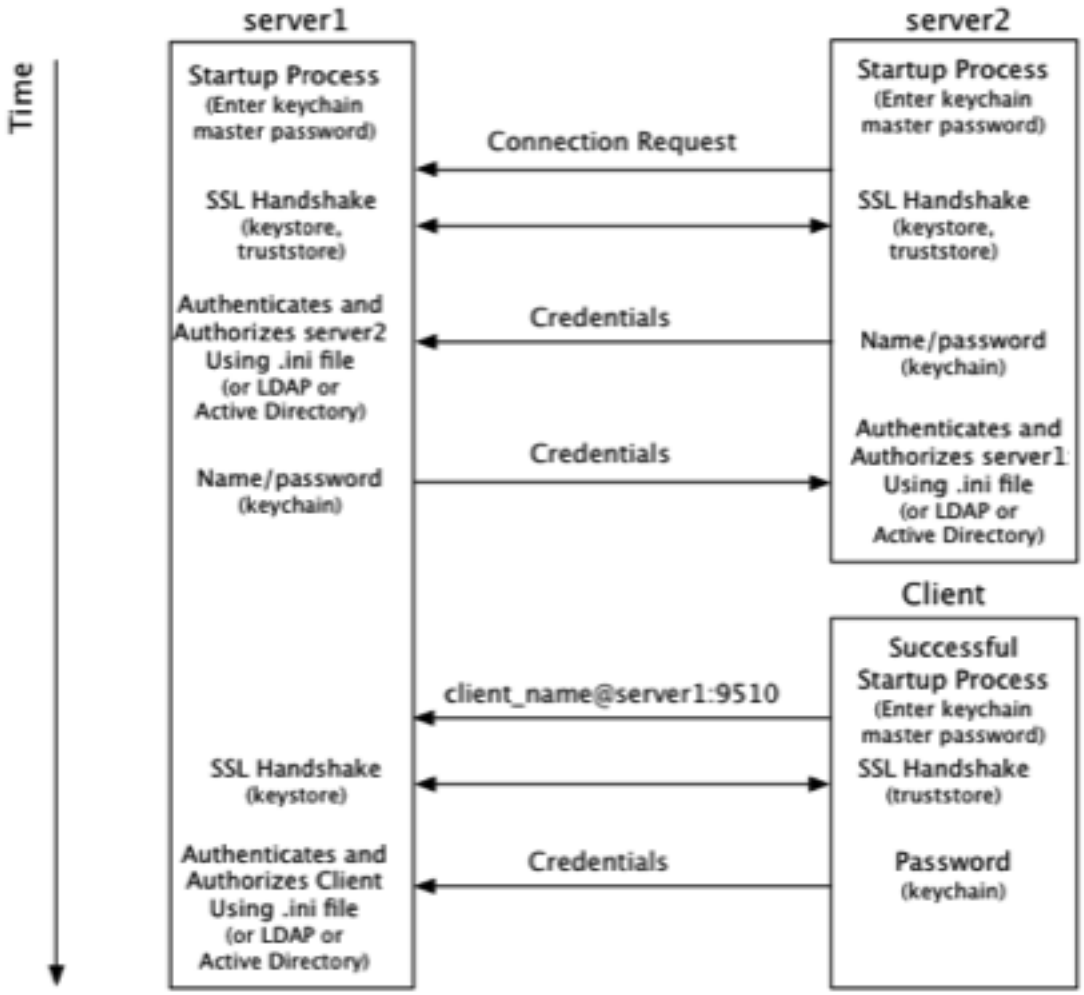
Security for the Terracotta Server Array (TSA) is set up using Terracotta configuration, tools provided in the Terracotta kit, standard Java tools, and public key infrastructure (via standard digital X.509 digital certificates). This setup process is described in "Setting Up Server Security" on page 17.

By default, the keychain script that creates Terracotta keychain files uses an obfuscation scheme to protect passwords. This scheme is adequate for development environments or environments where keychain-file security is already assured.

If your environment requires stronger protection for keychain files, use the encryption scheme described in this page. The encryption scheme requires a master password each time the keychain file is accessed.

Note: Except for the keychain setup, you must follow the setup instructions, including for authentication and SSL, as described in "Setting Up Server Security" on page 17.

The following diagram shows where the master password is required in the startup process of a Terracotta cluster.



From the point of view of the Terracotta *server*, security checks take place at startup and at the time a connection is made with another node on the cluster:

1. At startup, server1 requires a password to be entered directly from the console to complete its startup process. The password can also be "read from a file" on page 55 to avoid manual entry.
2. A connection request from server2 initiates the process of establishing a secure connection using SSL.
3. Server1 authenticates server2 using stored credentials. Credentials are also associated with a role that authorizes server2. The process is symmetrical: server2 authenticates and authorizes server1.
4. A connection request from a Terracotta client initiates the process of establishing a secure connection using SSL.
5. Server1 authenticates and authorizes the client using stored credentials and associated roles. Because a client might communicate with any active server in the cluster during its lifetime, the client must be able to authenticate with any active

server. Because active servers can fail over to mirror servers, each client should be able to authenticate against *all* servers in the cluster.

From the point of view of a Terracotta *client*, security checks occur at the time the client attempts to connect to an active server in the cluster:

1. The client uses a server URI that includes the client username.

   A typical (non-secure) URI is `<server-address>:<port>`. A URI that initiates a secure connection takes the form `<client-username>@<server-address>:<port>`.

2. A secure connection using SSL is established with the server.
3. The client sends a password fetched from a local keychain file. The password is associated with the client username.

Note that the diagram and process shown above are similar to those found in "Setting Up Server Security" on page 17. The main differences, described in this document, concern the use of the keychain file.

Configuration Example

The following configuration snippet is an example of how security could be set up for the servers in the illustration above:

```
<tc:tc-config xmlns:tc="http://www.terracotta.org/config">
...
  <servers secure="true">
    <server host="172.16.254.1" name="server1">
    ...
      <security>
        <ssl>
          <certificate>jks:server1alias@/the/path/keystore-file.jks</certificate>
        </ssl>
        <keychain>
          <secret-provider>
            com.terracotta.management.security.ConsoleFetchingBackend
          </secret-provider>
          <url>file:///%(user.dir)/server1keychain.tkc</url>
        </keychain>
        <auth>
          <realm>com.tc.net.core.security.ShiroIniRealm</realm>
          <url>file:///%(user.dir)/myShiroFile.ini</url>
          <user>server1username</user>
        </auth>
      </security>
   ...
   </server>
    <server host="172.16.254.2" name="server2">
    ...
      <security>
        <ssl>
          <certificate>jks:server2alias@/the/path/keystore-file.jks</certificate>
        </ssl>
        <keychain>
          <url>file:///%(user.dir)/server2keychain.tkc</url>
        </keychain>
        <auth>
```

```
          <realm>com.tc.net.core.security.ShiroIniRealm</realm>
          <url>file:///%(user.dir)/myShiroFile.ini</url>
          <user>server2username</user>
        </auth>
      </security>
   ...
   </server>
   ...
 </servers>
...
</tc:tc-config>
```

See the "configuration section" on page 24 for more information on the configuration elements in the example.

Note: Script names in the examples given below are for UNIX and Linux systems. Equivalent scripts are available for Microsoft Windows in the same locations. Replace the .sh extension with .bat and convert path delimiters as appropriate.

Configuring the Encrypted Server Keychain

By default, keychain files protect stored passwords using an obfuscation scheme. You can override this scheme by explicitly naming the secret provider for encryption:

```
<secret-provider>
com.terracotta.management.security.ConsoleFetchingBackend
</secret-provider>
```

This secret provider is also shown in the configuration example above.

Tip: Overriding the Configured Secret Provider - You can override the configured secret provider using the property `com.terracotta.SecretProvider` . For example, to use obfuscation without changing configuration, use

```
com.terracotta.SecretProvider=
  com.terracotta.management.security.ObfuscatingSecretProviderBackend
```

Adding Entries to Encrypted Keychain Files

You must also add entries to the keychain file as described in "Setting up the Server Keychain" on page 20, *but avoid using the* `-o` *flag when using the keychain script.*

For example, to create an entry for the local server's keystore password, use:

```
tools/security/bin/keychain.sh <keychain-file> <certificate-URI>
```

If the keychain file does not exist, add the `-c` option to create it:

```
tools/security/bin/keychain.sh -c <keychain-file> <certificate-URI>
```

You will be prompted for the keychain file's master password, then for a password to associate with the URI. *For the URI, you must enter the same password used to secure the server's certificate in the keystore.*

For example, to create an entry for server1 from the configuration example above, enter:

```
tools/security/bin/keychain.sh server1keychain.tkc jks:server1alias@keystore-file.jks
Terracotta Management Console - Keychain Client
Open the keychain by entering its master key:   xxxxxxx
Enter the password you wish to associate with this URL:   server1pass
Confirm the password to associate with this URL:   server1pass
Password for jks:server1alias@keystore-file.jks successfully stored
```

To create an entry for server2 in server1's keychain, use:

```
tools/security/bin/keychain.sh server1keychain.tkc
tc://server2username@172.16.254.2:9530
```

Configuring the Encrypted Client Keychain Files

For clients, set the secret provider with the following property:

```
com.terracotta.express.SecretProvider=
net.sf.ehcache.terracotta.security.ConsoleFetchingSecretProvider
```

Add entries to the keychain file as described in "Setting up the Server Keychain" on page 20, *but avoid using the* `-o` *flag when using the keychain script.*

For example:

```
tools/security/bin/keychain.sh clientKeychainFile tc://client1@172.16.254.1:9510
```

When you run the keychain script, the following prompt should appear:

```
Terracotta Management Console - Keychain Client
KeyChain file successfully created in clientKeychainFile
Open the keychain by entering its master key:
```

Enter the master key, then answer the prompts for the secret to be associated with the server URI:

```
Enter the password you wish to associate with this URL:
Password for tc://client1@172.16.254.1:9510 successfully stored
```

Note that the script does not verify the credentials or the server address.

If the keychain file does not already exist, use the `-c` flag to create it:

```
tools/security/bin/keychain.sh -c clientKeychainFile tc://client1@172.16.254.1:9510
```

If creating the keychain file, you will be prompted for a master password. To automate the entry of the master password, see "Clients Automatically Reading the Keychain Password" on page 57.

The Terracotta client searches for the keychain file in the following locations:

- %(user.home)/.tc/mgmt/keychain
- %(user.dir)/keychain.tkc
- The path specified by the system property `com.tc.security.keychain.url`

Securing with the TMS

If you are using the Terracotta Management Server (TMS), you must set up "JMX authentication" on page 34. Every node in the cluster must have the following entry in its keychain, all locked with the identical secret:

```
jmx:net.sf.ehcache:type=RepositoryService
```

In addition, server-server REST-agent communication must also be authorized using a keychain entry using the format `jmx://<user>@<host>:<group-port>`.

Add entries to the keychain file as described in "Setting up the Server Keychain" on page 20, *but avoid using the* `-o` *flag when using the keychain script.*

For example, to create an entry for server2 in server1's keychain, use:

```
tools/security/bin/keychain.sh server1keychain.tkc
jmx://server2username@172.16.254.2:9530
```

Each server must have an entry for itself and one for each other server in the TSA.

Reading the Keychain Master Password from a File

Instead of manually entering the master keychain password at startup, you can set servers and clients to automatically read the password.

Note: Cygwin (on Windows) is not supported for this feature.

Servers Automatically Reading the Keychain Password

1. Implement the interface com.terracotta.management.security.SecretProviderBackEnd (located in the JAR com.terracotta:security-keychain) to fetch a password from a given file. For example:

```
package com.foo;
import com.terracotta.management.security.SecretProviderBackEnd;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
public class MySecretProvider implements SecretProviderBackEnd {
  private byte[] bytes;
  // This method reads the password into a byte array.
  @Override
  public void fetchSecret() {
    try {
      bytes = readPasswordFile("password.pw");
    } catch (IOException ioe) {
      throw new RuntimeException("Cannot read password from file", ioe);
    }
  }
  private byte[] readPasswordFile(String filename) throws IOException {
    FileInputStream fis = new FileInputStream(filename);
    try {
      byte[] buffer = new byte[64];
```

```
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      while (true) {
        int read = fis.read(buffer);
        if (read == -1) {
          break;
        }
        baos.write(buffer, 0, read);
      }
      return baos.toByteArray();
    } finally {
      fis.close();
    }
  }
  // This method returns the byte array containing the password.
  @Override
  public byte[] getSecret() {
    return bytes;
  }
}
```

2. Create a JAR containing your implementation (MySecretProvider), then copy it to the BigMemory Max server/lib directory.
3. Assuming the new JAR file is called my-secret-provider.jar, edit the start-tc-server script in the BigMemory Max server/bin as follows:

   UNIX/LINUX

   Change the line

   ```
   -cp "${TC_INSTALL_DIR}/lib/tc.jar" \
   ```

   to

   ```
   -cp "${TC_INSTALL_DIR}/lib/tc.jarr:${TC_INSTALL_DIR}/lib/my-secret-provider.jar" \
   ```

   MICROSOFT WINDOWS

   Change the line

   ```
   set CLASSPATH=%TC_INSTALL_DIR%\lib\tc.jar
   ```

   to

   ```
   set CLASSPATH=%TC_INSTALL_DIR%\lib\tc.jar;%TC_INSTALL_DIR%\lib\my-secret-provider.jar
   ```

4. Ensure that the server's configuration includes the <secret-provider> element specifying your implementation:

   ```
   <security>
     ...
     <keychain>
       <url>/path/to/my/keychain</url>
       <secret-provider>com.foo.MySecretProvider</secret-provider>
     </keychain>
     ...
   </security>
   ```

At startup, the server will read the keychain password from the file specified in your implementation.

For a simpler solution, you could instead hardcode the password:

```
package com.foo;
import com.terracotta.management.security.SecretProviderBackEnd;
```

```
public class MySecretProvider implements SecretProviderBackEnd {
  // This method returns the byte array containing the password.
  @Override
  public byte[] getSecret() {
    return new byte[] {'p', 'a', 's', 's', 'w', 'o', 'r', 'd'};
  }
  @Override
  public void fetchSecret() {
  }
}
```

Clients Automatically Reading the Keychain Password

You can set up Terracotta clients to read their keychain's master password in a similar way as for servers. Import org.terracotta.toolkit.SecretProvider and override fetchSecret() and getSecret() as shown above.

Instead of packaging the implementation in a JAR, specify your implementing class by using the system property `com.terracotta.express.SecretProvider`.