GemStone®

GemStone/S 64 Bit™ X509-Secured GemStone Adminstration Guide

Version 3.5

June 2019

GEMTALK™ SYSTEMS

GemTalk Systems LLC
15220 NW Greenbrier Parkway
Suite 240
Beaverton, OR 97006

Preface

About This Documentation

This manual describes the features that support GemStone’s X509-Secured Environment. It is intended for users who are familiar with administration of a GemStone/S 64 Bit™ environment, and provides the additional information needed to administer an X509-Secured environment.

Refer to the *System Administration Guide for GemStone/S 64 Bit* for fundamental details on administering a GemStone environment.

The X509 features is licenced in addition to the GemStone/S 64 Bit product, and requires a keyfile that specifically enables the X509 features.

Technical Support

Support Website

gemtalksystems.com

GemTalk’s website provides a variety of resources to help you use GemTalk products:

- **Documentation** for the current and for previous released versions of all GemTalk products, in PDF form.
- **Product download** for the current and selected recent versions of GemTalk software.
- **Bugnotes**, identifying performance issues or error conditions that you may encounter when using a GemTalk product.
- **Supplemental Documentation** and **TechTips**, providing information and instructions that are not in the regular documentation.
- **Compatibility matrices**, listing supported platforms for GemTalk product versions.

We recommend checking this site on a regular basis for the latest updates.

Help Requests

GemTalk Technical Support is limited to customers with current support contracts. Requests for technical assistance may be submitted online (including by email), or by telephone. We recommend you use telephone contact only for urgent requests that require immediate evaluation, such as a production system down. The support website is the preferred way to contact Technical Support.

Website: techsupport.gemtalksystems.com

Email: techsupport@gemtalksystems.com

Telephone: (800) 243-4772 or (503) 766-4702

Please include the following, in addition to a description of the issue:

- The versions of GemStone/S 64 Bit and of all related GemTalk products, and of any other related products, such as client Smalltalk products, and the operating system and version you are using.
- Exact error message received, if any, including log files and statmonitor data if appropriate.

Technical Support is available from 8am to 5pm Pacific Time, Monday through Friday, excluding GemTalk holidays.

24x7 Emergency Technical Support

GemTalk offers, at an additional charge, 24x7 emergency technical support. This support entitles customers to contact us 24 hours a day, 7 days a week, 365 days a year, for issues impacting a production system. For more details, contact GemTalk Support Renewals.

Training and Consulting

GemTalk Professional Services provide consulting to help you succeed with GemStone products. Training for GemStone/S is available at your location, and training courses are offered periodically at our offices in Beaverton, Oregon. Contact GemTalk Professional Services for more details or to obtain consulting services.

Table of Contents

Chapter

1

Introduction to X.509-based Security Features

While GemStone/S 64 Bit has a number of security features, the GemStone server is normally run within a secured environment that is not at risk for malicious attack. Starting with v3.5, GemStone/S 64 Bit also includes support for highly secured remote logins, in which each connection to a process on the Stone's node is authenticated and the communication is protected by X509 certificates.

The security features in X509-Secured GemStone allows user applications in the cloud, for example, to login to a Stone on a secure node without any risk of compromising the Stone or any sensitive data.

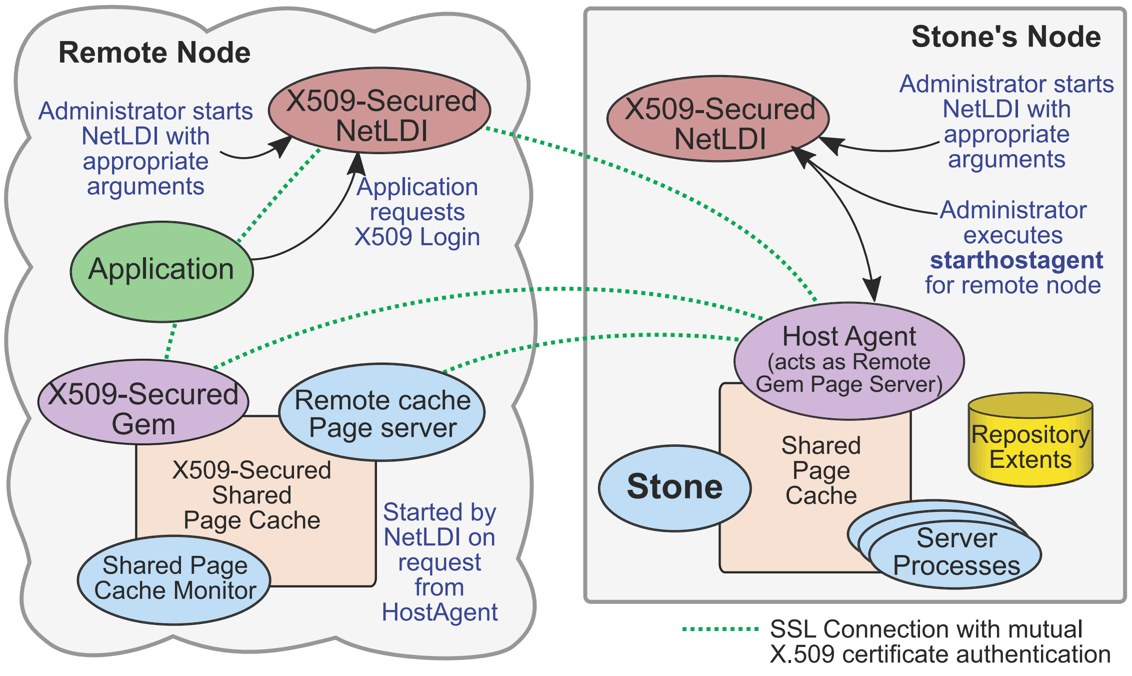
GemStone/S 64 Bit also includes support for generating X509 Certificates, including self-signed Certificate Authority Certificates that allow the use of X509-Secure GemStone in a test configuration. For security, applications should generate certificate-requests and submit these to a external certificate authority. Correct security procedures must be followed in order to create a secure application.

The X509-secured GemStone feature is separately licensed; you must have a keyfile with appropriate permissions in order to use X509-secured logins.

1.1 Overview

X509 logins and associated processes, such as X509-secured NetLDIs and X509-secured remote shared page caches, use a different architectural model than ordinary GemStone logins.

Figure 1.1 Overview of components in X509-Secure GemStone


Remote Node
Administrator starts NetLDI with appropriate arguments
X509-Secured NetLDI
Application requests X509 Login
Application
X509-Secured Gem
Remote cache Page server
X509-Secured Shared Page Cache
Shared Page Cache Monitor
Started by NetLDI on request from HostAgent
Stone's Node
X509-Secured NetLDI
Administrator starts NetLDI with appropriate arguments
Administrator executes starthostagent for remote node
Host Agent (acts as Remote Gem Page Server)
Shared Page Cache
Repository Extents
Stone
Server Processes
SSL Connection with mutual X.509 certificate authentication

Both kinds of logins are supported to a single Stone. However, the supporting processes such as the NetLDI and remote shared page cache are not shared between these two kinds of logins, which places some restrictions on the topography for configurations that will use both regular and X509-secured logins.

- **X509-Secured NetLDIs**
  To mediate the secure login, NetLDIs must be started in certificate-only mode on both the remote Gem host and the Stone's node, configured with the appropriate certificates. X509-Secured NetLDIs only support X509-Secured login, and X509-Secured logins require X509-Secured NetLDIs; you may run both X509-Secured and regular NetLDIs on the same node.

- **New Host Agent process**
  Each remote node that will hold an X509-Secured process needs to have a Host Agent running on the Stone's node. An additional script, **starthostagent**, must be executed on the Stone's node after both NetLDI are started, and before logging in. **starthostagent** executes for a single specific remote node, and requires the appropriate certificates.

- **Startup of the remote cache**
  The act of starting up a remote cache is initiated when the (**starthostagent**) script is executed for that remote node.

  When instructed by the newly started HostAgent, the remote shared page cache that supports the secure login is started up by the remote NetLDI. The remote cache is configured based on specific new arguments to **startnetldi** that were passed in when the remote NetLDI was started.

- **Login Parameters**
  Both secured and ordinary logins can be done to the same Stone at the same time, but they must use different NetLDIs. Secure logins are performed using new C calls that require a different set of login parameters than ordinary logins. Specific **set** commands in topaz, new GCI functions, and GemBuilder for Smalltalk parameter editor and classes added in GBS v8.4 support X509-secured logins.

  X509-Secured login only can be done for RPC logins. Linked logins do not use certificates.

- **Connections initiated from Stone's node**
  All interprocess connections are initiated from the Stone's node. So, for example, when an application logs in, it provides a listening socket to the HostAgent on the Stone's node, but the secure connection is initiated by the HostAgent.

  This also means that if the X509-Secured remote cache times out and shuts down (which occurs after a timeout, when no sessions are logged in and using that remote cache), which shuts down the Host Agent on the Stone's node, you must execute **starthostagent** again on the Stone's node before further X509-Secure logins are possible on that remote node.

- **Mid-level caches must also be X509-Secured**
  If a remote node is to be used as a Mid-Level cache by X509-Secured Gems, it must also be started in certificate-only mode with specific configuration values that give it the ability to act as a mid-level cache.

- **Object Filters**
  Additional object-level security is provided in an X509-Secured GemStone, which allows you to define object access security by IP address. This allows applications to avoid the risk that sensitive data may be present in a remote cache.

There are no differences in the way the Stone is configured and started up, although some configuration parameters (such as those managing the use of SSL) are not relevant for X509-secured sessions.

Chapter

2 X.509 Certificates

For an X509 login, X.509 certificates are required for authentication for all interprocess connections between any remote process and the Stone. These certificates are passed in as arguments on the command line, or as login parameters.

Different kinds of certificates are required.

- The Stone CA certificate, which acts as a trust anchor. This can be a self-signed certificate or an certificate signed by an external certificate authority.
- Host and User CA certificates, which are used to create chained certificates for the Host and User certificates
- Host certificates and private keys, which authenticate a remote node to allow remote processes to authenticate with the Stone during startup. Host certificates are named, but the name is optionally restricted to a specific host or set of hosts by an additional CIDR-based address restriction.
- User certificates and private keys, which are used to authenticate logins as specific users. When performing an X509 login, the login parameters do not specify a username; the user name is determined from the certificate. The certification also controls the name of the Stone, so the login parameters also do not include the Stone name.
- A (CRLs) certificate revocation list. This allows certificates to be reliably revoked. For more information on CRLs, see the section starting on page 18.

The GemStone distribution includes scripts that generate the required certificates. Alternatively, you can create certificates using any tools you wish, provided that the required information is included.

This chapter describes the process of creating certificates using the GemStone scripts.

2.1 Utilities to create Certificates

GemStone's utilities to create certificates require OpenSSL, but otherwise do not require GemStone or a running Stone. You may use the OpenSSL that is distributed with the GemStone server, or your own OpenSSL installation.

Specific utilities include:

```
newstone
newstone_csr
newhostCA
newhost
newuserCA
newuser
```

Trust Anchor Certificate Authority

To create a self-signed stone CA, use the `newstone` script. If you want an external CA to produce the CA, use the script `newstone_csr`, which generates a certificate signing request (CSR) which can be used by the external CA to create the stone CA certificate.

While this externally generated stone CA certificate is not self-signed, it is accepted as a trust anchor by GemStone logins, the same way a self-signed stone CA is used.

Either the `newstone` or the `newstone_csr` script must be run before the other scripts, since all other scripts are specific to a particular Stone name (and require the -s argument to specify that Stone name).

User and Host Certificates

Each Stone requires certificates for each user and each host, which are specific to that stone.

Before creating user or host certificates, you must generate user and host CA certificates, using the scripts `newuserCA` and `newhostCA`. This allows separate control over certificate validity for these different types of entities.

The `newuser` and `newhost` scripts then create a chained certificate that is used for authentication.

Certificate Utilities

The GemStone scripts to create X509 certificates for the stone, hosts, and users, are located in the directory `$GEMSTONE/bin/x509`. It is not required to use these scripts to create certificates, provided the necessary information is included.

These scripts require the following environment variables to be set.

- $GEMSTONE or $OPENSSL_PREFIX_DIR
  This specifies where to find the OpenSSL executable. $GEMSTONE indicates to use the OpenSSL executable in the GemStone distribution; $OPENSSL_PREFIX_DIR is used to specify your own custom OpenSSL executables. If both variables are defined then $OPENSSL_PREFIX_DIR is used.
- $GEMSTONE_CERT_DIR
  A writable directory that will be the root directory for the hierarchical directory structure holding certificates and keys that are generated for all stones, users and hosts.

There is an additional, optional environment variable $GEMSTONE_CERT_DEBUG. If this is set, then debugging information is printed to stdout.

The certificate creation scripts include:

`newstone -h | [-d` *daysValid*`]` *stoneName*
: create self-signed certificates for the stone with the given name.

`newstone_csr -h |` *stoneName*
: create a certificate-signing-request for the stone with the given name.

`newhostCA -h | [-d` *daysValid*`] -s` *stoneName*
: Create a CA certificate for the hosts associated with the Stone named *stoneName*. The Stone's certificates must already exist.

`newhost -h | [-d` *daysValid*`] [-a` *addr_restriction*`] -s` *stoneName certName*
: Create certificates for the host or hosts described by *hostName*, specifically associated with the Stone named *stoneName*. The Stone's certificates must already exist and the host CA certificate must already exist. By default, the certs named *certName* can be used on any host to authenticate with *stoneName*; to restrict the host certs, you must use the **-a** argument. The **-a** *addr_restriction* must be in CIDR notation and will limit the IP address on which this certificate can be used to the specified range. If this argument is omitted, there is no restriction (0.0.0.0/0).

`newuserCA -h | [-d` *daysValid*`] -s` *stoneName*
: Create a CA certificate for the user *userName*, specifically associated with the Stone named *stoneName*. The Stone's certificates must already exist.

`newuser -h | [-d` *daysValid*`] [-a` *addr_restriction*`] -s` *stoneName userName*
: Create certificates for the GemStone user *userName*, allowing login to a stone named *stoneName*. The Stone's certificates must already exist and the user CA certificate must already exist. *addr_restriction* must be in CIDR notation and will limit the IP address from which users can login (note that this limits the nodes for both the client application and the Gem). If this argument is omitted, there is no restriction (0.0.0.0/0).

The `newstone` script must be run before the other scripts. They can be run on any node, they do not need to be run on the specific host.

If the certificates are created correctly, the command returns status code 0, and prints no results. If an error occurs, the details are displayed.

Limiting the period for which a certificate is valid

Certificates generated by GemStone scripts are by default valid for 30 days. You can specify the period of validity for certificates and CA certificates by using the **-d** argument to any of the scripts. For a login to succeed, all chained certs and CA certs must be valid.

A day is a 24-hour period starting from the time the certificate is created. You may examine the validity period of a pem file, along with other details, using:

```
unix> openssl x509 -in certfilename.pem -text
```

Limiting the IP addresses for Hosts and Users

You may also further control access by specifying the subnet for a certificate, so that only nodes on that subnet may use that certificate. These authorizations are embedded in the certificates, and are configured using an argument to the `newhost` and `newuser` scripts.

Address restrictions are specified using subnet masking with CIDR (Classless Inter-Domain Routing) notation. This notation includes a final /N with the number of bits to mask in the IP address. So for example, 0.0.0.0/0 indicates no restriction, 10.94.141.45/32 limits to a single IP address, and 10.94.141.0/24 limits to any host in the subnet 10.94.141.x.

Restricting nodes on which a host certificate can be used

By default, host certificates can be used to start an X509-secured NetLDI on any node. You can restrict the certificates so that they are only valid for a single node or for a subnet of nodes, using the **-a** argument to the `newhost` script. For example:

```
unix> $GEMSTONE/bin/x509/newhost -s gs64stone
   -a "10.94.141.0/24" stn_host
```

The resulting certificate and private key can be used on any node that has an IP address within the subnet 10.94.141.x.

Restricting nodes on which Gem or Application can run

Likewise, you can limit the nodes from which a particular user can login by creating the user certificates using the `newuser` **-a** argument.

Note that the user certificate subnet limits the nodes for the application client, as well as the Gem host node, if the application client and Gem host are different nodes.

When the user certificate is restricted to a limited address range, the login parameters may still specify the NetLDI using localhost or other 127.x.x.x addresses (loopback addresses, and therefore always the same host on both ends of the connection). The remote NetLDI will not apply the subnet restriction to the loopback address, and will fork the Gem. The remote Gem uses the same user certificate to authenticate with the HostAgent on the Stone's node, using the IP address of the node it is on, that is, the remote Gem's node. As long as the user certificate address restriction allows the IP address of the Gem's node, this connection will be allowed.

Example certificate creation

If you have a stone named ***gs64stone*** that will be running on the node named ***stn_host***, and will be starting the remote cache on another node named ***remote_host***, the minimal set of certificates to login as DataCurator can be created using the following:

```
unix> setenv GEMSTONE_CERT_DIR $GEMSTONE/data
unix> $GEMSTONE/bin/x509/newstone gs64stone
unix> $GEMSTONE/bin/x509/newhostCA -s gs64stone
unix> $GEMSTONE/bin/x509/newhost -s gs64stone stn_host
unix> $GEMSTONE/bin/x509/newhost -s gs64stone remote_host
unix> $GEMSTONE/bin/x509/newuserCA -s gs64stone
unix> $GEMSTONE/bin/x509/newuser -s gs64stone DataCurator
```

These operations create the certificates and private keys and empty CRL (certificate revocation list) that are needed for both the remote NetLDI, the Stone's NetLDI, and the X509 login.

2.2 Examine and delete certificates

A hierarchy of certificates and other supporting files is created within the directory specified by $GEMSTONE_CERT_DIR. Once the certificates are generated by the scripts, there is no specific meaning to location within the hierarchy, and you may move these files elsewhere, or use them where they are via the full path within the hierarchy.

Certificate directory structure

The scripts create a directory structure under $GEMSTONE_CERT_DIR, holding the various certificate files that are needed as well as other OpenSSL generated files. This structure is:

```
$GEMSTONE_CERT_DIR
   stones
      aStoneName
         stoneCA
            stoneCA-aStoneName.cert.pem
            stoneCA-aStoneName.privkey.pem
            hostCA-userCA-combined-aStoneName.crl.pem
            <other file/s>
         hosts
            aHostname
               aHostname.chain.pem
               aHostname.privkey.pem
               <other file/s>
         users
            aUserName
               aUserName.chain.pem
               aUserName.privkey.pem
               <other file/s>
```

You may copy the files from the given locations or use them in place, or rename them as needed.

For clarity, the instructions recommend putting the files into a new directory and defining another environment variable, to avoid long directory paths, and to make commands more readable.

Duplicates and Deleting certificates

You may not create a certificate if a certificate for the given stone, host or user already exists.

You can delete an existing certificate, or all certificates, manually from the directories. This keeps the certificate directories less cluttered, but to ensure that a certificate was not copied and cannot be used, you will need to revoke the certificate.

The following scripts are provided for convenience in reporting and removing unnecessary certificate flies:

`lsstone -h |` *stoneName*
: List the Stone names for which certificates have been created.

`lshost -h | -s` *stoneName*
: List the host names for which certificates have been created for the given Stone.

`lsuser -h | -s` *stoneName*
: List the user names for which certificates have been created for the given Stone.

`rmstone -h | [-f]` *stoneName*
: Remove the certificates for the stone with the given name. This will prompt for confirmation; use the -f option to force remove without confirmation.

`rmhost -h | [-f] -s` *stoneName hostName*
: Remove the certificates for the host *hostName* associated with the given Stone. This will prompt for confirmation; use the -f option to force remove without confirmation.

`rmuser -h | [-f] -s` *stoneName userName*
: Remove the certificates for the user *userName* associated with the given Stone. This will prompt for confirmation; use the -f option to force remove without confirmation.

2.3 Certificate revocation list

A CRL (Certificate revocation list) is a list of digital certificates that have been revoked by the CA (certificate issuing certificate authority), and should no longer be trusted. CRLs are PEM files and like certificates, multiple CRLs can be combined in a single CRL file.

Only leaf certificates (certificates for users and hosts) can be revoked.

A CRL is now required by startnetldi for an x509-secured NetLDI on the Stone's node; this may initially be an empty CRL. An empty CRL file contains no revoked certs, but is a PEM file signed by the CA; this is necessary to ensure that the CRL is genuine.

The GemStone x509 certificate creation scripts now will automatically create an empty CRL when the host and user CAs are created. These are automatically combined into a single PEM file with the name

`hostCA-userCA-combined-`*stoneName*`.crl.pem`

When a host or user certificate is revoked, use the new scripts **revokehost** or **revokeuser** to revoke the cert; you must then restart the NetLDI on the Stone's node, passing in the updated CRL file.

You will also need to manually stop the HostAgent supporting the revoked remote host, and any logged in sessions for the revoked user.

scripts to revoke host and user

The following scripts have been added, to allow you to revoke a host or user certificate.

`revokehost -h | -s` *stoneName hostName*
: revoke a host certificate.

`revokeuser -h | -s` *stoneName userName*
: revoke a user certificate

Revoking a cert causes the cert and key to be moved into the revoked directory. There, a new subdirectory is created with a the name of the host or user, and a timestamp in the directory name.

For example:

```
unix> $GEMSTONE/bin/x509/revokehost -s gs64stone lark
[Info]: Certificate /certs/stones/gs64stone/hosts/lark/lark.ce
rt.pem has been revoked.
[Info]: Combined Certificate Revocation List (CRL) /certs/stone
s/gs64stone/stoneCA/hostCA-userCA-combined-gs64stone.crl.pem
has been updated.
[Info]: Please distribute the new CRL to all appropriate
locations.
```

After this, the directory `/certs/stones/gs64stone/hosts/lark/` has been moved to `/certs/stones/gs64stone/hosts/revoked/lark.revokedOn.May-22-2018-17-23-14/`.

Applying updated CRL

Revoking a cert updates the file `hostCA-userCA-combined-`*stoneName*`.crl.pem`.

1. This updated CRL file should get copied into the directory in which your certs are located, overwriting the existing one.
2. Stop the Stone's NetLDI and restart it using the -L option, passing in the updated CRL.
3. You must stop and restart the HostAgent process for the revoked host, or all HostAgents, by killing the process or using methods such as:

   ```
   System stopHostAgents
   System stopHostAgentSession:
   ```

   Note that as of this alpha version, the HostAgents must be manually restarted using the **starthostagent** script.

Login failures related to revoked certificates will be logged in the HostAgent log file.

Chapter 3 Getting Connected

X509 logins require a number of steps to ensure the supporting processes are setup with the appropriate certificates.

This chapter is organized to provide a summary of the instructions, followed by examples. The examples describe a Stone named ***devstone*** running on a node named ***alcatraz***, the Stone's node. The remote x509 Gem is running on ***fiji***.

GemStone NetLDIs can be named, or accessed via port numbers. Named NetLDIs must be setup in the hosts NIS for each node. For simplicity, the following examples use a port number, 54321.

3.1 Setup and Login

The following instructions will help you setup the Stone and NetLDI and login an X509-secure session.

It is assumed you have generated scripts has described in the previous chapter. You may use certificates that were generated using GemStone scripts, or certificates generated in other ways provided they are correctly composed.

1. Configuring the Stone and the remote node

To begin with, you should have a GemStone/S 64 Bit installation on both the Stone's node and on the remote cache node, and you should have setup and started a Stone process.

Before starting the process of setting up a X509 login, you might want to verify that your system is configured so you can perform a remote, non-X509 login between the remote and Stone's node.

Since unsecured NetLDIs do not support X509 logins, and vice versa, you must either use a different port to start the X509-secured NetLDI on the Stone and remote nodes, or shut down the unsecured NetLDIs and restart using the process described in this chapter.

2. Setup script and log directories

X509 logins require a number of certificates to be available; some are needed on the Stone's node, some on the remote node, and some on both nodes.

While there are a number of ways to manage the certificates, the simplest is to create new flat directories in which to place the required certificate files, and access these via an environment variable. These examples use $MyStoneCertDir and $MyRemoteCertDir.

On the Stone's Node

Create a directory on the Stone's node, and define an environment variable, for example `$MyStoneCertDir`, that points to the directory. Then copy the required certificate files to that directory.

Using the example certificates created in the previous step, on the Stone's node, you would perform the following copy operations:

```
export MyStoneCertDir=dirName

cp $GEMSTONE_CERT_DIR/stones/devstone/stoneCA/stoneCA-
   devstone.cert.pem $MyStoneCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/stoneCA/hostCA-userCA-
   combined-devstone.crl.pem $MyStoneCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/alcatraz/alcatraz.chain
   .pem $MyStoneCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/alcatraz/alcatraz.privk
   ey.pem $MyStoneCertDir
```

On the Remote Node

Likewise, create a directory on the remote node, and define an environment variable there to point to this directory, `$MyRemoteCertDir`. Copy the following files to that directory:

```
export MyRemoteCertDir=dirName

cp $GEMSTONE_CERT_DIR/stones/devstone/stoneCA/stoneCA-
   devstone.cert.pem $MyRemoteCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/fiji/fiji.chain.pem
   $MyRemoteCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/fiji/fiji.privkey.pem
   $MyRemoteCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/users/DataCurator/DataCur
   ator.chain.pem $MyRemoteCertDir

cp $GEMSTONE_CERT_DIR/stones/devstone/users/DataCurator/DataCur
   ator.privkey.pem $MyRemoteCertDir
```

Setup log directories

There are a few differences in how X509 process handle child process log files.

With X509-secured NetLDIs, the startnetldi -D argument, which specifies a directory for child process logs, is required. Since the -D argument overrides the default use of the home directory, this means that gem log files do not get written to the home directory

To make log management easier, it is recommended to create log file directories for the Stone's node and the remote node, and direct all related logs to that directory.

These examples use the environment variables **`$remoteLogDir`** and **`$stoneLogDir`** to refer to these directories.

For example:

```
fiji> mkdir $GEMSTONE/logs
fiji> export remoteLogDir=$GEMSTONE/logs

alcatraz> mkdir $GEMSTONE/logs
alcatraz> export stoneLogDir=$GEMSTONE/logs
```

3. Start certificate-only NetLDI on the Stone's Node

The Stone must be running before you start the x509-secured NetLDI.

On the Stone's node, you must start the NetLDI in certificate-only mode, using the arguments **-S -U -R -J** and **-L** in addition to any other **startnetldi** arguments. The **-D** argument is also required, to set the directory for log files. Do not use the **-E** argument.

For details on the **startnetldi** arguments, see "Utility details for X509" on page 59.

This NetLDI should be started as the same user as the one that started the Stone.

Example

This example specifies that the NetLDI's log file and the log files created by child processes should be written under `$stoneLogDir`.

```
alcatraz> startnetldi -D $stoneLogDir -l $stoneLogDir/54321.log
  -S -U $MyStoneCertDir/alcatraz.chain.pem
  -R $MyStoneCertDir/alcatraz.privkey.pem
  -J $MyStoneCertDir/stoneCA-devstone.cert.pem
  -L $MyStoneCertDir/hostCA-userCA-combined-devstone.crl.pem 54321
```

4. Start certificate-only NetLDI on the Remote Node

With X509 logins, the certificate-only remote cache on a particular node must be started up and running before it can be used by a Gem to login. This is unlike ordinary logins in which the remote cache is only started up when a Gem logs in.

The NetLDI on the remote node has the responsibility of starting up the remote cache, which it does on instructions by the `starthostagent` utility on the Stone's node.

Define or select a configuration file

Since the NetLDI is starting the remote cache, not the Gem, you must include a configuration file for the NetLDI, which includes the parameters used to configure the remote cache. To use the default configuration parameter values, you may pass in an empty file, or any configuration file.

The list of options used by remote caches is on page 58.

In addition to cache configuration, the NETLDI_PORT_RANGE is specific to remote x509-secured NetLDIs. You may include a value for NETLDI_PORT_RANGE to specify the upper and lower bounds (inclusive) of the sockets that the remote Gem will listen on for connections initiated from its Host Agent on the Stone's node.

For example,

```
NETLDI_PORT_RANGE = 50000, 50020;
```

Start the Remote NetLDI

For the remote certificate-only **startnetldi**, you must include the arguments **-S -U -R -J** which allow you to pass in the required certificates, and the **-D** argument for the log file location.

In addition, unlike with the **startnetldi** that is executed on the Stone's node, you must also pass in the remote configuration file using the **-E** argument.

For details on the **startnetldi** arguments, see "startnetldi" on page 60.

Example

```
fiji> startnetldi -E remote.conf -S -D $remoteLogDir
 -l $remoteLogDir/54321.log
 -U $MyRemoteCertDir/fiji.chain.pem
 -R $MyRemoteCertDir/fiji.privkey.pem
 -J $MyRemoteCertDir/stoneCA-devstone.cert.pem 54321
```

5. Start the HostAgent on the Stone's node

Once the remote and Stone's NetLDI are running, you must execute the **starthostagent** utility, on the Stone's node, to complete the startup.

Executing **starthostagent** initiates a number of important tasks. It requests that the Stone's x509 NetLDI do the following:

- The Stone's NetLDI mutually authenticates with the remote x509 NetLDI;
- The Host Agent on the Stone's node starts up and log in;
- The remote NetLDI starts the remote cache

The **starthostagent** utility requires a number of arguments to allow it to securely connect the local and remote NetLDIs:

- the name (or IP) of the remote node, using argument to **-m**.
- the name (or port) that the remote NetLDI is listening on, using argument to **-n**.
- the name (or port) that the Stone's NetLDI is listening on, using argument to **-N**.
- credentials to authenticate with the remote NetLDI, using **-U -R -J**

Example

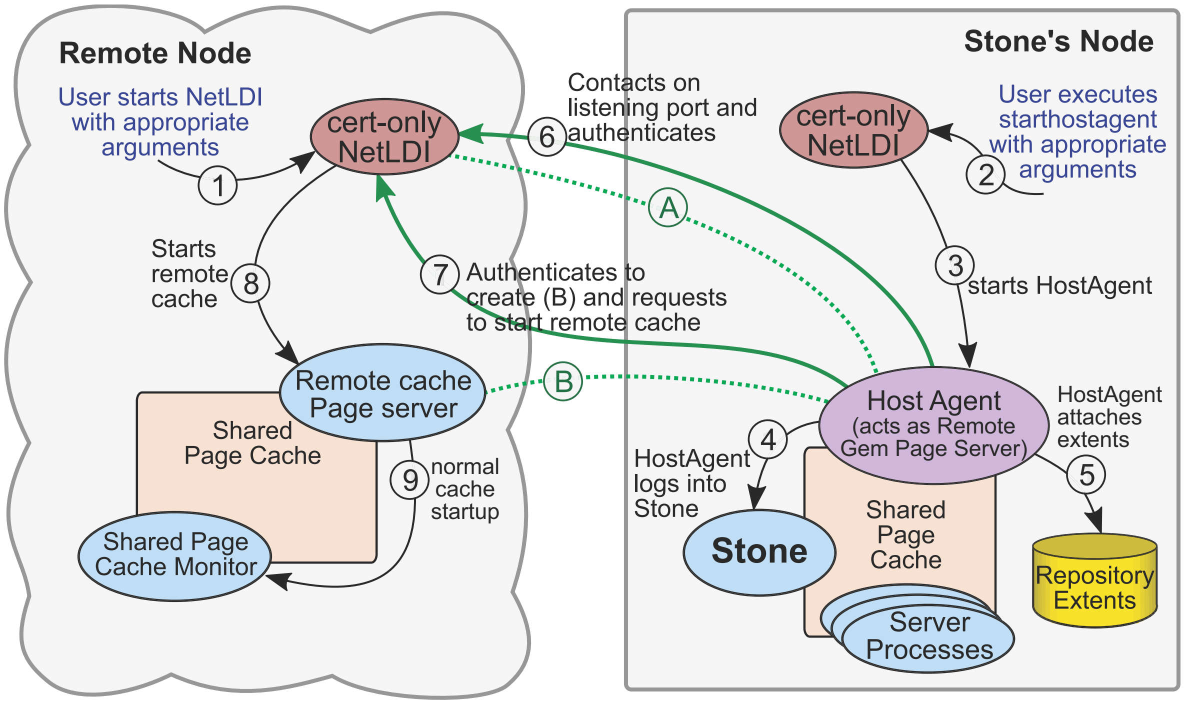
Note that since in our examples the Stone and Remote NetLDIs are both using the same port number, the **-N** *stoneNetLDI* and **-n** *remoteNetLDI* arguments are the same.

```
alcatraz> starthostagent -m fiji -N 54321 -n 54321
 -U $MyStoneCertDir/alcatraz.chain.pem
 -R $MyStoneCertDir/alcatraz.privkey.pem
 -J $MyStoneCertDir/stoneCA-devstone.cert.pem
```

Flow of Operations during HostAgent startup

The following diagram lists the internal steps that occur as a results of starthostagent.

Figure 3.1 Remote Cache and NetLDI startup


Remote Node
User starts NetLDI with appropriate arguments
cert-only NetLDI
1
Contacts on listening port and authenticates
6
A
Starts remote cache
8
7
Authenticates to create (B) and requests to start remote cache
Remote cache Page server
B
Shared Page Cache
9
normal cache startup
Shared Page Cache Monitor
Stone's Node
User executes starthostagent with appropriate arguments
cert-only NetLDI
2
3
starts HostAgent
Host Agent (acts as Remote Gem Page Server)
HostAgent attaches extents
5
4
HostAgent logs into Stone
Stone
Shared Page Cache
Server Processes
Repository Extents

In this diagram, the steps are as follows:

1. The user starts the remote NetLDI using appropriate flags including the certificate flags. These arguments configure the NetLDI to start in certificate-only mode, listen on a public port, and await connection.
2. The user invokes **starthostagent** with the appropriate arguments, specifying the node and remote NetLDI to connect to, and the required certificate files. The starthostagent process contacts the already-running cert-only NetLDI on the Stone's node.
3. The Stone's NetLDI starts (forks) the HostAgent.
4. The HostAgent creates the session in the Stone (logs in) as the user HostAgentUser.

5. The HostAgent attaches the extents (similar to the way the pages servers for a remote node attach the extents).
6. The HostAgent contacts the remote NetLDI on its public port, and performs mutual authentication using host credentials, creating secure connection (A).
7. The HostAgent connects and authenticates with the remote NetLDI again, and requests to fork a remote cache pageserver. The secure connection this creates, (B), is for the remote cache page server.
8. The remote NetLDI forks the remote cache page server. The secure socket (B) from the remote NetLDI to the HostAgent is inherited by the cache pageserver.
9. The cache page server starts associated processes to support the remote cache, as in ordinary remote cache startup.

Secure connection (A) between the HostAgent and the remote NetLDI remains active for use during logins.

6. Login

Logins can be done from topaz, GBS, GCI commands, and GemStone Smalltalk external session classes. Only RPC logins can be done using the secure protocol.

The X509 login requires a set of parameters that are distinct from those used for ordinary logins. The Stone name, GemStone user name and password, and host username and password are not required and must not be set, and NetLDI details are not included in the Gem's NRS (e.g., gemnetid); the NetLDI (of the Gem host) is specified in the form *hostname*:*netldiName*.

In topaz, there are commands to set these parameters. For example:

```
topaz > set cert $MyRemoteCertDir/DataCurator.chain.pem
topaz > set key $MyRemoteCertDir/DataCurator.privkey.pem
topaz > set cacert $MyRemoteCertDir/stoneCA-devstone.cert.pem
topaz > set netldi localhost:54321
topaz > login
```

Setting any of these X509 login parameters clears the login parameters used for ordinary logins (gemstone, username, password, etc.). Likewise, setting any of the ordinary login parameters unsets the X509 login parameters.

The Gem will run on the host specified by the **set netldi** command, which does not need to be the same host as the topaz client. Rather than `localhost`, you may specify the host name or IP of another host that is running a certificate-only NetLDI.

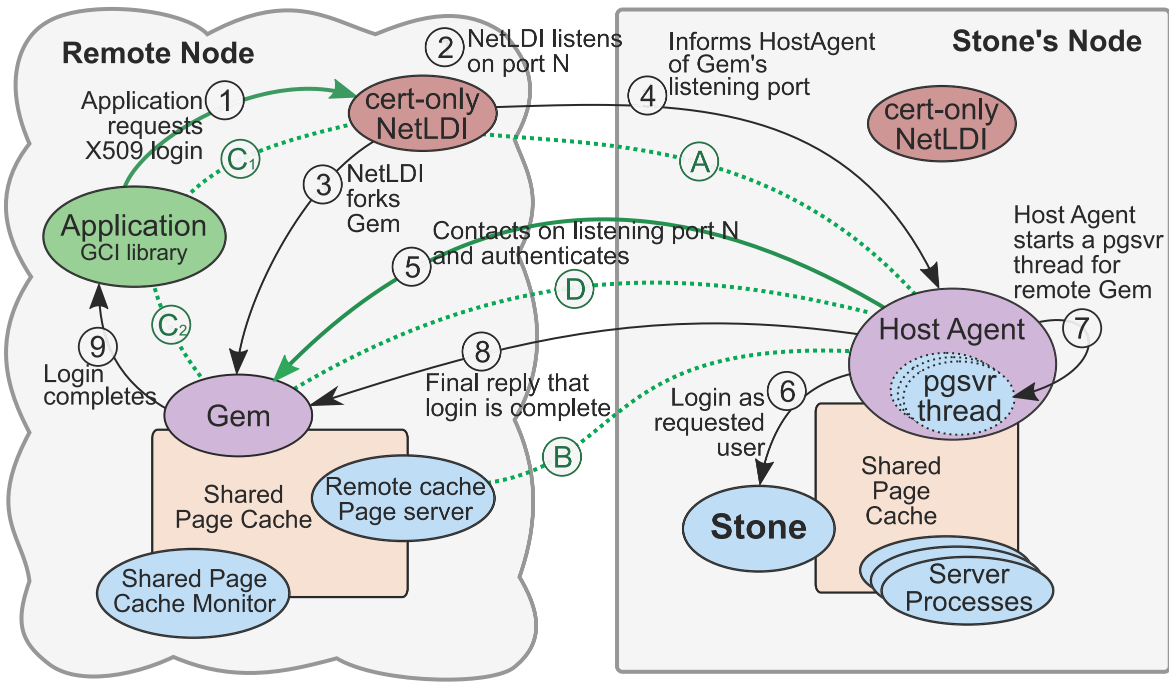
If there are no gems running on the remote cache, after the timeout specified by STN_REMOTE_CACHE_TIMEOUT, the remote cache and HostAgent are shut down. You must execute the **starthostagent** script again (Step 5., above) to restart the remote cache and HostAgent.

Both ordinary and X509 logins are allowed to the stone, but ordinary logins cannot use the certificate-only NetLDIs. The X509-secured remote shared page cache does not support ordinary Gems, so ordinary logins cannot start a Gem on a node where a certificate-based remote shared page cache exists.

Flow of Operations during Login

The following diagram shows the flow of operations when a remove X509 secured Gem logs in.

Figure 3.2 Remote X509 Login



The X509 login process differs considerably from ordinary remote cache login, since the NetLDI startup has already created the remote cache processes, and the socket connections are initiated from the Stone's node.

Secure connections (A) and (B) are established during the setup step in the previous diagram.

1. The application (via GCI library code) requests an X509 login. The remote NetLDI performs mutual authentication with the application, and establishes the secure connection (C_1).
2. The remote NetLDI listens on port N, selected from the range specified by NETLDI_PORT_RANGE. The Gem will accept a connection on that port in step (5).
3. The remote NetLDI forks the Gem, which inherits connection (C), now (C_2), and listening port N.
4. The NetLDI informs the HostAgent via pre-existing persistent secure connection (A) that there is a Gem awaiting login at port N.

5. The HostAgent contacts the Gem on listening port N, and the connection is mutually authenticated; the Gem using user credentials and the HostAgent using host credentials, establishing secure connection (D).
6. The HostAgent creates the session in the Stone (logs in), for the userId provided in the user certificate. No GemStone password is used. The Gem does not maintain a connection directly to the Stone.
7. The HostAgent starts a page server thread to service the remote Gem.
8. The HostAgent provides a final reply to the Gem (on the secure connection (D)), indicating that the login is complete. The Gem returns from the login call, and the application is logged in.

Troubleshooting startup failures

In most cases, details of login failures are not reported, for security reasons.

- If you have trouble starting the Stone's NetLDI, check the log file, and verify the existence, location, and validity of the certificate files.
- If you have trouble starting the Remote NetLDI, first check the remote NetLDI log file. Note that some errors, such as not being able to find the -E configuration file, are reported earlier in the log; be sure to read the entire log file, not merely the final statements.
- If the starthostagent script fails, verify that the Stone was started using a keyfile that allows X509 logins.

  Check the HostAgent log file on the Stone's node, which may contain error reports. The HostAgent log is located in the directory specified by -D argument to startnetldi, and has the name `hostagent-`*StoneName*`-`*RemoteNode*`-`*PIDStoneNodeName*`.log`.
- If the topaz login fails, check your parameters. The **set netldi** hostname must be the name of the Gem's node, not the Stone's node. The error message will also let you know if one or another of the required certs is missing.

Objects hidden by Object filtering

Default object filtering restricts the objects you can access in your remote session, and may make your application objects inaccessible. See Chapter 4 for a full description of object filtering.

An example script to allow objects to be transmitted to X509 remote caches is on page 35.

3.2 X509 logins from Topaz

X509 login parameters

The login command now can be used with two distinct sets of parameters.

- For classic logins, the parameters are unchanged.
- For secure X509 logins, new and distinct parameters are required.

The required parameters for X509 certificate login (RPC only) are:

SET CERT: *certfilepath*
SET CACERT: *cacertfilepath*
SET KEY: *privkeyfilepath*
SET NETLDI: *host:port*

The following parameters are optional, but if used, apply only to X509 logins:

LOGFILE: *gemlogfilepath*
EXTRAGEMARGS: *gemargs*
DIRECTORY: *dirname*

The GemStone user name and the stone name are specified in the certificates, and are not entered in topaz by the user, and the GemStone user's password is not needed.

With X509 logins, NRS is not used in any of the parameters. To specify the directory and log file name and any arguments to be passed to the gem, specific **set** commands are used.

If any of the X509 login parameters are set, the ordinary parameters are cleared, and the **login** command will perform an X509 login; if any of the ordinary parameters are set, the X509 parameters are cleared, and **login** performs a classic login.

The full set of new **set** commands are:

CACERT[**:**] *cacertfilepath*
Sets the path to the X509 certificate authority (CA) certificate to be used for login. The certificate must be in PEM format.

CERT[**:**] *certfilepath*
Sets the path to the X509 certificate to be used for login. The certificate must be in PEM format.

DIRECTORY[**:**] *dirname*
Set the name of the working directory for the RPC gem created by a certificate login. Also specifies the directory in which the Gem log will be written; if not specified, the log file is put into the directory provided in the startnetldi -D argument.

EXTRAGEMARGS[**:**] *gemargs*
Sets a list of extra command line arguments to be used when starting an RPC gem for a certificate login.

KEY[**:**] *privkeyfilepath*
Sets the path to the private key for the certificate specified by the SET CERT: command. The key must be in PEM format and must not be protected by a passphrase.

LOGFILE[**:**] *gemlogfilepath*
Set the name of the RPC gem's log file for certificate logins. If not specified, the default log file name is used.

NETLDI[**:**] *hostname:port*
Sets the hostname and port number for the certificate-only NetLDI to be used for X509 login. The format must include a the name or IP of the host, a colon, and the port or NetLDI service name. For example,

```
topaz> set netldi fiji.gemtalksystems.com:54321
```

That this is the Gem host's NetLDI. The Gem will be started on the specified node ***fiji***. You may use localhost to specify the Gem should be on the same node as the topaz process.

While the user certificate may include an address restriction, using `localhost` and other addresses 127.x.x.x (loopback addresses) are allowed, for running Gems on the same node. The address restrictions are applied when the Gem authenticates with the HostAgent.

The commands to set X509 parameters may be set in the .topazini file, or may be passed in on the command line using **-X -n**.

topaz arguments to configure X509 parameters on command line

Rather than specifying the certificates using the set command, you may pass them in on the command line using the topaz **-X** argument. The **-X** is only valid for RPC sessions, and takes a string containing three semicolon-delimited paths. For example:

```
'stoneCA-devstone.cert.pem;DataCurator.chain.pem;DataCurato
   r.privkey.pem'
```

The order is significant; the CA, cert, and private key must be in that order.

To set all the X509 login parameters required for login, you must also use the **-n** argument to specify the NetLDI name and port.

For example:

```
topaz -X '$MyCertDir/stoneCA-devstone.cert.pem;$MyCertDir/DataC
   urator.chain.pem;$MyCertDir/DataCurator.privkey.pem'
   -n localhost:54321
```

Status command

The topaz **status** command includes a section with the parameter information:

```
X509 Login Certificate Information:
Certificate_________ '$MyRemoteCertDir/DataCurator.chain.pem'
CaCertificate_______ '$MyRemoteCertDir/stoneCA-devstone.cert.pem'
Key_________________ '$MyRemoteCertDir/DataCurator.privkey.pem'
Netldi______________ 'localhost:54321'
Directory___________ ''
LogFile_____________ ''
ExtraGemArgs________ ''
```

3.3 X509 logins using the GCI interface

To perform X509 logins, use the `GciX509Login` function, which uses an instance of the C++ class `GciX509LoginArg` to hold the login parameters.

```
(BoolType) GciX509Login(
      GciX509LoginArg *args);

class CLS_EXPORT GciX509LoginArg
public:
      const char   *netldiHostOrIp;
      const char   *netldiNameOrPort;
      const char   *privateKey;
      const char   *cert;
      const char   *caCert;
      const char   *extraGemArgs;
      const char   *dirArg;
      const char   *logArg;
      unsigned int loginFlags;
      BoolType     argsArePemStrings;
      BoolType     executedSessionInit; // output
```

If `argsArePemStrings` is true, the `privateKey`, `cert`, and `caCert` are strings in PEM format. If false, these are strings containing the name of a file that is in PEM format.

3.4 X509 logins using GBS

GemBuilder for Smalltalk/VW v8.4 supports X509-Secured logins with a parameters class, GbsX509SessionParameters. Instances of this class may be created programmatically, or by using the GUI tools. Once instances are created and added to the set of available login parameters, login and logout can be done as usual using the GBS Launcher.

When creating an instance of GbsX509SessionParameters using the GBS Launcher, a dialog asks if the new parameters are for X509 login, and brings up the appropriate dialog.

To programmatically create a new instance of GbsX509SessionParameters, execute an expression such as:

```
| params |
params := GbsX509SessionParameters new.
params
    caCert: 'stoneCA-devstone.cert.pem';
    cert: 'DataCurator.chain.pem';
    privateKey: 'DataCurator.privkey.pem';
    netldiHostOrIp: 'fiji';
    netldiNameOrPort: '54321'.
GBSM addParameters: params.
```

Once the parameters are created, they can be used in the GBS Launcher or programmatically as ordinary parameters. For example, you can login using an expression such as:

```
session := GBSM loginWithParameters: params.
```

As with topaz, there are additional option arguments to allow you to configure the Gem log and other arguments. The following are also may be set in an instance of GbsX509SessionParameters. Currently, these can only be specified programmatically.

```
extraGemArgs:
dirArg:
logArg:
```

GemBuilder for Smalltalk/VA does not support X509-Secured logins.

3.5 X509 logins using External Sessions

The classes GsX509ExternalSession and GemStoneX509Parameters are variant of GsExternalSession and GemStoneParameters, which allow you to specify X509 certificates as login credentials rather than user id and password.

GsX509ExternalSessions are not supported on AIX.

For example:

```
topaz 1> run
| params session |
params := GemStoneX509Parameters
      newFromPemFilesWithNetldiPort: '54321'
      netldiHost: 'localhost'
      certificate: 'DataCurator.chain.pem'
      caCertificate: 'stoneCA-devstone.cert.pem'
      privateKey: 'DataCurator.privkey.pem'.
session := GsX509ExternalSession newWithX509Parameters: params.
session login.
%
```

3.6 Local Logins

The preceding instructions describe how to login a remote session, that is, with the Gem process on a different node than the Stone's node; remote logins constitute the greater security exposure.

X509-secured logins are also with the Gem on the Stone's node. This configuration can include a client that is also on the Stone's note, or on a remote node.

For a user certificate to be used for GciX509Login where the Gem is on the Stone's node, any -a argument to the certificate creation script **newuser** must specify either 0.0.0.0/0 or an address 127.0.x.x.

X509 logins with Gem on the Stone's node are not allowed by SystemUser.

Chapter

4

Remote Cache Object Filtering

GemStone provides object-level security by associating specific objects with an ObjectSecurityPolicy. By managing ObjectSecurityPolicies, you can prevent users from reading or modifying protected objects.

However, this does not prevent all risk of data exposure on a remote node. GemStone's database pages may contain multiple objects, and when pages containing objects that are allowed to be read by the user session are transmitted to a remote cache, objects that cannot be read may still "ride along".

Object filtering in X509-secured environments provides specific control over exactly which objects are allowed to be transmitted to any given remote cache.

4.1 Overview

Overview of Object level security

The previously existing object level security associated each object in the repository with an instance of ObjectSecurityPolicy (known as Segments in older versions of GemStone), or with nil, which means there are no restrictions. GemStone base defines ten system ObjectSecurityPolicies, and users may define up to 64K application-specific policies.

ObjectSecurityPolicy permissions are based on the GemStone user ID. Sessions that are logged in with a specific userId will be allowed to read or modify objects if the policy associated with that object give that user read or write permission for the owner, group or world.

Persistent objects in the extents and cache are on pages, and a page may contain multiple unrelated objects. When an object is read, the entire page holding that object is loaded into the shared page cache, and/or transmitted to the remote cache. When a Gem session accesses an object, the policy for that object is checked against the Gem's userId, before allowing the object on the page to be faulted into the application's object space.

While this effectively prevents the user f rom accessing sensitive data, the sensitive data may still be present in a remote shared page cache. Using Object Filtering, it is possible to prevent sensitive data from being transmitted out of the Stone's shared page cache at all.

Object Filtering

Object filtering adds another layer of protection for sensitive data. This is implemented by associating GemStone's existing object-level security mechanism, with IP address range-based filtering policies. By configuring the IP address range to which each object may be transmitted, you can control exactly where that data can be present.

- Object filtering is performed by the HostAgent on the Stone's node, and only applies to X509-secured remote caches.
- Filtering only applies to remote caches; there is (of necessity) no restriction on what is loaded into the Stone's shared page cache
- Objects associated with the SecurityDataObjectSecurityPolicy may never be transmitted to any host that is using x509 logins, regardless of any filtering configuration.

If a Smalltalk application with a X509-secured Gem attempts to access an object that is not allowed to be transmitted to the Gem's host, then the application cannot read that object, regardless of the object's object security policy read and write permissions.

If that application attempts to access an object that is allowed to be transmitted, then the object's object security policy applies, which may allow that object to be read or not, just as in previous GemStone versions.

Object Filtering support classes

The classes IPv4Subnet, ObjectFilteringPolicy, and ObjectFilteringPolicyMap allow object filtering to be configured.

IPv4Subnet allows naming of CIDR address masks, to make management easier. (IPv6 networks are not currently supported with GemStone x509 logins).

An instance of **ObjectFilteringPolicy** maps every possible GsObjectSecurityPolicy to an action of ALLOW or PROHIBIT. A single instance of ObjectFilteringPolicy specifies filtering for every single object in the repository; but different remote caches may use different instances of ObjectFilteringPolicy.

There is one **ObjectFilteringPolicyMap** installed, which maps every possible IP address to an instance of ObjectFilteringPolicy, which is what would be used for a remote node on that IP address.

The keys in the ObjectFilteringPolicyMap are instances of IPv4Subnet, which specify ranges rather than specific IP addresses. This allows the application to define multiple overlapping policies, including policies that apply to all IP address, one specific IP address, or various ranges of IP address. When looking up the IP address for a specific remote cache, the most specific IP address range that contains that IP address is used.

The global instance of ObjectFilteringPolicyMap is installed in the image by DataCurator, using the method

```
ObjectFilteringPolicyMap >> installObjectFilter
```

The map is stored under the key named **#ObjectFilter** in the HostAgentUser's UserGlobals.

Note that a new instance of ObjectFilteringPolicyMap, including the instance in new or newly upgraded repositories, has a default action of DISALLOW and may not be very usable; users with object security problems may not be able to login at all. Applications are expected to execute code to install an appropriately configured instance of ObjectFilteringPolicyMap. This default avoids inadvertent exposures at the cost of extra effort on the initial setup.

You can define and install a filter that allows all objects to be sent to remote caches on any of your organization's internal network nodes (with IP addresses 10.*.*.*), by logging in as DataCurator and executing code such as this:

```
topaz 1> run
| snet pol |
snet := IPv4Subnet named: 'internal_ips' forSubnet: '10.0.0.0/8'.
pol := ObjectFilteringPolicy new
      name: 'policy_allow_internal'; allowByDefault;
      yourself.
(ObjectFilteringPolicyMap new)
      atSubnet: snet putPolicy: pol;
      installObjectFilter.
System commit.
%
```

The newly added filtering classes and objects are themselves in a new GsObjectSecurityPolicy, HostAgentDataSecurityPolicy. This security policy is mapped to PROHIBIT, since they are never needed outside of the Stone's node.

4.2 Details on Classes that implement Object Filtering

Object filtering is implemented using several classes that allow you to specify the particular filtering requirements, and map these to one or more GsObjectSecurityPolicies.

IPv4Subnet

An IPv4Subnet allows subnets to be associated with names. This avoids the need to remember IP specific address ranges in a large network, and reduces the risk of errors.

Public class messages:

`IPv4Subnet class >> named:` *nameString* `forSubnet:` *cidrString*
This is the public instance creation message. The *nameString* argument can be any string. The *cidrString* must be a string specifying a valid CIDR subnet.

`IPv4Subnet >> name`
Answers the receiver's name string.

`IPv4Subnet >> cidrString`
Answers the receiver's CIDR string.

ObjectFilteringPolicy

An instance of ObjectFilteringPolicy maps all possible GsObjectSecurityPolicies to a filtering action of ALLOW or PROHIBIT.

When a particular ObjectFilteringPolicy is in use for a remote cache, an object's GsObjectSecurityPolicy will map to either ALLOW or PROHIBIT, which will determine if that particular object will be transmitted to that remote cache.

Creation

ObjectFilteringPolicies are created using `ObjectFilteringPolicy class >> new`. A newly-created ObjectFilteringPolicy maps all GsObjectSecurityPolicies to an action of PROHIBIT (with some exceptions).

Each ObjectFilteringPolicy has a name, by default nil. It is recommended to give each ObjectFilteringPolicy a unique name to make them easier to manage and validate.

`name:` *aString*
Sets the name of the policy.

`name`
Answers the name of the policy, or nil if no name has been set.

Specifying mappings

An ObjectFilteringPolicy has two collections of GsObjectSecurityPolicies: one collection whose action should be ALLOW and another whose action should be PROHIBIT. There is a default action of ALLOW or PROHIBIT which applies to any GsObjectSecurityPolicy that has not specifically been added to either collection.

The following methods are used to define the mappings:

`allowByDefault`
Sets the default action of the receiver to ALLOW.

`prohibitByDefault`
Sets the default action of the receiver to PROHIBIT.

`allow:` *aGsObjectSecurityPolicy*
Adds the given security policy to the ALLOW collection, and removes it from the PROHIBIT collection (if there).

`prohibit:` *aGsObjectSecurityPolicy*
Adds the given security policy to the PROHIBIT collection, and removes it from the ALLOW collection (if there).

`allowAll:` *aCollection*
Add every security policy in *aCollection* to the ALLOW collection, and remove from the PROHIBIT collection if present.

`prohibitAll:` *aCollection*
Add every security policy in *aCollection* to the PROHIBIT collection, and remove from the ALLOW collection if present.

ObjectFilteringPolicyMap

An ObjectFilteringPolicyMap maps every possible IPv4 address to an ObjectFilteringPolicy that should be used for a host at that IPv4 address.

The ObjectFilteringPolicyMap keys are subnet masks (instance of IPv4Subnet), rather than specific IPv4 addresses. There may be more than one mapping that applies to a specific address, if the keys have different mask integers.

At runtime, when the map is queried for the ObjectFilteringPolicy to be used for a specific IP address, it returns the most selective entry, which is the entry with the largest mask integer that includes the argument.

For example, a ObjectFilteringPolicyMap may includes entries for the following subnets:

```
0.0.0.0/0 -> anObjectFilterPolicy1
10.0.0.0/8 -> anObjectFilterPolicy2
10.12.0.0/16 -> anObjectFilterPolicy3
10.12.241.0/24 -> anObjectFilterPolicy4
10.12.241.67/32 -> anObjectFilterPolicy5
```

In this case, querying for the IP address `10.12.241.67` would return `anObjectFilterPolicy5`, and for the IP address 10.12.112.17 would return `anObjectFilterPolicy3.`

Each instance of ObjectFilteringPolicyMap always contains a mapping for subnet 0.0.0.0/0 (which is the CIDR subnet containing the entire IPv4 address space). This mapping will normally be overridden, but ensures that any well-formed IPv4 address has a mapping.

Specifying and looking up policies within a map

`ObjectFilteringPolicyMap >> atSubnet:` *anIPv4Subnet* `putPolicy:` *anObjectFilteringPolicy*
Add a mapping for the given subnet, or replace the mapping for the subnet if the receiver already has a mapping for that subnet.

`ObjectFilteringPolicyMap >> policiesForSubnet:` *anIPv4Subnet*
Answer a collection of Associations. Each association's key is an IPv4Subnet, and its value is an ObjectFilteringPolicy. The result will include mappings for all the policies in effect for all addresses in the subnet *anIPv4Subnet*.

This will always include the mapping for 0.0.0.0/0, and may include multiple mappings that apply to some or all of the addresses described by *anIPv4Subnet.* If there was no policy specified for the given *anIPv4Subnet*, then there will be no mapping with that key; only the mappings that cover that particular range are returned.

`ObjectFilteringPolicyMap >> policyForIP:` *ipAddrString*
Answers the policy for the given IP address string, which must be a valid dotted-quad string. The policy answered will be the policy defined for the smallest mapped subnet which contains the given IP address, that is, the one with the largest mask integer. Any well-formed IPv4 address string will always map to a policy, since the mapping for 0.0.0.0/0 covers all possible addresses.

Note that there is no API for removing mappings. It is recommended to maintain Smalltalk scripts to build and rebuild the map as security properties are updated. Reviewing Smalltalk code is less error-prone than making ad-hoc adjustments to a complex data structure.

Installing and finding out about the defined map/ObjectFilter

The following methods allow you to set and fetch the information about the current instance of ObjectFilteringPolicyMap. Note that these names are subject to change in later alpha versions.

`ObjectFilteringPolicyMap >> mappingReport`
Answers a string detailing the policies for all subnets.

`ObjectFilteringPolicyMap >> installObjectFilter`
Install the receiver into HostAgentUser's UserGlobals at: #ObjectFilter. Once committed, this will be the filter for newly started HostAgents.

Note that this does NOT affect existing HostAgents. You must restart the HostAgent on the Stone's node in order for the new filter map to be in effect.

`ObjectFilteringPolicyMap class >> installedObjectFilter`
Return the currently installed instance of ObjectFilteringPolicyMap, from HostAgentUser's UserGlobals at: #ObjectFilter.

UnauthorizedObjectStub

When the HostAgent filters objects from the results sent to the remote node, an instance of UnauthorizedObjectStub represent objects for which an object fault would signal a SecurityError for no read authorization.

This instance is in the data page, and will only be visible to a Gem on that node if the Gem attempts to access the object. If the UnauthorizedObjectStub is replacing an object on a page that happens to be "riding along" with data that a user does have authorization to see, the presence of the UnauthorizedObjectStub will be unnoticeable to the user.

4.3 ObjectFilter internal and usage details

GemStone may have a maximum of 65536 (64K) GsObjectSecurityPolicies, which allows a ObjectFilteringPolicy to be converted into a 8192-byte ByteArray, which has 64K bits. Each bit represents a possible GsObjectSecurityPolicy. It is likely most of them will not be in use, and be set according to the default action for the ObjectFilteringPolicy. Each bit is either 0 to allow objects with the associated security policy to be transmitted, or 1 to disallow transmission.

When a HostAgent is started up for a specific remote node, it looks up that node's IP address in the installed ObjectFilter. This returns the object filtering policy that applies for the remote node.

The filter is converted into a ByteArray, which is passed to the page server thread in the HostAgent that serves that remote node. As processes on the remote node request pages, each object on the page is examined, and for any objects that are disallowed for transmission, it will replace an UnauthorizedObjectStub for that object on that portion of the page.

Changing the ObjectFilter

Changes to ObjectFilter during the lifetime of a HostAgent have no effect, since the ObjectFilter map is only consulted at HostAgent startup. After an ObjectFilter is reinstalled, all existing HostAgents will continue to use the old policies, but any newly-created HostAgents will use the new policies.

To update the policy for a remote node, shut down all Gems and the shared page cache on the remote node machine, then stop the HostAgent's session. The next restart of the NetLDI on the remote node, or new login using the existing NetLDI, will restart the cache and HostAgent, which will use the new ObjectFilter.

Filtering and mid level caches

When a mid level cache is used, there can be two transmissions of objects:

- from the stone host to the mid level cache host
- from the mid level cache host to the leaf cache host.

To prevent inconsistencies between the contents of pages in the mid level cache and pages in the leaf caches it serves, ObjectFilteringPolicy for the mid level cache host must be equal to the ObjectFilteringPolicy of each of its leaf caches. This restriction is enforced by HostAgents on the Stone's node.

When a Gem on a leaf cache attempts to connect to a mid-level cache, and the ObjectFilteringPolicy for the Gem's leaf cache does not match the ObjectFilteringPolicy for the mid-level cache node, the Stone's HostAgent will reject the request.

Since the mid level cache and all of the leaf cache hosts it serves have equivalent ObjectFilteringPolicies, mid level cache hosts do not need to perform filtering when sending pages to leaf hosts. The Mid-level cache HostAgent does not do any filtering.

Chapter
5
X509 Mid Level Cache

This chapter describes how to setup and login using an X509-secured mid level cache (MLC).

5.1 Overview

In a distributed system over a Wide Area Network (WAN), with many remote nodes that are topographically distant from the Stone but close to each other, a mid-level cache can improve performance for the remote sessions.

This configuration involves not at least three nodes, which of which will contain a shared page cache:

- the Stone's node, which contains the Stone's SPC
- the mid level cache node, containing the mid-level cache. Gems may run on the mid-level cache node in which case the mid-level cache acts as the Gem's cache.
- the leaf node where the Gem is running; the Gem's client application and be on the leaf node or on yet another node. This node contains the remote cache; it is remote from the Stone, but local to the Gem, so the term "leaf" is used to avoid confusion.

When the Gem needs a page but can't find it in its leaf cache, it first looks in the mid-level cache. If the page is not in found in the mid-level cache, it then forwards the request to the page server on the Stone's host.

X509-secured Mid-level caches

You may configure your system to use an X509-secured mid-level cache on a node that will service X509-secured Gems on other leaf nodes.

As with ordinary remote caches, x509-secured mid-level caches are not compatible with non-secured mid-level caches, remote caches, NetLDIs, or Gems.

X509-secured mid level caches, like secured remote caches, require the NetLDI be started up and the full authentication initiated from the Stone's node, before a Gem can connect and start using the mid-level cache.

The following are additional differences from X509-secured remote caches:

- An X509-secured NetLDI must be started on the mid-level cache node, before the connection from the Gem is initiated. This must be started with a configuration file including NETLDI_START_MIDCACHE, which specifically allows this Netldi to start a mid-level cache.
- During startup of the MLC cache, the MLC node's NetLDI starts a HostAgent process on the mid-level cache node; this is distinct from the HostAgent on the Stone's node. To allow login as HostAgentUser, the HostAgentUser's certificates and private key must be configured. This extra HostAgent allows this cache to operate as a mid-level cache.
- The extra MLC HostAgent is counted as a session and will prevent the mid-level cache from automatically timing out and shutting down the way ordinary remote caches do.

5.2 Configuring and Starting the X509 Mid Level Cache

The following instructions provide the steps to startup, then connect to, a mid-level cache node.

The examples use the name ***brisbane*** for the mid-level cache node name, and as with the examples in Chapter 3, the Stone named ***devstone*** running on a node named ***alcatraz***, and a remote x509 Gem is running on ***fiji***. The NetLDI is accessed via the port number 54321.

Starting the mid-level cache NetLDI

1. Create Certificates and configure on mid-cache host

Create certificates

A HostAgent is started on the mid-level cache node, which logs into the Stone as HostAgentUser. This requires cert files for the HostAgentUser on the mid-level cache node.

The mid-level cache node also requires Host certificates.

For example:

```
unix> $GEMSTONE/bin/x509/newuser -s devstone HostAgentUser
unix> $GEMSTONE/bin/x509/newhost -s devstone brisbane
```

Copy to a conveniently accessible location

For convenience, you may create a new flat directory in which to place the required certificate files, and access it via an environment variable. These examples use $MyRemoteCertDir.

```
export MyRemoteCertDir=dirName
```

```
cp $GEMSTONE_CERT_DIR/stones/devstone/stoneCA/stoneCA-
   devstone.cert.pem $MyRemoteCertDir
cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/brisbane/brisbane.chai
   n.pem $MyRemoteCertDir
cp $GEMSTONE_CERT_DIR/stones/devstone/hosts/brisbane/brisbane.priv
   key.pem $MyRemoteCertDir
cp $GEMSTONE_CERT_DIR/stones/devstone/users/HostAgentUser/HostA
   gentUser.chain.pem $MyRemoteCertDir
cp $GEMSTONE_CERT_DIR/stones/devstone/users/HostAgentUser/HostA
   gentUser.privkey.pem $MyRemoteCertDir
```

Set up log directories

To make it easier to locate the log files and diagnose connection issues, it is recommended to create a log file directory, and direct all related logs to that directory.

These examples use the environment variables **`$remoteLogDir`** to refer to these directories

For example:

```
brisbane> mkdir $GEMSTONE/logs
brisbane> export remoteLogDir=$GEMSTONE/logs
```

2. Start the mid-level cache's NetLDI

As with leaf X509 NetLDIs, the mid-level cache NetLDI has the responsibility of starting up the remote cache, which it does on instructions by the `starthostagent` utility on the Stone's node.

Define a configuration file

One of the arguments to a X509-secured remote cache is a configuration file, which includes the parameters used to configure the remote cache, and for mid-level caches, there are additional configuration parameters required.

```
NETLDI_START_MIDCACHE = true ;
NETLDI_HostAgentUser_cert = pathForHostAgentUserCert ;
NETLDI_HostAgentUser_key = pathForHostAgentUserPrivateKey ;
```

You may also optionally include `NETLDI_PORT_RANGE` and other remote cache configuration options in this configuration file. Supported options are listed on page 58.

Start the Mid-Cache NetLDI

The Stone must be running before you start the x509-secured mid-cache NetLDI.

For the mid-cache certificate-only NetLDI, you must include the arguments **-S -U -R -J** which allow you to pass in the required certificates, and the required **-D** argument to set the directory for log files.

In addition, you must also pass in the remote configuration file using the **-E** argument, which must include the configuration parameter settings NETLDI_START_MIDCACHE (set to true), NETLDI_HostAgentUser_cert and NETLDI_HostAgentUser_key.

For details on the **startnetldi** arguments, see "Utility details for X509" on page 59.

Example

To start a mid-level cache on node ***brisbane***, the configuration file `midcache.conf` could contain:

```
NETLDI_START_MIDCACHE = true;
NETLDI_HostAgentUser_cert =
   $MyRemoteCertDir/HostAgentUser.chain.pem;
NETLDI_HostAgentUser_key =
   $MyRemoteCertDir/HostAgentUser.privkey.pem;
```

and the following would start the NetLDI on the mid-level cache node:

```
brisbane> startnetldi -E midcache.conf -S -D $remoteLogDir
   -U $MyRemoteCertDir/brisbane.chain.pem
   -R $MyRemoteCertDir/brisbane.privkey.pem
   -J $MyRemoteCertDir/stoneCA-devstone.cert.pem 54321
```

3. Start the two HostAgents from the Stone's node

When **starthostagent** is executed on the Stone's node, and the **-m** argument node is running a mid-cache enabled X509-secured NetLDI, then connections are initiated that setup the argument node to use for a mid-level cache.

As with regular x509-secured remote nodes, a HostAgent is started on the Stone's node to service that remote node, and the remote NetLDI is instructed to start a remote shared cache on its node.

In addition, the mid-cache NetLDI starts a HostAgent on the mid-level cache node. This remote HostAgent allows this node to act as a mid-level cache.

Internal Details

The MLC NetLDI starts the MLC HostAgent by initiating a topaz -r session, and executing the .ini file `$GEMSTONE/sys/midhostagenttopaz.ini`.

The Gem session logs in securely as HostAgentUser, using the CA cert passed in to **startnetldi** with the **-J** argument and the HostAgentUser's certificate and private key as configured in the **-E** configuration file.

Once a secure login is established, the topaz session executes code to become the HostAgent.

Log files

The topaz and gem logs will be located in the directory specified by the **-D** argument to **startnetldi**, with names:

```
midcacheHostagentGemPIDnodeName.log
midcacheHostagentTopazPIDnodeName.log
```

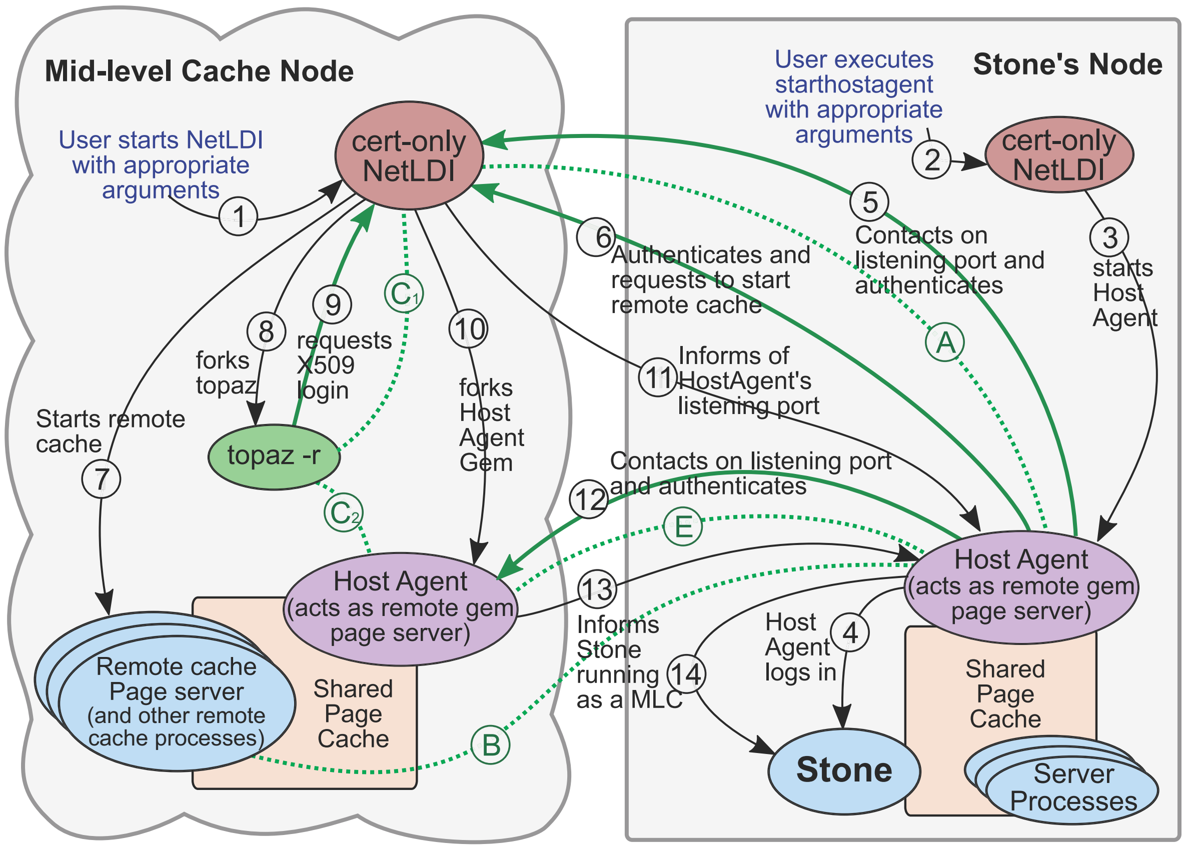
Example

```
alcatraz> starthostagent -m brisbane -N 54321 -n 54321
   -U $MyStoneCertDir/alcatraz.chain.pem
   -R $MyStoneCertDir/alcatraz.privkey.pem
   -J $MyStoneCertDir/stoneCA-devstone.cert.pem
```

Flow of Operations

The following diagrams shows the order of operations in the startup of a mid-level cache.

Figure 5.1 Mid-Level Cache startup


Mid-level Cache Node
User starts NetLDI with appropriate arguments
1
cert-only NetLDI
Starts remote cache
7
8
forks topaz
9
requests X509 login
C1
10
forks Host Agent Gem
topaz -r
C2
Host Agent (acts as remote gem page server)
Remote cache Page server (and other remote cache processes)
Shared Page Cache
B
Stone's Node
User executes starthostagent with appropriate arguments
2
cert-only NetLDI
3
starts Host Agent
5
Contacts on listening port and authenticates
6
Authenticates and requests to start remote cache
A
11
Informs of HostAgent's listening port
12
Contacts on listening port and authenticates
E
13
Informs Stone running as a MLC
14
Host Agent logs in
4
Host Agent (acts as remote gem page server)
Shared Page Cache
Stone
Server Processes

In this diagram, the steps are as follows:

1. The user starts the remote NetLDI using appropriate flags including the certificate flags, and the -E configuration file with NETLDI_START_MIDCACHE = true. These arguments configure the NetLDI to start in certificate-only mode, listen on a public port, and await connection.
2. The user invokes **starthostagent** with the appropriate arguments, specifying the node and mid-level NetLDI to connect to, and the required certificate files. The starthostagent process contacts the already-running cert-only NetLDI on the Stone's node.
3. The Stone’s NetLDI starts (forks) the HostAgent.
4. As is done for any X509 remote cache, the HostAgent logs in.

5. The HostAgent contacts the mid-level NetLDI on its public port, and performs mutual authentication using host credentials, creating secure connection (A).
6. The HostAgent connects and authenticates with the mid-level NetLDI again, and requests to fork a remote cache pageserver. The secure connection creates (B), which will be inherited by the remote cache page server.
7. As for any remote X509 cache, the mid-level NetLDI forks the remote cache pageserver based the -E configuration values, and starts the associated processes.
8. The remote NetLDI spawns a topaz process that will be used to start the Mid Level Cache HostAgent, passing along the location of the cert files for the login in step (9), which are in the NetLDI's -E configuration file and its -J CA cert.
9. topaz requests the X509 secured login as HostAgentUser, using the supplied certs. This authentication creates secure connection (C), which will remain active, since topaz is the GCI client for the HostAgent.
10. The remote NetLDI selects and listens on a port N, which will be used by the mid-level HostAgent to accept a connection from the Stone's HostAgent, in step (12). The NetLDI spawns the MLC HostAgent Gem on the mid-level cache node.
11. The remote NetLDI informs the HostAgent on the Stone's node of listening port N for the HostAgent Gem, via secure connection (A).
12. The Stone's HostAgent contacts the MLC HostAgent on listening port N and authenticates, establishing secure connection (E). Note that (D) is skipped, to make the following mid-level cache connection diagram clearer.
13. The MLC HostAgent, via secure connection (E), requests the Stone's HostAgent to update the Stone, indicating it is running as a mid-level cache node.
14. The Stone's HostAgent passes the information to the Stone.

NetLDI, cache, and HostAgent startup are complete, and a Gem can now connect to the mid-level cache on this node.

5.3 Connecting to a mid-level cache

Connecting a Gem to a mid-level cache is operationally much like connecting an ordinary session to an ordinary mid-level cache.

The Gem must perform an x509-secured login. After successful login, the Gem session connects to a mid-level cache using:

```
System class >> midLevelCacheConnect:
```

You may use the same mid level cache connect methods that were used in previous versions to make connections between an ordinary gem and an ordinary mid-level cache. These methods include keywords with arguments for configure the mid-cache; these arguments are ignored when connecting an X509-secured Gem to an x509-secured mid level cache.

You may execute:

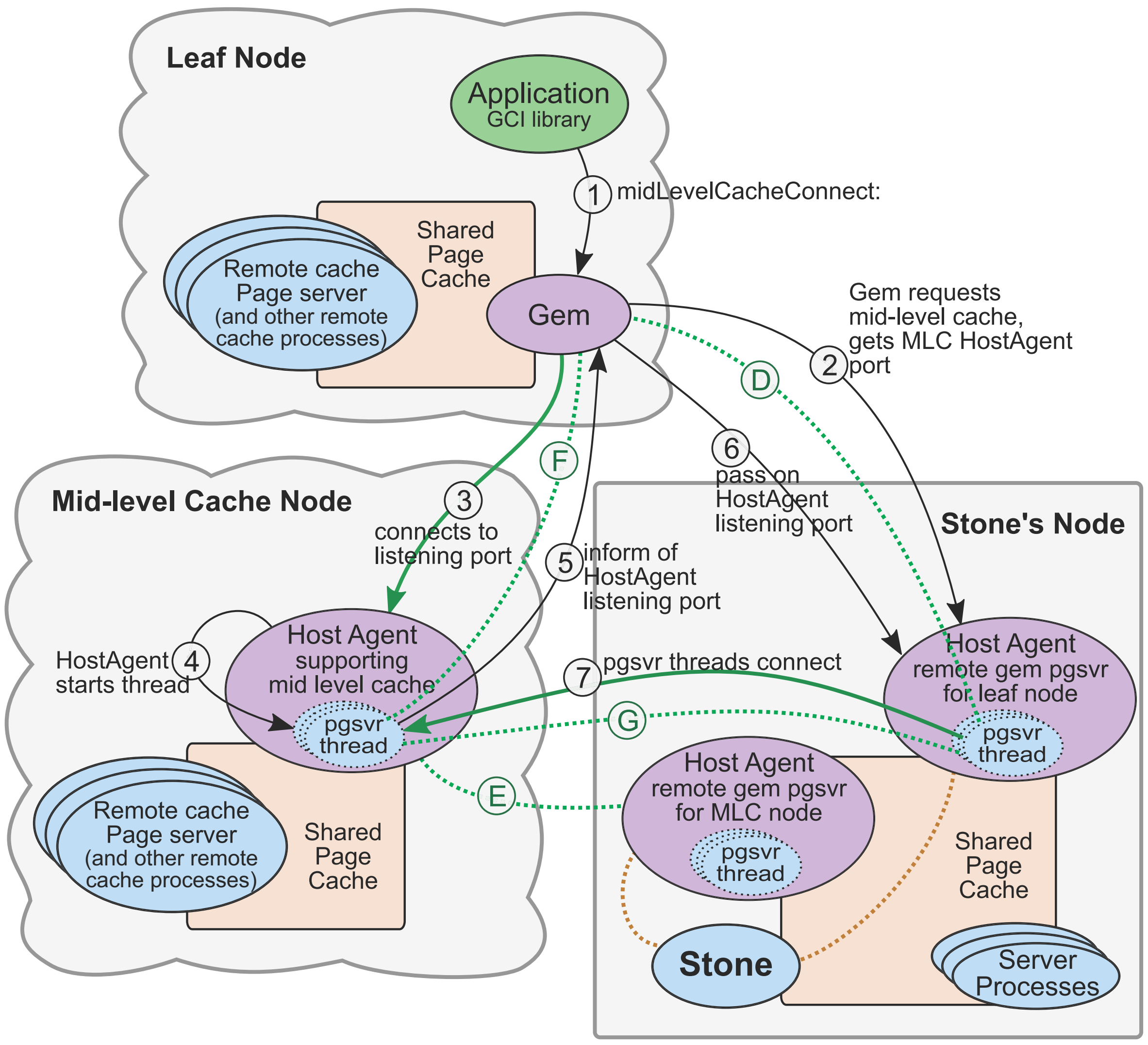
```
System class >> midLevelCachesReport
```

To see that the cache type **x509mid** is present.

Flow of Operations

The following diagrams shows the order of operations when a Gem connects to a mid-level cache.

Figure 5.2 Connecting Leaf Gem to Mid-level cache


Leaf Node
Application
GCI library
1 midLevelCacheConnect:
Shared Page Cache
Remote cache Page server (and other remote cache processes)
Gem
Gem requests mid-level cache, gets MLC HostAgent port
2
D
F
6 pass on HostAgent listening port
Mid-level Cache Node
3 connects to listening port
5 inform of HostAgent listening port
Stone's Node
HostAgent starts thread
4
Host Agent supporting mid level cache
pgsvr thread
7 pgsvr threads connect
Host Agent remote gem pgsvr for leaf node
pgsvr thread
G
E
Host Agent remote gem pgsvr for MLC node
pgsvr thread
Remote cache Page server (and other remote cache processes)
Shared Page Cache
Shared Page Cache
Stone
Server Processes

When an X509 mid-level cache connection is requested, the mid-level cache and associated processes must already have been started, so it is a matter of starting a thread on the mid-level cache to act as a pageserver, and connecting it securely to the Gem and the Gem's HostAgent pgsvr thread.

This diagram is simplified to avoid details that are shown in previous diagrams. The (D) and (E) secure connections are existing secured connections setup during the initial login of the remote X509 Gem (Figure 3.2) and during the startup of the Mid Level cache (Figure 5.1).

1. The application calls `System midLevelCacheConnect:`, specifying the mid-level cache node.
2. Via secure connection (D), the Gem requests the mid-level cache connection. The Stone's HostAgent knows that there is a mid-level cache on that node, and returns the listening port for that MLC HostAgent to the Gem.
3. The Gem contacts the MLC HostAgent on the listening port returned by the Stone in Step (2). The Gem presents the certificates and private key used for the X509 login that started that Gem to the MLC HostAgent, to establish secure connection (F).
4. The MLC HostAgent starts a pgsvr thread for the Gem.
5. The MLC HostAgent passes the listening port for the mid-cache pgsvr thread to the Gem.
6. The Gem passes the listening port for the mid-cache pgsvr thread to the HostAgent on the Stone.
7. The HostAgent on the Stone's node contacts the mid-cache pgsvr thread in the MLC HostAgent, and they authenticate, establishing secure connection (G). This connection is between the MLC's HostAgent pgsvr thread and the pgsvr thread for that session in the HostAgent on the Stone's node.

The mid level cache is now connected and used by the Gem for page accesses.

You can verify that a session is using a mid-level cache by sending

```
System >> midLevelCacheAddress
```

Which returns the IP address of a mid-level cache that this session is connected to, or nil if this session is not using a mid-level cache.

Example

In this example, the topaz application is running on remote leaf node ***fiji***, logging into the Stone on ***alcatraz***. After login, it connects to the mid-level cache on ***brisbane***.

```
fiji> topaz
<startup>
topaz > set cert $MyRemoteCertDir/DataCurator.chain.pem
topaz > set key $MyRemoteCertDir/DataCurator.privkey.pem
topaz > set cacert $MyRemoteCertDir/stoneCA-devstone.cert.pem
topaz > set netldi localhost:54321
topaz > login
<login details>
topaz 1 > run
System midLevelCacheConnect: 'brisbane'
%
true
topaz 1> run
System midLevelCacheAddress
%
10.94.162.81
```

Reconnecting

When there are multiple mid-level caches connected to the Stone, you can disconnect from one mid-level cache and reconnect to a different one in a single operation, using the method:

```
System class >> midLevelCacheReconnect: hostName
```

This is similar to `midLevelCacheConnect:`, but if the current session is connected to a mid-level cache, it will be disconnected before it tries to connect to the new mid-level cache.

Chapter 6 Administration

This chapter contains additional information that may be useful in managing an X509-Secure GemStone system.

Administering an X509-Secure GemStone system includes additional responsibilities over a regular GemStone environment. The significant tasks are covered in the earlier chapters of this document. You should also be familiar with the basic administration tasks, such as garbage collection, backups, and so on, as described in *System Administration Guide*.

Topics in this chapter include:

- **Managing HostAgents** (page 51) – Getting information about HostAgents and stopping HostAgents.
- **Managing Caches** (page 53) – Remote cache timeout, getting details about remote caches, and warming remote caches.
- **Managing NetLDIs** (page 55) – Additional information on managing NetLDIs in an X509-Secured configuration.
- **Log Files** (page 55) – The location and names of log files in X509-Secured GemStone.
- **Other Administration** (page 56)– Other useful information related to administration in X509-Secure Gemstone configurations.

6.1 Managing HostAgents

Information about HostAgents

HostAgents are included in the **gslist** report on the Stone's or mid-cache node, when the **gslist -H** flag is used. **gslist** on the Stone's node reports lines of the form:

```
exists   3.5.0   gsadmin   Feb 26 16:24 hostagent   hostagent-
    gs64stone-10.95.143.15
```

Where the IP is the IP of the remote node that is being serviced by this HostAgent.

From Smalltalk, `System class >> hostAgentSessions` reports the sessions IDs of running HostAgents.

`System class >> descriptionOfSession:` for a HostAgent session ID reports 'hostagent' in slot #17, and its listening port in slot #24. The other slots provide the usual details for the HostAgent session.

To find out if the current session is using a HostAgent (and therefor is X509-secured), use

```
System class >> sessionIsUsingHostagent
```

To get a report of all HostAgent sessions, use `System >> hostAgentSessionsReport`. This produces a report with lines containing Host Agent session details, for example:

```
session   5  hostagent          servicing: 10.94.141.15 gemPid:
   9848 listeningPort: 46627
```

Stopping HostAgents

Stopping from the command line

The **stophostagent** utility will stop a HostAgent process. The command line arguments are the same as the ones used by **starthostagent**, except **-n** is not used.

For example,

```
alcatraz> stophostagent -m fiji -N 54321
   -U $MyStoneCertDir/alcatraz.chain.pem
   -R $MyStoneCertDir/alcatraz.privkey.pem
   -J $MyStoneCertDir/stoneCA-gs64stone.cert.pem
```

Stopping from within the image

`System class >> stopUserSessions` will stop secure gem sessions, but does not stop the HostAgent session itself.

HostAgents can be stopped using the following methods. The `stopHostAgent*` methods attempt a graceful shutdown, and return an error if unable to complete. The `killhostAgent*` methods terminate the HostAgent with a fatal error.

```
System class >> stopHostAgents
System class >> stopHostAgentSession: sessionId
System class >> stopHostAgentForHost: hostNameOrIp
System class >> killHostAgents
System class >> killHostAgentSession: sessionId
```

Stopping a HostAgent also stops all x509 sessions using that HostAgent.

Restarting after stopping host agent

When the HostAgent has been shut down, either by timeout or manually, you must run **startnetldi** on the remote host, then execute **starthostagent** on the Stone’s node, to reestablish the connection and allow logins to complete.

6.2 Managing Caches

Timeout of the secure remote cache

With regular GemStone, remote caches can be automatically started or 0restarted when a remote Gem logs in; but with X509-Secured GemStone, since connections are initiated from the Stone's host, the remote cache must be explicitly started (or restarted, if it has timed out and shut down), by executing utilities on the Stone's node.

If there are no Gems on the remote node, the remote shared page cache will shut down according to the STN_REMOTE_CACHE_TIMEOUT configuration (by default, 5 minutes).

This also causes the HostAgent on the stone's node to shut down.

When the HostAgent has been shut down, either by timeout or manually, you must restart the NetLDI on the remote host, then execute **starthostagent** on the Stone's node, to reestablish the connection and allow logins to complete.

Mid level caches require explicit stop

Because a mid-level HostAgent is running on the remote node, mid-level remote caches do not timeout. You must manually stop a mid-level cache when you are done with it.

When a mid level cache is connected, **stopstone** will report that users are logged in; either stop the mid level cache remote hostagents, or use **stopstone -i**.

Information on caches

The results of the methods:

```
System class >> midLevelCachesReport
System class >> remoteCachesReport
```

include cache types 'x509mid' and 'x509remote', in addition to the existing types 'mid' and 'remote'.

Warming caches on startup

You can configure cache warming on a remote X509-secured cache (leaf cache) or X509-secured mid-level cache by setting a configuration parameter in the configuration file passed with the -E argument to startnetldi.

The configuration parameter NETLDI_WARMER_ARGS can be set to a String, which provide the arguments that control out the cache will be warmed. Cache warming will be done by the page server or host agent on the newly started cache.

If the argument string is empty, no warming is done; otherwise the argument string may contain the following options:

-d include data pages. If not included, only warm object table pages.

-M *midCacheHostName*
Include this argument if you wish to warm the newly started cache from an already running mid level cache.

If -M specifies a host on which a mid-level cache is running (that is associated with the same Stone), the newly started mid-level cache will be warmed with the pages from the other mid-level cache.

If no -M argument is provided, or if the specified host resolves to localhost or the mid level cache is not found, then pages will be pulled from the stone cache to warm the newly started mid-level cache. Pages pulled from the Stone are based on the current view as of the login of the warmer session.

-n *int*
the integer number of threads. These must be in the range 1..20 for warming a mid-level cache and in the range 1..2 for warming a leaf cache. This further limits the value for SHR_PUSH_TO_MIDCACHES_THREADS.

For example,

```
NETLDI_WARMER_ARGS = '-d -M adelaide -i4';
```

When the remote cache is a mid-level cache (NETLDI_START_MIDCACHE=true), multiple pusher threads are started in either the mid cache host agent for the source of the pages, or in the HostAgent on stone host, and those threads scan the source cache pushing pages to the newly started mid cache.

When the remote cache is a leaf cache (that is, it is not a mid-level cache, so NETLDI_START_MIDCACHE=false), the number of threads is limited to 2, since a warmer session on an x509 leaf cache has only a single threaded connection to a mid or stone cache.

You must specify the NETLDI_HostAgentUser_cert and NETLDI_HostAgentUser_key configuration parameters when using cache warming.

Keeping mid-level caches warm

As changes are committed by X509-secured sessions, you can configure your system to have these changes pushed to mid-level caches, ensuring that the mid-level caches contain the latest used pages.

This feature is limited to X509-secured mid level caches and commits performed by X509-secured sessions; changes made, for example, by administrative sessions running in linked mode on the Stone's node will not get pushed to mid-level caches.

To enable, set the configuration parameter SHR_PUSH_TO_MIDCACHES_THREADS to the number of threads to use. This is normally set in the range 2-5, depending on the network bandwidth between the Stone and mid-level cache hosts.

For example

```
SHR_PUSH_TO_MIDCACHES_THREADS = 3;
```

6.3 Managing NetLDIs

Stopping certificate-only NetLDIs

Shutting down the certificate-only NetLDI on the Stone's node does not disrupt existing HostAgents for remote caches.

If the certificate-only NetLDI on a remote node is shut down, the associated HostAgent and all x509 sessions using that HostAgent will be shut down.

You must restart the NetLDI on the remote host, then execute **starthostagent** on the Stone's node, to restart the cache on the remote node.

Multiple NetLDIs

For a given Stone and host, you may have only one certificate-only NetLDI, and each Stone on a given host must have its own separate certificate-only NetLDI.

This differs from ordinary NetLDIs, for which you may have multiple NetLDIs per Stone, or one ordinary NetLDI that services multiple Stones.

You may start one or more ordinary NetLDIs in addition to one certificate-only NetLDI on a host, although this may compromise the security of Stone access.

6.4 Log Files

Processes that support X509-secured GemStone write log files with additional connection information. These can be useful to detect problems in a multi-node system. It is recommended that all log files are written to a single log directory.

Netldi default log file directory

X509-secured NetLDIs require the **-D** argument to **startnetldi**, specifying the default log file directory. This simplifies management of log files in a distributed system.

The -D argument is required both for the Stone's and the remote NetLDIs, and is used for system log files, as well as in composing Gem log file names and locations. It is not used for the NetLDI log itself; this is specified using the **-l** argument.

- The **-D** directory for the Stone's NetLDI is used for HostAgent log files.
- The **-D** directory for the remote NetLDI is used for the remote cache process log files, and is the default directory for the log files for Gems on that node. It also is the working directory for Gems on that node.

Gem logs

X509-secured Gem logs are not written in the user's home directory.

The Gem created by an X509 login is located by default in the directory supplied to the **startnetldi -D** argument, and has the default log file name `gem`*PIDnodename*`.log`.

The X509 login parameters include a field **`directory`** (Unlike regular GemStone parameters, X509 parameters do not include the Gem NRS, but instead have specific fields for supported features). This directory field overrides the -D setting in startnetldi.

If this **`directory`** field is set to a value that includes a %D, the **startnetldi -D** argument replaces the %D to compose the log and working directory.

HostAgent logs

The HostAgent log is located in the directory specified by **-D** argument to **startnetldi** on the Stone's node (or on the mid-cache node, for mid-cache HostAgents), and has the name:

`hostagent-`*StoneName***`-`***RemoteNodeName***`-`***PIDStoneNodeName***`.log`**

Other process log files

The other processes associated with the X509-secured remote cache also have different default log file names than those for ordinary remote caches, and are located in the directory specified by the **startnetldi -D** option.

`pgsvrmain`*PIDnodeName*`.log`

remoteCacheName_PID`pgsvrff_`*nodeName*`.log`

remoteCacheName_PID`pcmon_`*nodeName*`.log`

These log files are not deleted on logout.

6.5 Other Administration

Requiring UserProfiles to use X509 Authentication

You may specify that individual UserProfiles may only authenticate using X509 logins. Configuring UserProfiles in this way means that no other means of authentication is allowed, ensuring that all accesses are always certificate-based and secure.

System accounts may not be configured in this way.

Note that since X509 does not support linked sessions, users configured with this authentication will not be able to log in a linked.

The following methods set and report the status of this authentication scheme.

```
UserProfile >> enableX509Authentication
UserProfile >> authenticationSchemeIsX509
```

The following method has also been added

```
UserProfile >> x509loginStatus
```

This method creates a new UserProfile that can only log in using X509 authentication.

```
UserProfile class >> newX509WithUserId:
```

Disallowed Operations in a X509 session

Some operations may not be run in an X509 sessions. This includes operations such as `markForCollection`, `objectAudit`, `fullBackupTo:`, and similar operations. These operations will return an error including the phrase. "not allowed when gem is remote with an X509 login".

To run a multi-threaded scan operation in a secure environment, login linked on the Stone's node.

Chapter

A X509-related Utilities and Configurations

A.1 Configuration Parameters specific to X509-Secured GemStone

This section includes configuration parameters that are specific to X509-Secured GemStone. Refer to the *System Administration Guide* for information on additional configuration parameters that apply to all GemStone systems.

NetLDI configuration Parameters

The following configuration parameters only apply to X509 NetLDIs. They are used in a configuration file passed into the **startnetldi -E** argument.

NETLDI_PORT_RANGE

Specifies the range of port numbers, which will be used for listening sockets for remote X509 Gems during login. The two elements must be in the port range 1..65535, and the second element must be greater than the first.

For example:

```
NETLDI_PORT_RANGE = 50000, 50020;
```

If this is not set, or set with the pair of values 10000,65535, random ports in the range used by ephemeral ports will be used. On Linux the ephemeral port range is in `/proc/sys/net/ipv4/ip_local_port_range`.

NETLDI_START_MIDCACHE

This should be set to true only when starting an X509 secured mid-level cache node.

When set to true, the startnetldi processing, after starting the shared cache on the mid-cache node, then starts a HostAgent process on the mid-cache node, which will log into the Stone's HostAgent.

NETLDI_HostAgentUser_cert

Required when starting an X509 secured mid-level cache node, or when doing cache warming. The path and name of the cert file for HostAgentUser, `HostAgentUser.chain.pem`.

NETLDI_HostAgentUser_key

Required when starting an X509 secured mid-level cache node, or when doing cache warming. The path and name of the private key file for HostAgentUser, `HostAgentUser.privkey.pem`.

NETLDI_WARMER_ARGS

Enables warming a remote X509 leaf or mid-level cache, either from the Stone's cache or from another mid-level cache. For details, see the description on page 53.

SHR_PUSH_TO_MIDCACHES_THREADS

Enables pushing of pages committed by X509-secured Gems to a mid-level cache. Specified as the number of threads to do the pushing. The allowed range is 0..20. This is normally set in the range 2-5, depending on the network bandwidth between the Stone and mid-level cache hosts. For details, see the description on page 54.

Configuration parameters used for x509 remote caches

In addition to the x509-specific configuration parameters listed above, the following configuration parameters, if set in the configuration file provided with the **startnetldi -E** argument, are used to configure how the remote cache is started:

GEM_STATMONITOR_ARGS
GEM_STATMONITOR_MID_CACHE_ARGS
SHR_NUM_FREE_FRAME_SERVERS
SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_POLICY
SHR_PAGE_CACHE_LARGE_MEMORY_PAGE_SIZE_MB
SHR_PAGE_CACHE_LOCKED
SHR_PAGE_CACHE_NUM_PROCS
SHR_PAGE_CACHE_NUM_SHARED_COUNTERS
SHR_PAGE_CACHE_PERMISSIONS
SHR_PAGE_CACHE_SIZE
SHR_PUSH_TO_MIDCACHES_THREADS
SHR_SPIN_LOCK_COUNT
SHR_TARGET_FREE_FRAME_COUNT
SHR_WELL_KNOWN_PORT_NUMBER

Gem Configuration Parameters

GemRemoteCommit

If TRUE, a gem on a remote gem will execute the critical region of commit in the session's thread in the pgsvr or HostAgent on stone host. This avoids latency and decreases network traffic between the Stone and Gem hosts. Can only be enabled, and is the default, if gem and stone host have same byte order.

Runtime name: #GemRemoteCommit
Default: true (for remote X509 Gem on a host with the same byte order as stone host.

Other parameters with specific behavior in X509-secured processes

GEM_PGSVR_COMPRESS_PAGE_TRANSFERS is always true, regardless of the configuration parameter setting, for X509-secured Gems.

GEM_PGSVR_USE_SSL is always true for X509-secured sessions.

When a remote cache is started by a X509-secured NetLDI, the default computation for SHR_PAGE_CACHE_NUM_PROCS on Linux is half of the smaller of the maximum open files or the maximum size of a semaphore array. On other platforms the default is 256.

A.2 Utility details for X509

This section includes additional features of utilities that are specific to X509-Secured GemStone. Refer to the *System Administration Guide* for information on these utilities, and other utilities that are required to manage a GemStone installation.

gslist

gslist (without -v) reporting on processes the local machine uses only the lock file. Since it does not need to connect to any NetLDI, it treats X509-secured processes no differently than regular GemStone processes.

On the local machine, **gslist -v** option attempts to connect to a process to verify status. For X509-secured processes, **gslist -v** reports "OK" if **gslist** gets a response indicating that an SSL handshake is requested. It does not complete the handshake, and does not require credentials.

gslist for remote nodes

When using **gslist -m** to request the status of processes on a different node, **gslist** must have credentials that allow it to connect to the NetLDI on the other node. The following arguments are required to gslist:

-J Specify an X.509 CA certificate file.

-R Specify a private key file.

-U Specify an X.509 certificate file.

For example, on the host ***remote_host***, to query for the GemStone processes running on the Stone's node ***stn_host***, with the certificate-only NetLDI at port 54321:

```
unix> gslist -m stn_host -v -N 54321
   -U $MyCertDir/remote_host.chain.pem
   -R $MyCertDir/remote_host.privkey.pem
   -J $MyCertDir/stoneCA-gs64stone.cert.pem
```

HostAgent information

HostAgents are included in the **gslist** report on the Stone's or mid-cache node only when the **-H** option is used. This avoids an excessive amount of information if there a very large number of HostAgents for remote nodes.

When the **-H** option to **gslist** is included, **gslist** on the Stone's node includes lines of the form:

```
exists   3.5.0   gsadmin   Feb 26 12:24 hostagent   hostagent-
   gs64stone-10.95.143.15
```

Mid-level cache HostAgents are:

```
exists   3.5.0   gsadmin   Feb 26 16:49 hostagent   hostagent-
   gs64stone-midcache-remote_host
```

gslist -x reports detailed information on the HostAgent only if the **-H** flag is also included.

starthostagent

Once the Stone and the remote NetLDI are started, you execute **starthostagent** on the Stone's node. This initiate the steps to authenticate with the remote node, start the HostAgent, and start the remote shared page cache.

starthostagent requires the following arguments:

-J *CACertFilePath*
Specifies a certificate authority certificate (CA) in PEM format.

-m *remoteNodeNameOrIP*
The name or IP address of the remote node for which the HostAgent is to be started. A startnetldi -E must have been executed on that remote node.

-N *stoneNetLDInameOrPort*
The name or port of the NetLDI running on the Stone's node (the node this script is executing on).

-n *remoteNetLDInameOrPort*
The name or port of the NetLDI on the remote node *remoteNodeNameOrIP*. This must have been started using the **startnetldi -E** (along with other appropriate arguments).

-R *privateKeyFilePath*
Specifies the host private key chain certificate (for the host named *remoteNodeNameOrIP*), in PEM format.

-U *publicKeyFilePath*
Specifies host public key chain certificate (for the host named *remoteNodeNameOrIP*) in PEM format.

Note that there is no argument to pass in the name of the Stone; the Stone name is determined from the certificate file *CACertFilePath,* which is passed in with the **-J** argument.

starthostagent also accepts **-h** to print help information, and **-V** to print version information.

startnetldi

The X509-secure NetLDI process has a number of different behaviors and requirements than an ordinary NetLDI. X509-secured NetLDIs only work with X509 remote caches and Gems, and do not support ordinary caches and Gems, and vice versa.

The X509-secured NetLDI on the Stone's node and the one on the remote node have quite different responsibilities; the Stone's NetLDI is responsible for starting the HostAgents for remote nodes, and the remote NetLDI is responsible for starting the remote cache and remote Gems.

To start a Netldi using **startnetldi**, use the -S argument to specify an X509-Secured Netldi, and include the arguments that provide the X509 credentials.

Using **startnetldi** with the **-S** argument also requires that you include the **-D** argument, which provides the default log directory for the processes started during X509 logins.

The following are the **startnetldi** arguments that specifically support certificate-only mode:

-E *configFileName*
: For use in secure certificate-only mode for a remote NetLDI, **not** for the Stone's NetLDI. This enables startup of a remote shared page cache or a mid-level cache on this node.

  The specified configuration file includes parameters that define settings for the shared page cache. For mid-level caches, it includes parameters for the mid-level cache.

-J *path*
: Specifies a certificate authority certificate (CA) in PEM format to use. Requires -S.

-L *path*
: Specifies a certificate revocation list (CRL) file in PEM format.Used on the Stone's NetLDI, not with remote NetLDIs. Requires -S.

-R *path*
: Specifies the host private key in PEM format to use. Requires -S.

-S
: start NetLDI in secure certificate mode; must include -D, -J, -R, and -U, and on the remote node, also -E.

-U *path*
: Specifies the host X.509 certificate in PEM format to use. Requires -S.

startnetldi has a number of other command line options, which are required, such as the **-D** argument to specify log file locations. Refer to the *System Administration Guide* for details, or see **startnetldi -h** output.

stophostagent

Stopping a HostAgent requires the same arguments as starting a HostAgent, except the **-n** is not used. The arguments are:

-J *CACertFilePath*
: Specifies a certificate authority certificate (CA) in PEM format.

-m *remoteNodeNameOrIP*
: The name or IP address of the remote node that the HostAgent is servicing.

-N *stoneNetLDInameOrPort*
: The name or port of the NetLDI running on the Stone's node (the node this script is executing on).

-R *privateKeyFilePath*
: Specifies the host private key chain certificate (for the host named *remoteNodeNameOrIP*), in PEM format.

-U *publicKeyFilePath*
: Specifies host public key chain certificate (for the host named *remoteNodeNameOrIP*) in PEM format.

stophostagent also accepts **-h** to print help information, and **-V** to print version information.